IM&F시리즈 17

금융공학 VII

Scientific Computing for Finance and Economics

최병선 지음

김구재단

[표지사진]
Havasu Falls@Havasupai Indian Reservation
2015. 12. 08.
F6.3, 1/40sec, ISO-100
Canon EOS 5D Mark II with Canon EF 28-300mm F/3.5-5.6

머릿글

본서를 읽는 독자라면 이미 최근 경제학이나 재무학에서 컴퓨터를 효율적으로 다루는 능력이 얼마나 중요한지 잘 알고 있으리라 생각한다. 이를 가름해보기 위해서 월가 최고 수준의 회사에서 사람을 채용하는 가이드라인을 살펴보자.

Renaissance Technologies Website wrote: Renaissance Technologies is a quantitative investment management company trading in global financial markets, dedicated to producing exceptional returns for its investors by strictly adhering to mathematical and statistical methods. Our research scientists use machine-learning, applied mathematics, and techniques from modern statistics to develop and refine models of the financial markets and to develop trading algorithms based on those models. An ideal candidate for research scientist will have a degree in computer science, mathematics, physics, statistics, or a related discipline a demonstrated capacity to do first-class research strong computer programming skills. Experience in finance is not required.

The D. E. Shaw Group Website wrote: The D. E. Shaw group offers opportunities for individuals of all backgrounds and levels of experience who display evidence of outstanding ability and have a track record of exceptional accomplishment. We hire gifted mathematicians and computer scientists as well as extraordinary generalists with liberal arts or other non-technical backgrounds. (Some of the firm's managing directors hold degrees in fields as disparate as English, psychology, and Russian studies.) Many of our employees have little knowledge of finance when they join the firm, but all share uncommon intelligence, analytical ability, and drive.

Jane Street Website wrote: Jane Street recurites a people with the following personalities: Honesty, both personal and intellectual; The ability to work in a team while also being self-motivated on specific tasks; Intellectual curiosity and the passion to find and build the tools to answer questions more efficiently and effectively; The ability and desire to learn from mistakes; Comfort with thinking logically in a quantitative and probabilistic way; and A drive to improve and achieve.

Two Sigma Website wrote: We hire students across all degree levels, with varied backgrounds. While we analyze the data-rich financial markets, prior experience in the financial industry is not a requirement. In fact, more than

> 70 percents of our employees come from outside of finance. We're looking for intellectual curiosity, passion for science and technology, and a desire to join a dynamic and talented team. Learn more about how we hire and what makes Two Sigma a career destination for top talent.

본저자는 앞으로 경제학과 재무학의 특성상 이를 전공하는 학생들이나 학자들이 컴퓨터에 의존하는 정도는 다른 어느 학문 분야에 못지 않을 것이라고 생각한다. 본서는 이러한 시대에 적응하기 위해서 경제학이나 재무학을 전공하는 학생들과 실무자들이 알아야 할 과학적컴퓨팅(scientific computing)의 기초를 다루고 있다. 본서에서 사용한 컴퓨터언어는 Python과 MATLAB이다. 본저자의 생각으로는 결국 Python이 보편적으로 사용될 것이다. 따라서 새로이 컴퓨터언어를 배우고자 하는 독자는 Python을 배울 것을 권한다. 본서에서는 컴퓨터언어 Julia를 사용하지 않았다. 아직은 본저자가 Julia를 사용할 줄 모르는 것이 주된 이유이지만, 본서에서 작성된 프로그램은 Julia를 사용할 만큼 길지 않기 때문이다. Julia는 Python이나 R보다 훨씬 빨라서 C++과 비슷한 속도의 처리능력을 갖는다고 한다. 이미 재무학이나 경제학에서 나타나는 문제, 특히 기계학습(machine learning)으로 문제를 푸는데 Julia가 많이 사용되고 있다. 예를 들어, New York FED에서는 Julia를 사용해서 FRBNY DSGE모형을 프로그램하였다. 앞으로 경제학이나 재무학을 깊이 공부할 학생은 Julia를 배우는 것이 좋을 것이라 생각된다.

본서를 만드는 데 많은 사람들의 도움이 있었다. 우선 박사과정을 이수하는 바쁜 중임에도 불구하고 교정을 보아준 수학과 김원세박사, 경영학과 김민직군, 경제학부 김찬수군과 서영선군, 산업공학과 이민혁박사에게 감사한다. 또한 2018년도 1학기 수리금융경제학특수연구를 수강하면서 교정에 참여해준 학생들에게 감사한다. 본서는 LaTeX으로 작성한 것으로, 김찬수군이 LaTeX스타일파일과 표지를 만들었다. [1]

버킷리스트 속에 깊이 숨겨 놓았던
Camino de Santiago 순례를 마치고 ...

최병선
2018년 3월 15일

[1] 현대문화사(hun1249@hanmail.net)에서 본서의 POD판을 구하실 수 있습니다.

차 례

제 1 장

서론

제1.1절 과학적 컴퓨팅

초등수학에서 다루는 방정식은 방정식 자체에 약간 변형을 가함으로써 해석해(analytic solution)를 구할 수 있다. 그러나, 현실에서 나타나는 문제를 수리적으로 해결하고자 할 때 해석해를 구할 수 있는 경우는 그리 많지 않다. 실제 상황을 잘 표현하는 수학적 모형을 구축했다해도, 그 모형 하에서 해석해를 구할 수 없는 경우가 많다. 이러한 경우, 우리가 사용할 수 있는 방법이 수치계산법(numerical anlaysis method)이다. 수치계산법은 지금으로부터 오천년 전 Sumer인이 2차방정식을 풀기 위해서 사용한 이래, 인간이 방정식을 풀기 위해서 사용하는 보편적 방법이다. 우리가 유의해야 할 점은 수치계산기법들은 정밀도(precision)가 뛰어나다는 것이다. 예를 들어, 방정식의 해를 구하는데 적용되는 Newton-Raphson법이나 Gauss구적법(Gaussian quadrature)의 정밀성은 놀랄만하다. 본격적으로 컴퓨터가 등장하기 전에는 가능한 한 적은 수고로 원하는 정밀도를 달성하려는 연구가 널리 행해졌다. 본저자는 학부시절 Cecil Hastings교수 [29]가 제시한 많은 근사식들에 매료된 적이 있었다. 그렇게 정밀한 근사식들을 어떻게 유도할 수 있었을까? 컴퓨터가 발전한 오늘날에는 해석해를 구할 수 없는 경우뿐 아니라 해석해를 구할 수 있는 경우에도 수치계산기법들은 실제 문제를 푸는 강력한 도구이다. 예를 들어, 재무론에서 가장 중요한 문제들 중 하나인 금융파생상품의 가치평가에서도 Black-Scholes환경에서와 같은 특수한 경우에는 해석해를 구할 수 있다. 그러나, 금융파생상품의 가치평가문제에서 해석해를 구할 수 있는 경우는 많지 않으며, 이러한 경우에는 수치계산기법을 사용해서 경계값문제(boundary value problem)를 풀어야 한다. 설사 이론적으로 그 경계값문제의 해석해를 구할 수 있다하더라도 그 해석해를 계산하는 자체가 복잡하다면, 수치계산기법을 사용해서 경계값문제 자체를 푸는 것이 더 좋을 수도

있다.

본서의 목적은 경제학이나 재무학에 나오는 수리적 모형(mathematical model)을 컴퓨터를 사용해서 분석하는데 필요한 기본적 지식을 다루는 것이다. 먼 미래에는 어떻게 될지 모르겠지만 적어도 현시점에서 우리가 다루는 수학적 모형의 대부분은 연속함수(continuous function)이거나 구분적연속함수(piecewise continuous function)로 나타낼 수 있다. 컴퓨터를 사용해서 이러한 모형을 다루려면, 연속함수를 이산함수(discrete function)로 근사시켜야 한다. 이러한 이산화과정을 효율적이고(efficiently), 정확하게(accurately), 그리고 신뢰할 수 있게(reliably) 푸는 알고리즘을 찾는 것이 과학적 컴퓨팅(scientific computing)의 핵심이다. 수치해석(numerical analysis)은 이러한 알고리즘을 연구하는 이론적 배경이다. 그러나, 과학적 컴퓨팅은 단순히 수리적으로 뛰어난 알고리즘을 다루는 것은 아니다. 수리적으로 뛰어난 알고리즘이라도 이를 효율적으로 실장할(implementing) 수 없다면, 실제 문제를 푸는데 큰 도움이 되지 않는다. 풀고자 하는 문제와 배경이 되는 지식, 프로그래밍언어(programming language), 데이터구조(data structure), 컴퓨터아키텍쳐(computer architecture) 등에 대한 이해가 깊을수록 효율적으로 실장할 수 있는 좋은 알고리즘을 만들 수 있는 가능성이 커질 것이다. 과학적 컴퓨팅이란 실장을 고려한 뛰어난 알고리즘을 찾아내어 수치적으로 문제를 해결하는 것이다.

본서의 원고를 작성할 때 가장 많이 참고한 문헌들은 Judd [31], Ascher & Grief [11], Golub & Van Loan [23], 그리고 *Handbook of Computational Economics* 시리즈 [8], [53], [60]이다.

제1.2절 함수의 전개와 근사

이 절에서는 과학적 컴퓨팅의 기초가 되는 함수의 전개와 근사에 대해서 간단히 살펴보자.

1.2.1 Landau의 리틀오우와 빅오우

함수의 전개나 근사식을 논의할 때는 근사의 정도가 문제된다. 이 소절에서는 오차를 나타내는데 중요한 역할을 하는 무한소 차수와 무한대 차수에 대해 살펴보자.

정의 1.2.1: 무한소와 무한대

만약 함수 $f(x)$ 가 식 $\lim_{x\to a} f(x) = 0$를 만족하면, 함수 $f(x)$는 $x = a$에서 무한소라 한다.
만약 함수 $f(x)$가 식 $\lim_{x\to a} f(x) = \infty$를 만족하면, 함수 $f(x)$는 $x = a$에서 무한대라 한다.

여기서 a는 ∞ 또는 $-\infty$ 일 수도 있다.

정의 1.2.2: Landau의 오우

무한소 또는 무한대인 함수들의 관계를 다음과 같이 표기한다.

(a) 만약 $x = a$에서 무한소인 함수들 $f(x)$와 $g(x)$에 대해서 다음 식이 성립하면, $f(x)$는 $g(x)$보다 높은 차수의 무한소라 하고, $f(x) = o(g(x))$ 또는 $f(x) = o(g(x))(x \to a)$로 표기한다.

$$\lim_{x \to a} \frac{f(x)}{g(x)} = 0$$

이 o를 Landau의 리틀오우(little-o)라 한다.

(b) 만약 $x = a$에서 무한소인 함수들 $f(x)$와 $g(x)$에 대해서 다음 식이 성립하면, $f(x)$는 $g(x)$와 같은 차수의 무한소라 하고, $f(x) = O(g(x))$ 또는 $f(x) = O(g(x))(x \to a)$로 표기한다.

$$\lim_{x \to a} \frac{f(x)}{g(x)} = c$$

여기서 c는 0이 아닌 유한인(finite) 상수이다. 이 O를 Landau의 빅오우(big-O)라 한다.

(c) 만약 $x = a$에서 무한대인 함수들 $f(x)$와 $g(x)$에 대해서 다음 식이 성립하면, $f(x)$는 $g(x)$보다 낮은 차수의 무한대라 하고, $f(x) = o(g(x))$ 또는 $f(x) = o(g(x))(x \to a)$로 표기한다. 이 o를 Landau의 리틀오우라 한다.

$$\lim_{x \to a} \frac{f(x)}{g(x)} = 0$$

(d) 만약 $x = a$에서 무한대인 함수들 $f(x)$와 $g(x)$에 대해서 다음 식이 성립하면, $f(x)$는 $g(x)$와 같은 차수의 무한대라 하고, $f(x) = O(g(x))$ 또는 $f(x) = O(g(x))(x \to a)$로 표기한다.

$$\lim_{x \to a} \frac{f(x)}{g(x)} = c$$

여기서 c는 0인 아닌 유한인 상수이다. 이 O를 Landau의 빅오우라 한다.

예제 1.2.1 극한 $\lim_{x \to 0} \sin x = 0$이 성립하므로, $\sin x$는 극한 $x \to 0$에서 무한소이다. 또한,

다음 식이 성립한다.

$$\lim_{x\to 0}\frac{\sin x}{x}=1 \tag{1}$$

따라서, 식 $\sin x = O(x)(x\to 0)$가 성립한다.

극한 $\lim_{x\to 0}\cos x = 1$이 성립하므로, 함수 $1-\cos x$는 극한 $x\to 0$에서 무한소이다. 다음 식들이 성립한다.

$$\lim_{x\to 0}\frac{1-\cos x}{x}=0,\quad \lim_{x\to 0}\frac{1-\cos x}{x^2}=\frac{1}{2} \tag{2}$$

즉, 식 $1-\cos x = o(x)$와 식 $1-\cos x = O(x^2)$가 성립한다. ■

예제 1.2.2 극한 $\lim_{x\to\infty}\ln x = \infty$이 성립하므로, $\ln x$는 극한 $x\to\infty$에서 무한대이다. 임의의 실수 α에 대해서, 다음 식이 성립한다.

$$\lim_{x\to\infty}\frac{\ln x}{x^\alpha}=0 \tag{1}$$

따라서, 각 $\alpha(>0)$ 에 대해서 x^α는 $\ln x$보다 높은 차수의 무한대이다. 그 이유는 로그함수가 ∞로 발산하는 속도는 아주 늦기 때문이다.

극한 $\lim_{x\to\infty} e^x = \infty$이 성립하므로, e^x는 극한 $x\to\infty$에서 무한대이다. 임의의 양수 α에 대해서, 다음 식이 성립한다.

$$\lim_{x\to\infty}\frac{x^\alpha}{e^x}=0 \tag{2}$$

따라서, 각 $\alpha(>0)$ 에 대해서 x^α는 e^x보다 낮은 차수의 무한대이다. 그 이유는 지수함수 e^x는 ∞로 발산하는 속도가 아주 빠르기 때문이다. ■

명제 1.2.1

만약 $f(x)=O(x^m)(x\to a)$이고 $g(x)=O(x^n)(x\to a)$이며 또한 $m<n$이면, 다음 명제들이 성립한다.

(a) 만약 식 $\lim_{x\to a} f(x)=0$과 식 $\lim_{x\to a} g(x)=0$이 성립하면, 식 $f(x)+g(x)=O(x^m)(x\to a)$가 성립한다.

(b) 만약 식 $\lim_{x\to a} f(x)=\infty$와 식 $\lim_{x\to a} g(x)=\infty$가 성립하면, 식 $f(x)+g(x)=$

$O(x^n)\,(x \to a)$가 성립한다.

(c) 식 $f(x)g(x) = O(x^{m+n})\,(x \to a)$가 성립한다.

다음 예제는 MATLAB을 사용해서 Landau의 리틀오우과 빅오우를 구하는 예를 보여준다.

예제 1.2.3 MATLAB의 Symbolic Math Toolbox에 포함된 MATLAB함수 limit.m을 사용해서 극한을 구하기 위해서, 다음 MATLAB프로그램 LittleOBigO.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename LittleOBigO.m
% Landau's Big-O and little-o
% Programmed by CBS
% ------------------------------------------------------------------
syms x p c
p1 = limit((1+x)^100/(x^99+2),x,inf)
p2 = limit((1+x)^100/(x^100+1),x,inf)
p3 = limit((1+x)^100/(x^101+3),x,inf)
s1 = limit((sin(x) - x + 1/6*x^3)/x^4,x,0)
s2 = limit((sin(x) - x + 1/6*x^3)/x^5,x,0)
s3 = limit((sin(x) - x + 1/6*x^3)/x^6,x,0)
f = (exp(x) - (1 + x + x^2/2 + x^3/6 + x^4/24))/x^p
f1 = limit(subs(f,p,4),x,0)
f2 = limit(subs(f,p,5),x,0)
f3 = limit(subs(f,p,6),x,0)
g1 = limit(tan(x)/x^100,x,pi/2,'left')
g2 = limit(tan(x)/x^100,x,pi/2,'right')
h = abs(x)/x
h1 = limit(h,x,0,'left')
h2 = limit(h,x,0,'right')
% End of program
% ------------------------------------------------------------------
```

첫째, 다음 극한값들이 출력된다.

$$\lim_{x\to\infty} \frac{[1+x]^{100}}{x^{99}+2} = \infty, \quad \lim_{x\to\infty} \frac{[1+x]^{100}}{x^{100}+1} = 1, \quad \lim_{x\to\infty} \frac{[1+x]^{100}}{x^{101}+3} = 0 \tag{1}$$

따라서, 다음 식이 성립한다.

$$[1+x]^{100} = O(x^{100}+1) \tag{2}$$

둘째, 다음 극한값이 출력된다.

$$\lim_{x\to 0} \frac{\sin x - x - \frac{1}{6}x^3}{x^5} = \frac{1}{120} \tag{3}$$

따라서, 다음 식이 성립한다.

$$\sin x - x - \frac{1}{6}x^3 = O(x^5) \tag{4}$$

셋째, 다음 극한값이 출력된다.

$$\lim_{x\to 0} \frac{e^x - 1 - x - \frac{1}{2!}x^2 - \frac{1}{3!}x^3 - \frac{1}{4!}x^4}{x^5} = \frac{1}{120} \tag{5}$$

따라서, 다음 식이 성립한다.

$$e^x - 1 - x - \frac{1}{2!}x^2 - \frac{1}{3!}x^3 - \frac{1}{4!}x^4 = O(x^5) \tag{6}$$

넷째, MATLAB 함수 limit.m을 사용해서 좌극한값과 우극한값을 구할 수 있다. 다음 극한값들이 출력된다.

$$\lim_{x\to \pi/2-} \frac{\tan x}{x^{100}} = \infty, \qquad \lim_{x\to \pi/2+} \frac{\tan x}{x^{100}} = -\infty \tag{7}$$

즉, 함수 $|\tan x/x^{100}|$은 극한 $x \to \pi/2$에서 무한대이다.

다섯째, 다음 극한값들이 출력된다.

$$\lim_{x\to 0-} \frac{|x|}{x} = -1, \qquad \lim_{x\to 0+} \frac{|x|}{x} = 1 \tag{8}$$

즉, $|x|$는 $x = 0$에서 Landau의 빅오우를 갖지 못한다. ■

예제 1.2.4 Python을 사용해서 예제 1.2.3을 다시 다루기 위해서, 다음 Python 프로그램 LittleOBigO.Py를 실행해 보자.

```
"""
Created on Wed Jan 11 22:13:11 2017
@author: CBS
"""

from sympy import *
x = Symbol('x')

p1 = limit((1+x)**100/(x**99+2),x,oo)
print('p1 = ', p1)
p2 = limit((1+x)**100/(x**100+1),x,oo)
print('p2 = ', p2)
```

```
p3 = limit((1+x)**100/(x**101+3),x,oo)
print('p3 = ', p3)

s1 = limit( (sin(x)-x+1/6*x**3)/x**4, x, 0 )
print('s1 = ', s1)
s2 = limit( (sin(x)-x+1/6*x**3)/x**5, x, 0 )
print('s2 = ', s2)
s3 = limit( (sin(x)-x+1/6*x**3)/x**6, x, 0 )
print('s3 = ', s3)

p = Symbol('p')
f = ( exp(x)-(1+x+x**2/2+x**3/6+x**4/24 ) )/x**p
f1 = limit(f.subs(p, 4),x,0)
print('f1 = ', f1)
f2 = limit(f.subs(p, 5),x,0)
print('f2 = ', f2)
f3 = limit(f.subs(p, 6),x,0)
print('f3 = ', f3)

g1 = limit(tan(x)/x**100, x, pi/2, dir='+')
print('g1 = ', g1)
g2 = limit(tan(x)/x**100, x, pi/2, dir='-')
print('g2 = ', g2)

h = abs(x)/x
h1 = limit(h, x, 0, dir='+')
print('h1 = ', h1)
h2 = limit(h, x, 0, dir='-')
print('h2 = ', h2)

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.3의 결과와 같다. ■

1.2.2 Taylor근사

점 x_0에서 함수 $f(x)$에 관한 정보, 즉 $f(x_0)$, $f'(x_0)$, $f''(x_0)$ 등이 주어졌을때, 이 정보를 바탕으로 함수 $f(x)$의 점 x_0 주변에서 근사식을 구하는 방법을 국소근사법(local approximation method)이라 한다. 이러한 국소근사(local approximate)가 전체 구간에서 좋은 근사가 되는 경우도 있다. 대표적인 국소근사법으로는 Taylor근사와 Padé근사가 있다. 이 소절에서는 Taylor근사에 대해서 살펴보고, 다음 소절에서 Padé근사를 살펴보자.

정리 1.2.1: Taylor정리

함수 $f(x)$의 도함수들 $f'(x)$, $f''(x)$, $\cdots$, $f^{(n)}(x)$가 폐구간 $[a,b]$에서 연속이라고 가정하

고, 다음과 같이 R_{n+1}을 정의하자.

$$R_{n+1} \doteq f(b) - \left\{ f(a) + \frac{f'(a)}{1!}[b-a] + \frac{f''(a)}{2!}[b-a]^2 + \cdots + \frac{f^{(n)}(a)}{n!}[b-a]^n \right\}$$

이 R_{n+1}을 Lagrange잉여항 (Lagrange's residual term)이라 부른다. 만약 개구간 (a,b)에서 $f^{(n+1)}(x)$가 존재한다면, 개구간 (a,b)에서 다음 식을 만족하는 상수 c가 존재한다.

$$R_{n+1} = \frac{f^{(n+1)}(c)}{(n+1)!}[b-a]^{(n+1)}$$

Taylor정리는 점 $x=b$에서 함수값 $f(b)$의 전개를 제공하지만, 점 b를 임의로 선택함으로써 함수 $f(x)$ 자체의 전개를 제공한다고 할 수 있다. 만약 $n \to \infty$인 경우에 잉여항 R_{n+1}이 0에 수렴하면, 함수 $f(x)$의 멱함수 전개를 얻는다. 즉, 점 $x=a$에서 함수 $f(x)$의 Taylor 식을 다음과 같이 쓸 수 있다.

$$f(x) = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(a)[x-a]^k + O\left(|x-a|^{n+1}\right) \tag{1.2.1}$$

식 (1.2.1)의 우변의 첫 번째 항을 함수 $f(x)$의 n차 Taylor근사식이라 한다. 만약 함수 $f(x)$가 C^∞급이면, 다음과 같이 점 $x=a$ 주변에서 함수 $f(x)$를 무한차원 멱함수로 나타낼 수 있다.

$$f(x) = \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(a)[x-a]^n \tag{1.2.2}$$

즉, 다음 정리가 성립한다.

정리 1.2.2: Taylor전개

함수 $f(x)$가 점 $x=a$를 포함하는 개구간 I에 있어서 C^∞급이고, 개구간 I의 각 점에서 다음 식이 성립한다고 가정하자.

$$\lim_{n\to\infty} R_{n+1} = 0$$

이러한 조건 하에서, 각 점 $x(\in I)$에서 함수 $f(x)$를 다음과 같이 Taylor전개할 수 있다.

$$f(x) = \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(a)[x-a]^n$$

특히, 만약 $a = 0$이면, 각 점 $x(\in I)$에서 함수 $f(x)$를 다음과 같이 Maclaurin전개할 수 있다.

$$f(x) = \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(0) x^n$$

Taylor전개는 점 $x = a$ 주변에서 함수 $f(x)$의 움직임 나타내는 것이지만, 함수에 따라서는 점 $x = a$ 주변 뿐 아니라 대역적(global) 움직임도 기술할 수 있다. 만약 멱급수 $\sum_{n=0}^{\infty} a_n x^n$이 점 $x = a$에서 수렴한다면, 조건 $|x| < |a|$를 만족하는 임의의 점 x에 대해서 이 멱급수는 절대수렴한다.

정의 1.2.3: Landau의 오우

수렴반경(radius of convergence) r을 다음과 같이 정의한다. .

$$r \doteq \sup \left\{ |x| \;\middle|\; \left| \sum_{n=0}^{\infty} a_n x^n \right| < \infty \right\} \tag{1.2.3}$$

또한, $\{x \mid |x| < r\}$을 수렴영역(region of convergence: ROC)이라 부른다.

즉, 만약 $|x| < r$이면, 멱급수 $\sum_{n=0}^{\infty} a_n x^n$은 수렴한다. 반면에, 만약 $|x| > r$이면, 멱급수 $\sum_{n=0}^{\infty} a_n x^n$은 발산한다. 자주 사용하는 Maclaurin전개는 다음과 같다.

따름정리 1.2.1: Maclaurin전개

(a) $\frac{1}{a-x} = \sum_{n=0}^{\infty} \frac{x^n}{a^{n+1}} = \frac{1}{a} + \frac{x}{a^2} + \frac{x^2}{a^3} + \frac{x^3}{a^4} + \cdots, \quad (|x| < |a|)$

(b) $\frac{1}{a+x} = \sum_{n=0}^{\infty} [-1]^n \frac{x^n}{a^{n+1}} = \frac{1}{a} - \frac{x}{a^2} + \frac{x^2}{a^3} - \frac{x^3}{a^4} + \cdots, \quad (|x| < |a|)$

(c) $[1+x]^q = 1 + qx + \frac{q[q-1]}{2!} + \frac{q[q-1][q-2]}{3!} + \cdots, \quad (|x| < 1)$

(d) $e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots, \quad (|x| < \infty)$

(e) $a^x = \sum_{n=0}^{\infty} \frac{[\ln a]^n x^n}{n!} = 1 + \frac{[\ln a]x}{1!} + \frac{[\ln a]^2 x^2}{2!} + \cdots, \quad (|x| < \infty)$

(f) $\ln(1+x) = \sum_{n=1}^{\infty} [-1]^{n-1} \frac{x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots, \quad (-1 < x \leq 1)$

(g) $\ln(1-x) = -\sum_{n=1}^{\infty} \frac{x^n}{n} = -\left[x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4} + \cdots\right], \quad (-1 < x \le 1)$

(h) $\ln\frac{1+x}{1-x} = 2\sum_{n=1}^{\infty} \frac{x^{2n-1}}{2n-1} = 2\left[x + \frac{x^3}{3} + \frac{x^5}{5} + \cdots\right], \quad (-1 < x < 1)$

(i) $\sin x = \sum_{n=0}^{\infty} [-1]^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, \quad (|x| < \infty)$

(j) $\cos x = \sum_{n=0}^{\infty} [-1]^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots, \quad (|x| < \infty)$

(k) $\sinh x = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!} = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots, \quad (|x| < \infty)$

(l) $\cosh x = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \cdots, \quad (|x| < \infty)$

따름정리 1.2.1의 급수 (c)의 수렴성은 다음과 같다. 만약 q가 0이거나 자연수가 아니면, 이 급수는 영역 $|x| < 1$에서 수렴하고, 또한 영역 $|x| > 1$에서 발산한다. 만약 $q > -1$이면, 점 $x = 1$에서 이 급수는 수렴한다. 만약 $q > 0$이면, 점 $x = 1$에서 이 급수는 절대수렴한다. 만약 $q \le -1$이면, 점 $x = 1$에서 이 급수는 발산한다. 만약 $q > 0$이면, 점 $x = -1$에서 이 급수는 절대수렴한다. 만약 $q < 0$이면, 점 $x = -1$에서 이 급수는 발산한다. 만약 q가 자연수 n이면, 이 급수는 다음과 같은 이항정리가 된다.

$$[1+x]^n = \sum_{k=0}^{n} \binom{n}{k} x^k \tag{1.2.4}$$

어떤 무한급수의 수렴성이나 수렴속도를 조사할 때, 감마함수는 중요한 역할을 한다. 적당히 큰 양수에 대해서도 감마함수값은 아주 크다. 다음 예제는 감마함수를 편하게 다룰 수 있는 형태를 갖는 함수로 근사시키는 Stirling근사식에 관한 것이다.

예제 1.2.5 음의 정수가 아닌 실수 α에 대해서 다음 함수를 정의하자.

$$\Gamma(\alpha) \doteq \int_0^{\infty} x^{a-1} e^{-x} dx \tag{1}$$

이 $\Gamma(\cdot)$를 감마함수(gamma function)라 부른다. 만약 $\alpha = 1$이면, 다음 식들이 성립한다.

$$\Gamma(1) = \int_0^{\infty} x^{1-1} e^{-x} dx = \int_0^{\infty} e^{-x} dx = 1 \tag{2}$$

만약 $\alpha = 1/2$ 이면, 다음 식들이 성립한다.

$$\Gamma\left(\frac{1}{2}\right) = \int_0^\infty x^{1/2-1} e^{-x} dx = \int_0^\infty x^{-1/2} e^{-x} dx \tag{3}$$

식 (3)의 우변에 변수변환 $x = y^2/2$를 적용하면, 다음 식들이 성립함을 알 수 있다.

$$\Gamma\left(\frac{1}{2}\right) = \sqrt{2} \int_0^\infty \exp\left(-\frac{1}{2} y^2\right) dy = \sqrt{2} \cdot \frac{\sqrt{2\pi}}{2} = \sqrt{\pi} \tag{4}$$

여기서 두 번째 등호는 정규분포의 정의에 의해서 성립한다.

식 (1)에 부분적분을 적용하면, 다음 식들이 성립함을 알 수 있다.

$$\Gamma(\alpha) = \int_0^\infty x^{a-1} e^{-x} dx = [-x^{a-1} e^{-x}]_0^\infty + [a-1] \int_0^\infty x^{a-2} e^{-x} dx \tag{5}$$

즉, 다음 점화식이 성립한다.

$$\Gamma(\alpha) = [\alpha - 1]\Gamma(\alpha - 1) \tag{6}$$

만약 n이 자연수이면, 식 (2)와 식 (6)에서 알 수 있듯이 다음 식이 성립한다.

$$\Gamma(n) = (n-1)!, \quad (n = 1, 2, \cdots) \tag{7}$$

또한, 다음과 같이 $\Gamma(0)$를 정의한다.

$$\Gamma(0) \doteq 1 \tag{8}$$

각 $z(> 0)$에 대해서 감마함수 $\Gamma(z)$는 다음 식을 만족한다.

$$\ln \Gamma(z) = \frac{1}{2} \ln 2\pi + \left[z - \frac{1}{2}\right] \ln z - z + \frac{1}{12z} - \frac{1}{360z^3} + \frac{1}{1260z^5} - \cdots \tag{9}$$

식 (9)에 대해서는 Abramowitz & Stegun [7, p. 257]을 참조하라. 식 (9)를 다음과 같이 쓸 수 있다.

$$\Gamma(z) = \sqrt{2\pi} e^{-z} z^{z-1/2} \left[1 + \frac{1}{12z} + \frac{1}{288z^2} - \frac{139}{51840z^3} + \cdots\right] \tag{10}$$

만약 z가 큰 자연수 $n+1$이면, 식 (10)에서 알 수 있듯이 다음 근사식이 성립한다.

$$n! \approx \sqrt{2\pi} e^{-n-1} [n+1]^{n+1-1/2} \tag{11}$$

다음 식들이 성립한다.

$$\sqrt{2\pi}e^{-n-1}\,[n+1]^{n+1-1/2} = \sqrt{2\pi}\sqrt{n+1}\,e^{-n}n^n e^{-1}\left[1+\frac{1}{n}\right]^n \tag{12}$$

식 (11)과 식 (12)에서 알 수 있듯이, 충분히 큰 n에 대해서 다음 근사식이 성립한다.

$$n! \approx \sqrt{2\pi n}\; n^n e^{-n} \tag{13}$$

식 (13)을 Stirling근사식이라 부른다. 식 (10)에서 한 항을 더 취하면, 식 (13)보다 약간 정교한 다음 근사식을 얻는다.

$$n! \approx \sqrt{2\pi n}\; n^n e^{-n}\left[1+\frac{1}{12n}\right] \tag{14}$$

Stirling근사식에 대한 자세한 내용은 Arfken & Weber & Harris [9, pp. 622-625]를 참조하라.

Stirling근사식의 정밀도를 조사하기 위해서, 다음 MATLAB프로그램 StirlingFormula101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: StirlingFormula101.m
% Gamma Function and Stirling's formula
% Programmed by CBS
% ------------------------------------------------------------------------
s = 1:1:15;
Gam = factorial(s);
logS1 = log(sqrt(2*pi)) + (s+0.5).*log(s) - s;
S1 = exp(logS1);
S2 = S1.*(1+1/12./s);
r1 = S1./Gam;
r2 = S2./Gam;
% Plotting
plot(s,r1,'k*',s,r2,'kd')
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf Stirling 1/Gamma','\bf Stirling 2/Gamma','Location','SE')
hold on
plot(s,r1,'g',s,r2,'g',s,r1,'k*',s,r2,'kd','LineWidth',1.5)
xlabel('\bf n','FontSize',12), ylabel('\bf Ratio','FontSize',12)
axis( [ 1 15 0.91 1.01 ])
hold off
saveas(gcf,'StirlingFormula101','jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 StirlingFormula101.m에서는 Stirling근사식을 계산할 때 로그를 취해서 계산한 다음, 역으로 지수함수를 사용했음에 유의하라. 그 이유는 오버플로우(overflow)가 되는 것을 방지하기 위해서이다. 이 MATLAB프로그램에서는 첫 번째 Stirling근사식 (13)

은 변수 S1으로, 그리고 두 번째 Stirling근사식 (14)는 변수 S2로 나타낸다.

이 MATLAB프로그램을 실행하면, 그림 1.2.1이 그려진다. 그림 1.2.1에서 알 수 있듯이, 첫 번째 Stirling근사식 (13)의 정확도는 점점 증가해서, $n = 10$인 경우 Stirling근사식 (13)에 의한 값을 $\Gamma(11)$로 나눈 값이 0.99170이다. 또한, 두 번째 Stirling근사식 (14)의 정확도 역시 점점 증가해서, $n = 10$인 경우 Stirling근사식 (14)에 의한 값을 $\Gamma(11)$로 나눈 값이 0.99998이다. ■

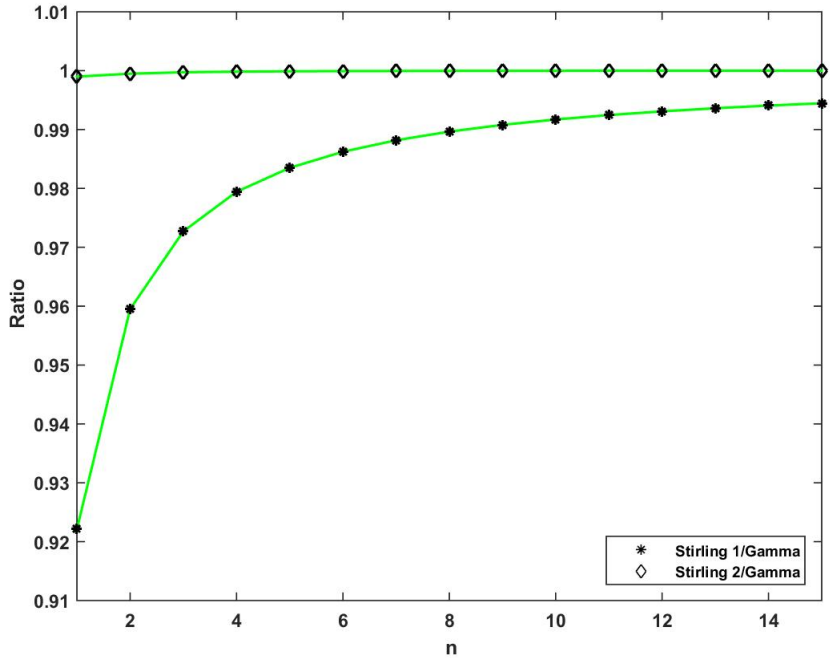


그림 1.2.1. Stirling근사식

예제 1.2.6 Python을 사용해서 예제 1.2.5을 다시 다루기 위해서, 다음 Python프로그램 StirlingFormula101.Py를 실행해 보자.

```
"""
Created on Thu Jan 12 08:59:40 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
import math

def stirling(n):
    return math.sqrt(2*math.pi*n)*(n/math.e)**n
def stirling2(n):
    return math.sqrt(2*math.pi*n)*(n/math.e)**n*(1+1/(12*n))

```

```
nMax = 16
# nObs = ratio1 = ratio2 = np.ones((nMax,))

print("n","\t", "Stirling","\t\tFactorial")
nf = 1
ratio1 = []
ratio2 = []
nObs = []
for x in range(1,nMax):
    nf *= x
    print(x,"\t", stirling(x), "\t\t", nf)
    nObs = np.append(nObs, x)
    ratio1 = np.append(ratio1, stirling(x)/nf)
    ratio2 = np.append(ratio2, stirling2(x)/nf)
fig, axes = plt.subplots(nrows=1, ncols=1)
plt.plot(nObs, ratio1,'r-*', lw=3, label='Stirlings 1/Gamma')
plt.plot(nObs, ratio2,'b-d', lw=3, label='Stirlings 2/Gamma')
plt.axis([0, 15, 0.91, 1.01])
plt.legend( loc='lower right', numpoints = 1 )
plt.show()
fig.savefig('StirlingFormula101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.5의 결과와 같다. ■

예제 1.2.7 어떤 $\alpha(\in (0,1))$에 대해서 함수 $f(x) = x^{\alpha}$를 살펴보자. 함수 $f(x)$를 $x = 1$에서 Taylor전개하면, 다음과 같다.

$$f(x) = \sum_{n=0}^{\infty} \frac{1}{n!}\alpha[\alpha-1]\cdots[\alpha-n+1][x-1]^n \doteq \sum_{n=0}^{\infty} c_n[x-1]^n \tag{1}$$

다음 식들이 성립한다.

$$\begin{aligned} c_n &\doteq \frac{\alpha[\alpha-1]\cdots[\alpha-n+1]}{1\cdot 2\cdot\cdots\cdot n} \\ &= [-1]^n \frac{[-\alpha][-\alpha+1]\cdots[-\alpha+n-1]}{1\cdot 2\cdot\cdots\cdot n} \\ &= [-1]^n \frac{\Gamma(-\alpha+n)}{\Gamma(-\alpha)\Gamma(n+1)} \end{aligned} \tag{2}$$

예제 1.2.5에서 알 수 있듯이, 충분히 큰 $z(>0)$에 대해서, 다음 Stirling근사식이 성립한다.

$$\Gamma(z+1) \approx \sqrt{2\pi z}\; z^z e^{-z} \tag{3}$$

식 (3)을 식 (2)에 대입하면, 다음 식들이 성립함을 알 수 있다.

$$\begin{aligned}
&\frac{\alpha[\alpha-1]\cdots[\alpha-n+1]}{1\cdot 2\cdot\cdots\cdot n} \\
&\approx [-1]^n \frac{1}{\Gamma(-\alpha)} \frac{\sqrt{2\pi[-\alpha+n-1]}[-\alpha+n-1]^{-\alpha+n-1}e^{-[-\alpha+n-1]}}{\sqrt{2\pi n}\, n^n e^{-n}} \\
&= [-1]^n \frac{1}{\Gamma(-\alpha)} \frac{1}{[n-\alpha-1]^{\alpha+1}} e^{1+\alpha} \left[1-\frac{1+\alpha}{n}\right]^n \left[1-\frac{1+\alpha}{n}\right]^{\frac{1}{2}} \\
&\approx [-1]^n \frac{1}{\Gamma(-\alpha)} \frac{1}{[n-\alpha-1]^{1+\alpha}} \qquad (4)
\end{aligned}$$

식 (2)와 식 (4)에서 알 수 있듯이, 다음 근사식이 성립한다.

$$|c_n| \approx \left| \frac{1}{\Gamma(-\alpha)} \frac{1}{[n-\alpha-1]^{1+\alpha}} \right| \qquad (5)$$

따라서, 식 (2)의 Taylor계수의 절대값 $|c_n|$의 감소속도가 느리다.

Taylor계수의 절대값의 감소속도를 조사하기 위해서, 다음 MATLAB프로그램 Taylor-Converge101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: TaylorConverge101.m
% Convergence of Taylor series
% Programmed by CBS
% ------------------------------------------------------------------------
clear, clf
n = 5:35;
alpha = 1/4;
absC = 1/( abs(gamma(-alpha) ) )...
       ./( (sign(n - alpha -1).*(n - alpha -1) ).^( 1+alpha ) );
plot(n,absC,'g',n,absC,'kd','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf n','FontSize',12),
ylabel('{\bf Absolute Value of Taylor Coefficient}','FontSize',12)
saveas(gcf,'TaylorConverge101','jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, $\alpha = 0.25$인 경우에 Taylor계수의 절대값 $|c_n|$을 그린 그래프를 출력한다. 이 그래프가 그림 1.2.2에 수록되어 있다. 그림 1.2.2에서 X축은 n이고 Y축은 Taylor계수의 절대값 $|c_n|$이다. 그림 1.2.2에서 $|c_n|$의 감소속도가 느리다는 것을 확인할 수 있다. ■

예제 1.2.8 Python을 사용해서 예제 1.2.7을 다시 다루기 위해서, 다음 Python프로그램 TaylorConverge101.Py를 실행해 보자.

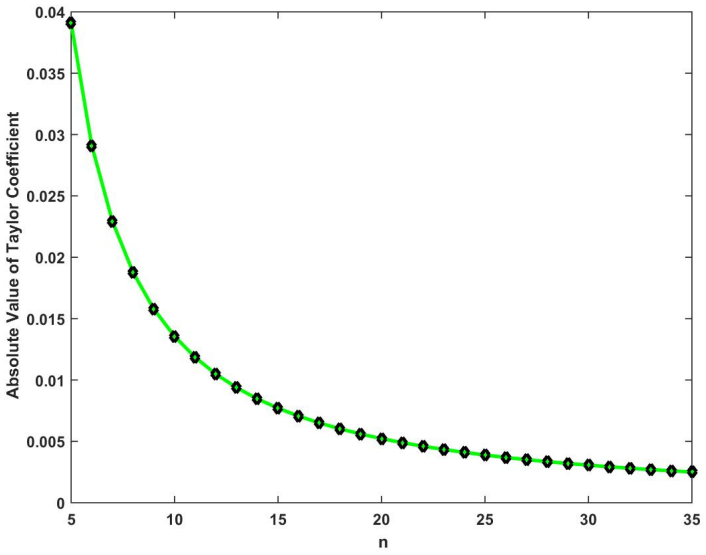


그림 1.2.2. Taylor계수의 수렴

```
"""
Created on Thu Jan 12 08:59:40 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
import math

def AbsError(alpha,n):
    return 1/( abs(math.gamma(-alpha)*(n-alpha-1)**(1+alpha)) )

alpha = 1/4

n = np.arange(5,36,1)
y = AbsError(alpha,n)
fig = plt.figure()
plt.plot(n,y,'r^-')
plt.xlabel('n')
plt.ylabel('Absolute Value')
plt.show()
fig.savefig('TaylorConverge101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.7의 결과와 같다. ■

Taylor근사를 사용하기 위해서는 어떤 차수까지 미분가능한지가 매우 중요하다. 다음 예제에서는 미분가능하지만 도함수가 연속이 아닌 함수를 보여준다.

예제 1.2.9 함수 f가 C^m급이면, 이 함수의 m차 Taylor근사식을 구할 수 있다. 다음

함수를 살펴보자.

$$f(x) = \begin{cases} x^2 \sin \frac{1}{x}, & (x \neq 0) \\ 0, & (x = 0) \end{cases} \tag{1}$$

함수 $f(x)$는 점 $x = 0$에서 연속이고, 미분가능하다. 이 함수의 1차 도함수는 다음과 같다.

$$f'(x) = \begin{cases} 2x \sin \frac{1}{x} - \cos \frac{1}{x}, & (x \neq 0) \\ 0, & (x = 0) \end{cases} \tag{2}$$

그러나, 도함수 $f'(x)$는 점 $x = 0$에서 연속이 아니며, 또한 미분가능하지 않다. 즉, 함수 f가 C^0급이다. 따라서, 함수 f의 $x = 0$에서 Taylor근사식은 다음과 같다.

$$f(x) \approx f(0) = 0 \tag{3}$$

지금까지 내용을 확인하기 위해서, 다음 MATLAB프로그램 NotContiDiff101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: NotContiDiff101.m
% Example of Not-Continuously Differentiable Function
% Programmed by CBS
% ------------------------------------------------------------------------
syms x
f = x^2*sin(1/x)
df = diff(f)
d2f = diff(f,2)
% ploting
xx = -0.5+eps:0.001:0.5+eps;
ff = xx.^2.*sin(1./xx);
dff = 2*xx.*sin(1./xx) - cos(1./xx);
d2ff = 2*sin(1./xx) - (2*cos(1./xx))./xx - sin(1./xx)./xx.^2;
subplot(1,2,1)
plot(xx,ff,'k-','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
axis([-0.5 0.5 -0.2 0.2])
xlabel('\bf x','FontSize',12)
ylabel('\bf y','FontSize',12,'Rotation',0)
subplot(1,2,2)
plot(xx,dff,'r-','LineWidth',1.5)
set(gca,'fontsize',11,'fontweigh','bold')
axis([-0.5 0.5 -1.5 1.5])
xlabel('\bf x','FontSize',12)
ylabel('\bf dy/dx','FontSize',12,'Rotation',0)
saveas(gcf,'NotContiDiff101','jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 그림 1.2.3이 그려진다. 그림 1.2.3에서 좌측 그래프는

함수 $f(x)$를, 그리고 우측은 1차 도함수 $f'(x)$를 그린 것이다. 그림 1.2.3에서 알 수 있듯이, 점 $x=0$에서 함수 $f(x)$는 평활하나 1차 도함수 $f'(x)$는 진동이 아주 심하다. ■

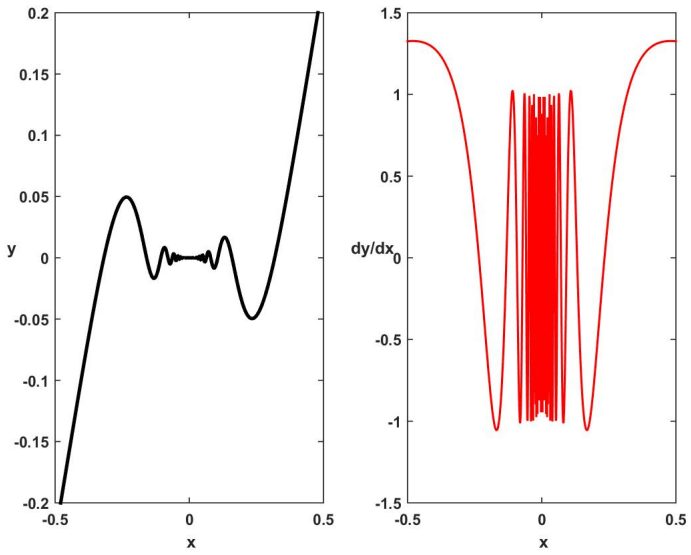


그림 1.2.3. 도함수가 불연속인 함수

예제 1.2.10 Python을 사용해서 예제 1.2.9을 다시 다루기 위해서, 다음 Python프로그램 NotContiDiff101.Py를 실행해 보자.

```
"""
Created on Thu Jan 12 08:59:40 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

from sympy import *
x = Symbol('x')
f = x**2*sin(1/x)
df = diff(f,x,1)
d2f = diff(f,x,2)

# Plotting
MachEps = np.finfo(float).eps
xx = np.linspace(-0.5+MachEps, 0.5+MachEps, 1024, endpoint=True)
ff = xx**2*np.sin(1/xx)
dff = 2*xx*np.sin(1/xx) - np.cos(1/xx)
d2ff = 2*np.sin(1/xx) - 2*np.cos(1/xx)/xx - np.sin(1/xx)/xx**2

# plt.figure(1)
fig, axes = plt.subplots(nrows=1, ncols=2)
ax1 = plt.subplot(121)
plt.plot(xx,ff,'k')
plt.xlabel('x'), plt.ylabel('f(x)')
ax2 = plt.subplot(122)
```

```
plt.plot(xx,dff,'r')
plt.xlabel('x'), plt.ylabel('df/dx')
ax2.yaxis.set_ticks_position("right") # ticks and ticklabels
ax2.yaxis.set_label_position("right") # axis label
fig.tight_layout()
plt.show()
fig.savefig('NotContiDiff101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.9의 결과와 같다. ■

복소함수론(complex analysis)에서 멱급수의 성질이 자세히 연구되고 있다. 지금부터는 복소함수론에서 근사이론에 필요한 정의와 명제를 빌려와 간단히 기술하고자 한다. 이에 대한 좀 더 자세한 내용은 복소함수론에 관한 서적을 참조하라. 본저자는 복소함수론에 관해서는 Wegert [65]를 읽을 것을 권한다.

정의 1.2.4

복소평면 C의 부분집합이며 연결된(connected) 개집합(open set) Ω에 대해서 함수 $f:\Omega\to C$를 살펴보자.

(a) 만약 각 $\alpha(\in\Omega)$에 대해서 다음 명제를 만족하는 양수 r과 수열 $\{c_k\}$가 존재하면, 함수 $f(z)$은 집합 Ω 상에서 해석적(analytic)이라 한다.

$$|z-a|<r \;\Rightarrow\; f(z)=\sum_{n=0}^{\infty} c_n[z-a]^n$$

(b) 만약 함수 f가 집합 Ω 상에서 해석적은 아니나 집합 $\Omega-\{a\}$ 상에서 해석적이면, 점 a를 함수 f의 특이점(singularity)이라 한다.

(c) 만약 a가 함수 f의 특이점이며, 또한 만약 $f(z)[z-a]^m$이 집합 Ω 상에서 해석적인 자연수 m이 존재하면, 점 a를 함수 f의 극점(pole)이라 한다.

(d) 만약 $f(z)[z-a]^m$이 집합 Ω 상에서 해석적이고, 또한 만약 $f(z)=[z-a]^{m-1}$이 집합 Ω 상에서 해석적이 아니면, 극점 a의 중복도(multiplicity)는 m이라 한다.

(e) 만약 함수 f가 식 $|z|<\infty$를 만족하는 각 점 $z(\in\Omega)$에서 해석적이면, 함수 $f(z)$를 정함수(**整函數**, entire function)라 한다.

(f) 만약 함수 g와 함수 h가 정함수들이고, 또한 만약 함수 h가 각 점 $z(\in \Omega)$에서 0이 아니면, 함수 $f \doteq g/h$를 유리형함수(meromorphic funtion)라 한다.

Weisstein [67, p. 1896]에 의하면, 그리스어에서 meros는 부분(part)이라는 뜻이고 morphe는 형태(form) 또는 모습(appearance)이라는 뜻이다.

명제 1.2.2

함수 f가 점 $z(\in C)$에서 해석적이라 하자. 만약 함수 f나 이 함수의 도함수들 중 하나가 $z_S(\in C)$를 특이점으로 갖으면, 점 a에서 Taylor급수 $\sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(a)[z-a]^n$ 의 수렴반경 r은 $||\, a - z_S \,||$보다 작다.

예제 1.2.11 점 $x = 0$에서 함수 $f(x) = [x+1]^{1/4}$의 Taylor전개는 다음과 같다.

$$f(x) = \sum_{n=0}^{\infty} [-1]^n \frac{\Gamma\left(-\frac{1}{4}+n\right)}{\Gamma\left(-\frac{1}{4}\right) n!} x^n \tag{1}$$

함수 $f(x)$를 점 $x = 0$에서 6차까지 Taylor전개한 근사식의 그래프를 그리기 위해서, 다음 MATLAB프로그램 TaylorApprox101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: TaylorApprox101.m
% Convergence of Taylor series
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
syms xx
yy = (xx+1).^(1/4)
yy6 = taylor(yy,xx,7)  % Taylor Expansion w/ order 7
% plot
x = -1:0.003:30;
y = (x+1).^(1/4);
y6 = 1 + 1/4*x - 3/32*x.^2 + 7/128*x.^3 - 77/2048.*x.^4 ...
      + 231/8192*x.^5 - 1463/65536*x.^6; % Taylor approx w/order 6
err6 = abs(y - y6)+eps;
dumy = logspace(-20,10,300);
dumx1 = -1*ones(300);
dumx2 = 1*ones(300);
semilogy(x,err6,'r',dumx1,dumy,'b:',dumx2,dumy,'b:','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('{\bf x}','fontsize',12)
ylabel('{\bf Error}','fontsize',12,'rotation',0)
text(-0.7,10^(-17),'\bf ROC')
axis([-2,18,10^(-18),10^8 ])
saveas(gcf,'TaylorApprox101','jpg')
% end of program
```

```
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 TaylorApprox101.m에서 MATLAB함수 taylor는 점 $x = 0$에서 Taylor전개를 하기 위한 것이다. 특히, MATLAB명령문 'yy6 = taylor(yy,xx,7)'은 함수 yy를 6차까지 전개하는 것이다. 이 MATLAB명령문을 실행한 결과, Taylor근사식이 다음과 같음을 알 수 있다.

$$y = 1 + \frac{1}{4}x - \frac{3}{32}x^2 + \frac{7}{128}x^3 - \frac{77}{2048}x^4 + \frac{231}{8192}x^5 - \frac{1463}{65536}x^6 \tag{2}$$

또한, MATLAB함수 semilogy는 Y 축을 로그스케일로 표기하는 2차원 그래프를 그리기 위한 것이다.

이 MATLAB프로그램을 수행하면, 그림 1.2.4가 그려진다. 그림 1.2.4에서 알 수 있듯이, 식 $|x| < 1$을 만족하지 못하는 점에서 이 Taylor근사식은 좋은 근사식이 아니다. ■

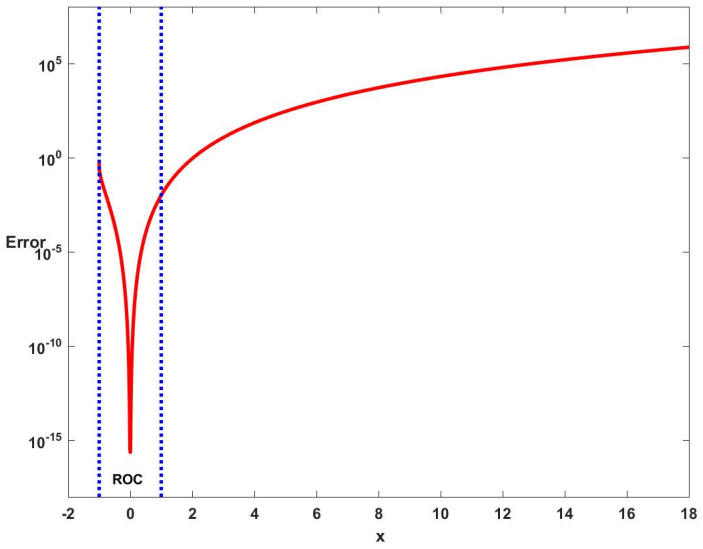


그림 1.2.4. Taylor전개와 ROC

예제 1.2.12 Python을 사용해서 예제 1.2.11을 다시 다루기 위해서, 다음 Python프로그램 TaylorApprox101.Py를 실행해 보자.

```
"""
Created on Thu Jan 12 20:13:46 2017
@author: CBS
"""
import numpy as np
```

```
import matplotlib.pyplot as plt
import math

from sympy import *
xx = Symbol('xx')
yy0 = (xx+1)**(1/4)
yy1 = yy0.diff(xx)
yy2 = yy1.diff(xx)
yy3 = yy2.diff(xx)
yy4 = yy3.diff(xx)
yy5 = yy4.diff(xx)
yy6 = yy5.diff(xx)
coeff0 = yy0.subs(xx, 0)/math.gamma(1)
coeff1 = yy1.subs(xx, 0)/math.gamma(2)
coeff2 = yy2.subs(xx, 0)/math.gamma(3)
coeff3 = yy3.subs(xx, 0)/math.gamma(4)
coeff4 = yy4.subs(xx, 0)/math.gamma(5)
coeff5 = yy5.subs(xx, 0)/math.gamma(6)
coeff6 = yy6.subs(xx, 0)/math.gamma(7)

# Plotting
x = np.linspace(-1, 30, 1000, endpoint=True)
y = (x+1)**(1/4)
y6 = coeff0 + x*( coeff1 + x*( coeff2 + x*( coeff3 + x*( coeff4 + x*(
     coeff5 + x) ) ) ) )
MachEps = np.finfo(float).eps
err6 = np.abs(y - y6) + MachEps

fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(x,err6,'r-')
plt.yscale('log')
plt.xlabel('x')
plt.ylabel('Error')
plt.axis([-2, 18, 10**(-18), 10**(8)])
ax.text(-0.5,10**(-17),'ROC')
plt.show()
fig.savefig('TaylorApprox101Py.png')


# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.11의 결과와 같다. ■

예제 1.2.13 신고전파경제학의 성장모형에서 경제성장경로가 안정성을 갖기 위한 충분조건은 생산함수 $Y = F(K, L)$이 다음 6가지 조건들을 만족해야 한다.

(a) $F(0, 0) = 0$

(b) $F \in C^1 \times C^1$

(c) 함수 Y는 K와 L의 단조증가함수이다.

(d) 함수 Y는 K와 L의 (아래로) 오목함수이다.

(e) $\lim_{K\to\infty}\frac{\partial F}{\partial K}=0, \quad \lim_{L\to\infty}\frac{\partial F}{\partial L}=0$

(f) $\lim_{K\to 0}\frac{\partial F}{\partial K}=\infty, \quad \lim_{L\to 0}\frac{\partial F}{\partial L}=\infty$

이러한 조건들을 Inada조건이라 한다. Inada조건에 대해서는 Inada [30]과 Uzawa [62]을 참조하라.

Inada조건을 만족하는 대표적인 생산함수는 Cobb-Douglas함수이다. 조건 (f)는 점 $(K,L)=(0,0)$가 생산함수 $Y=F(K,L)$의 특이점임을 의미한다. 따라서, 점 $(K,L)=(1,1)$에서 Taylor급수를 구하면, 조건 $||(K^*,L^*)-(1,1)|| \geq \sqrt{2}$을 만족하는 점 (K^*,L^*)에서 Taylor급수를 바탕으로 하는 근사식이 정확도를 갖는다고 할 수는 없다. 일반적으로 생산함수나 효용함수는 특이점을 갖는다. 따라서, 명제 1.2.2는 경제학에서 매우 중요한 것이다. ■

Taylor근사식을 구하기 위해서, 함수 f의 도함수들이 꼭 필요한 것은 아니다. 다음 함수를 생각해보자.

$$t(x)=t_0+t_1[x-x_0]+\cdots+t_n[x-x_0]^n \tag{1.2.5}$$

또한, 다음 점들을 생각해보자.

$$x_1<x_2<\cdots<x_n \tag{1.2.6}$$

다음 값들을 정의하자.

$$f_0\doteq f(x_0), \quad f_i\doteq f(x_i), \quad (i=1,2,\cdots,n) \tag{1.2.7}$$

함수 $f(x)$의 Taylor근사식 $t(x)$가 다음 식들을 만족한다고 가정하자.

$$t_0=f_0, \quad t(x_i)=f_i, \quad (i=1,2,\cdots,n) \tag{1.2.8}$$

식 (1.2.5)에서 알 수 있듯이, 다음 연립방정식이 성립한다.

$$t_0+t_1[x_i-x_0]+\cdots+t_n[x_i-x_0]^n=f_i, \quad (i=1,2,\cdots,n) \tag{1.2.9}$$

다음 식이 성립함은 명백하다.

$$t_0=f_0 \tag{1.2.10}$$

다음 행렬들과 벡터들을 정의하자.

$$A \doteq \begin{bmatrix} 1 & [x_1 - x_0] & \cdots & [x_1 - x_0]^{n-2} & [x_1 - x_0]^{n-1} \\ 1 & [x_2 - x_0] & \cdots & [x_2 - x_0]^{n-2} & [x_2 - x_0]^{n-1} \\ \vdots & \vdots & & \vdots & \vdots \\ 1 & [x_{n-1} - x_0] & \cdots & [x_{n-1} - x_0]^{n-2} & [x_{n-1} - x_0]^{n-1} \\ 1 & [x_n - x_0] & \cdots & [x_n - x_0]^{n-2} & [x_n - x_0]^{n-1} \end{bmatrix} \tag{1.2.11}$$

$$\boldsymbol{t} \doteq [t_1 \; t_2 \; \cdots \; t_n]^t \tag{1.2.12}$$

$$\boldsymbol{b} \doteq \left[\frac{f_1 - f_0}{x_1 - x_0} \; \frac{f_2 - f_0}{x_2 - x_0} \; \cdots \; \frac{f_n - f_0}{x_n - x_0} \right]^t \tag{1.2.13}$$

따라서, 식 (1.2.9)을 다음과 같이 쓸 수 있다.

$$A\boldsymbol{t} = \boldsymbol{b} \tag{1.2.14}$$

다음 예제는 식 (1.2.14)를 풀어서 Taylor근사식을 구하는 예이다.

예제 1.2.14 다음 5개 점들을 지나는 4차 멱함수인 Taylor근사식을 구해보자.

$$I \doteq \{(10^{-1}, 10^{-1/4}), (10^0, 10^0), (10^1, 10^{1/4}), (10^2, 10^{1/2}), (10^3, 10^{3/4})\} \tag{1}$$

다음 멱함수를 정의하자.

$$t(x) \doteq \sum_{i=0}^{4} t_i [x - 1]^i \tag{2}$$

또한, 다음 값들을 선택하자.

$$t_0 = 1, \quad f_0 = 1 \tag{3}$$

다음 식들이 성립한다.

$$A = \begin{bmatrix} 1 & [10^{-1} - 1] & [10^{-1} - 1]^2 & [10^{-1} - 1]^3 \\ 1 & [10^1 - 1] & [10^1 - 1]^2 & [10^1 - 1]^3 \\ 1 & [10^2 - 1] & [10^2 - 1]^2 & [10^2 - 1]^3 \\ 1 & [10^3 - 1] & [10^3 - 1]^2 & [10^3 - 1]^3 \end{bmatrix} \tag{4}$$

$$\boldsymbol{b} = \left[\frac{10^{-1/4}-1}{10^{-1}-1}, \frac{10^{1/4}-1}{10^{1}-1}, \frac{10^{1/2}-1}{10^{2}-1}, \frac{10^{3/4}-1}{10^{3}-1} \right]^t \tag{5}$$

우리의 문제는 다음 연립방정식을 구하는 것으로 귀착된다.

$$A\boldsymbol{t} = \boldsymbol{b} \tag{6}$$

연립방정식 (6) 의 근을 구하기 위해서, 다음 MATLAB프로그램 TaylorInter101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: TaylorInter101.m
% Graph of Taylor approximation of x^0.25
% Programmed by CBS
% ------------------------------------------------------------------------
clear, clf, format long
% Find Taylor coefficients
nterm = 4;
x0 = 1; f0 = 1;
x = [ 10^-1 10^1 10^2 10^3 ]';
f = [ 10^(-1/4) 10^(1/4) 10^(2/4) 10^(3/4) ]';
xc = x - x0;
cc = (f - f0)./xc;
A = zeros(nterm,nterm);
for jj=1:nterm
    A(:,jj) = xc.^(jj-1);
end
   % Can use matlab function vander.m instead.
tt = inv(A)*cc
if abs(det(A)) < 1.e-10
    disp('Warning: Matrix is near singular')
    disp('determinant is'), disp(det(A))
end
% Plot Taylor approximate
xx = logspace(-2,3,1000);
yy = xx.^(1/4);
tf = f0 + xx.*(tt(1) + xx.*( tt(2) + xx.*( tt(3) + xx.* tt(4) )));
plot(xx,yy,'b-.',x,f,'r*',xx,tf,'k-','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('function','Data Point','Taylor','location','NorthWest')
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
saveas(gcf,'TaylorInter101','jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB명령문이 실행하면, 연립방정식의 해가 다음과 같음을 알 수 있다.

$$\boldsymbol{t} = \begin{bmatrix} 0.446407 \\ -0.043916 \\ 0.000440 \\ -0.000000 \end{bmatrix} \tag{7}$$

즉, 함수 $f(x) = x^{1/4}$ 의 4차 Taylor근사식은 다음과 같다.

$$t(x) = 1 + x\,(0.446407 + x\,\{-0.043916 + x[0.000440 - 0.000000x]\}) \tag{8}$$

뒤에서 다시 설명하겠지만, 식 (8)과 같은 형식을 Horner형식이라 한다.

이 MATLAB명령문이 실행되면, 함수 $f(x)$와 Taylor근사식 $t(x)$을 비교하기 위한 그림 1.2.5가 그려진다. 그림 1.2.5에서 알 수 있듯이, Taylor근사식은 데이터세트 I의 점들을 지난다. 그러나, x가 커지면 Taylor근사식 $t(x)$는 함수 $f(x)$로부터 멀리 떨어지는 경향이 있다. ■

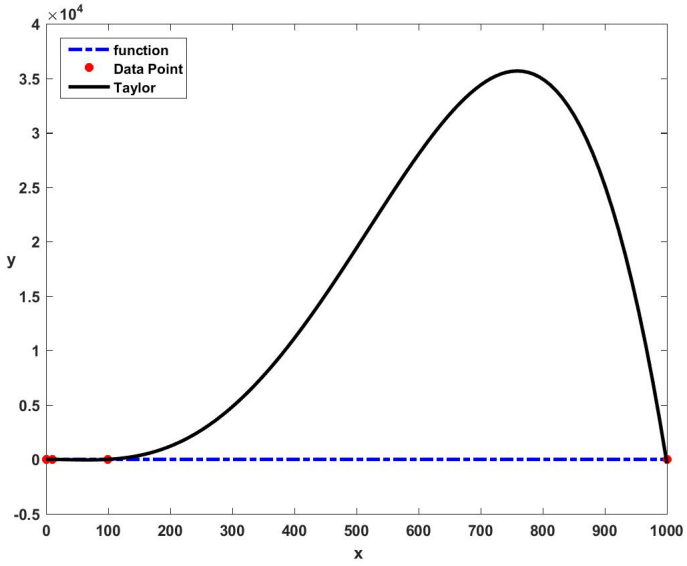


그림 1.2.5. Taylor근사식

예제 1.2.15 Python을 사용해서 예제 1.2.14을 다시 다루기 위해서, 다음 Python프로그램 TaylorInter101.Py를 실행해 보자.

```
"""
Created on Thu Jan 12 08:59:40 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

nterm = 4
x0 = 1
f0 = 1
x = np.array( [ 10**(-1), 10**1, 10**2, 10**3 ] )
f = np.array( [ 10**(-1/4), 10**(1/4), 10**(2/4), 10**(3/4) ] )
xc = x-x0;
cc = (f-f0)/xc
A = np.zeros((nterm,nterm),float)
for ii in range(1,nterm+1):
    for jj in range(1,nterm+1):
        A[ii-1,jj-1] = xc[ii-1]**(jj-1)
invA = np.linalg.inv(A)
tt = np.dot(invA,cc)
if abs(np.linalg.det(A)) < 10**(-10):
    print('Warning: Matrix is near singular')
    print('determinant is'), print(np.linalg.det(A))

# Plot Taylor approximate
xx = np.logspace(-2, 3, num=1000);
yy = xx**(1/4);
tf = f0 + xx*(tt[0] + xx*( tt[1] + xx*( tt[2] + xx* tt[3] )))
fig, axes = plt.subplots(nrows=1, ncols=1)
plt.plot(xx,yy,'b-.', lw=3, label='function')
plt.plot(x,f,'rd', lw=3, label='Data Point')
plt.plot(xx,tf,'k-', lw=3, label='Taylor')
plt.legend(loc='upper left', numpoints = 1 )
plt.show()
fig.savefig('TaylorInter101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.14의 결과와 같다. ■

예제 1.2.16 주어진 데이터세트에 적합한 근사식을 구할 때는 주어진 데이터세트에 적합한 변수변환을 해야 한다. 예를 들어, 예제 1.2.14에서 변수 x를 $\log_{10} x$로 변환하면, 훨씬 더 적합한 근사식을 구할 수 있다. 이 근사식을 구하기 위해서, 다음 MATLAB프로그램 TaylorInter102.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: TaylorInter102.m
% Graph of log-transformed Taylor approximation of x^0.25
% Programmed by CBS
% ------------------------------------------------------------------------
clear, clf, format long
% find Taylor coefficients
nterm = 4;
x = [ 10^-1 10^1 10^2 10^3 ]';
```

```
f = [ 10^(-1/4) 10^(1/4) 10^(2/4) 10^(3/4) ]';
logx0 = log10(1); f0 = 1;
logx = log10(x) - logx0
cc = (f - f0)./logx;
A = zeros(nterm,nterm);
for ii=1:nterm
    A(:,ii) = logx.^(ii-1);
end
tt = inv(A)*cc
if abs(det(A)) < 1.e-10
    disp('Warning: Matrix is near singular')
    disp('determinant is'), disp(det(A))
end
% plot Taylor approximate
logxx =  linspace(-2/log(10),4/log(10),1000)*log(10);
yy = 10.^(logxx/4);
tt = f0+logxx.*(tt(1)+logxx.*(tt(2)+logxx.*(tt(3)+logxx.*tt(4))));
% plot
plot(logxx,yy,'b-.',log10(x),f,'rd',logxx,tt,'k-','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('function','Data Point','Log-Taylor','location','NW')
xlabel('\bf log_{10} x','FontSize',12)
ylabel('\bf y','FontSize',12,'rotation',0)
saveas(gcf,'TaylorInter102','jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB명령문이 실행하면, 연립방정식의 해가 다음과 같음을 알 수 있다.

$$t = \begin{bmatrix} 0.580979 \\ 0.161714 \\ 0.026990 \\ 0.008597 \end{bmatrix} \tag{1}$$

즉, 함수 $f(x) = x^{1/4}$ 의 근사식은 다음과 같다.

$$\begin{aligned} t(x) =& 1 + 0.580979[\log_{10} x] + 0.161714[\log_{10} x]^2 \\ &+ 0.026990[\log_{10} x]^3 + 0.008597[\log_{10} x]^4 \end{aligned} \tag{2}$$

이 MATLAB명령문이 실행되면, 함수 $f(x)$와 Taylor근사식 $t(x)$을 비교하기 위한 그림 1.2.6이 출력된다. 그림 1.2.6에서 알 수 있듯이, Taylor근사식은 데이터세트 I의 점들을 지난다. 그림 1.2.5와는 달리 근사식 $t(x)$는 함수 $f(x)$와 거의 동일함을 알 수 있다. 또한, 만약 변수 y대신 $\log_{10} y$를 사용하면, 이 근사식은 선형으로 나타내지고 또한 오차가 거의 사라질 것이다. ■

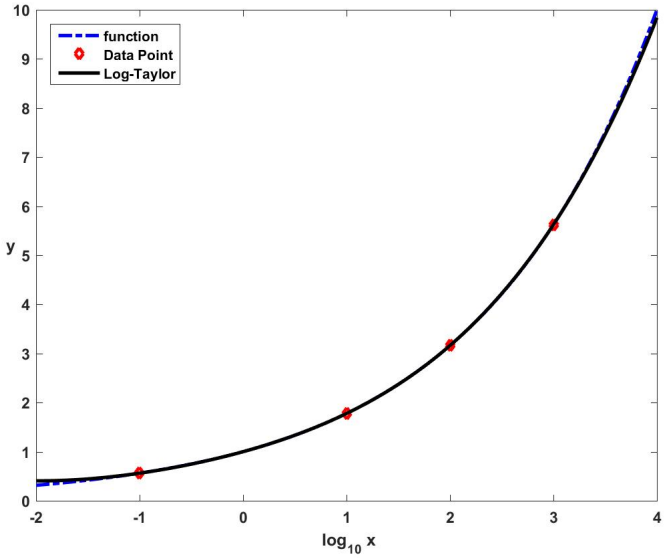


그림 1.2.6. Taylor근사식

예제 1.2.17 Python을 사용해서 예제 1.2.16을 다시 다루기 위해서, 다음 Python프로그램 TaylorInter102.Py를 실행해 보자.

```
"""
Created on Thu Jan 12 08:59:40 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

nterm = 4
x = np.array( [ 10**(-1), 10**1, 10**2, 10**3 ] )
f = np.array( [ 10**(-1/4), 10**(1/4), 10**(2/4), 10**(3/4) ] )
logx0 = np.log10(1)
f0 = 1.0
logx = np.log10(x) - logx0
cc = (f-f0)/logx
A = np.zeros((nterm,nterm),float)
for ii in range(1,nterm+1):
    for jj in range(1,nterm+1):
        A[ii-1,jj-1] = logx[ii-1]**(jj-1)
invA = np.linalg.inv(A)
tt = np.dot(invA,cc)
if abs(np.linalg.det(A)) < 10**(-10):
    print('Warning: Matrix is near singular')
    print('determinant is'), print(np.linalg.det(A))

# Plot Taylor approximate
logxx = np.linspace(-2/np.log(10), 4/np.log(10), num=1000)*np.log(10)
yy = 10**(logxx/4)
tf = f0 + logxx*(tt[0] + logxx*( tt[1] + logxx*( tt[2] + logxx* tt[3] )))
fig, axes = plt.subplots(nrows=1, ncols=1)
plt.plot(logxx,yy,'b-.', lw=3, label='function')
plt.plot(np.log10(x),f,'rd', lw=3, label='Data Point')
plt.plot(logxx,tf,'g-', lw=2, label='Log-Taylor')
plt.legend(loc='upper left', numpoints = 1 )
```

```
plt.show()
fig.savefig('TaylorInter102Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.16의 결과와 같다. ■

1.2.3 Padé근사

Taylor근사는 멱함수를 사용하는 근사인데 반해서, Padé근사는 유리함수를 사용하는 근사이다. Padé근사에 대한 좀 더 자세한 내용은 Petrushev & Popov [42]를 참조하라.

정의 1.2.5: Padé근사

점 x_0에서 함수 f의 (m,n)차 Padé근사는 다음 식을 만족하는 유리함수 $r(x)=p(x)/q(x)$이다.

$$\left.\frac{d^k[p-fq]}{dx^k}\right|_{x=x_0}=0, \quad (k=0,1,\cdots,m+n)$$

여기서 $p(x)$와 $q(x)$는 각각 최대차수가 m과 n인 멱함수들이다.

정의 1.2.5의 조건들을 다음과 같이 쓸 수 있다.

$$q(x_0)=1 \tag{1.2.15}$$

$$p^{(k)}(x_0)=(fq)^{(k)}(x_0), \quad (k=0,1,\cdots,m+n) \tag{1.2.16}$$

실제 분석에서는 $m=n$이거나 $m=n+1$인 경우를 주로 다룬다. 또한, 함수 f의 일부 특이점들이, 특히 극점(pole)들이 함수 q의 근이 되도록 함수 q를 선택한다. 이렇게 함수 q를 선택하면, Taylor전개가 잘 수렴하지 않는 점에서 Padé근사가 잘 작동하게 할 수 있다.

예제 1.2.18 함수 $f(x)=x^{1/4}$의 점 $x=1$에서 $(2,1)$차 Padé근사를 구하기로 하자. 우선, 함수 $f(x)$의 점 $x=1$에서 $[2+1]$차 Taylor근사식 $t(x)$를 구하자.

예제 1.2.7에서 알 수 있듯이, 다음 식이 성립한다.

$$t(x)=1+\frac{x-1}{4}-\frac{3[x-1]^2}{32}+\frac{7[x-1]^3}{128} \tag{1}$$

다음 멱함수들을 정의하자.

$$p(x) \doteq p_0 + p_1[x-1] + p_2[x-1]^2 \tag{2}$$

$$q(x) \doteq 1 + q_1[x-1] \tag{3}$$

다음 식을 만족하는 멱함수들 $p(x)$와 $q(x)$의 계수들을 구해보자.

$$t(x) = \frac{p(x)}{q(x)} \tag{4}$$

즉, 다음 식을 만족하는 계수들을 구해보자.

$$p_0 + p_1[x-1] + p_2[x-1]^2 - t(x)\{1 + q_1[x-1]\} = 0 \tag{5}$$

이 계수들이 다음 식들을 만족함은 명백하다.

$$p^{(k)}(1) = (fq)^{(k)}(1) = (tq)^{(k)}(1), \quad (k = 0, 1, 2, 3) \tag{6}$$

식 (6)에 항등식원리를 적용하면, 다음 식들이 성립함을 알 수 있다.

$$p_0 = 1, \quad p_1 = \frac{5}{6}, \quad p_2 = \frac{5}{96}, \quad q_1 = \frac{7}{12} \tag{7}$$

즉, 다음 식들이 성립한다.

$$p(x) = 1 + \frac{5}{6}[x-1] + \frac{5}{96}[x-1]^2 = \frac{7}{32} + \frac{35}{48}x + \frac{5}{96}x^2 \tag{8}$$

$$q(x) = 1 + \frac{7}{12}[x-1] = \frac{5}{12} + \frac{7}{12}x \tag{9}$$

따라서, 다음 Padé근사식이 성립한다.

$$r(x) = \frac{21 + 70x + 5x^2}{40 + 56x} \tag{10}$$

함수 $f(x) = x^{1/4}$, Taylor근사식 $t(x)$ 그리고 Padé근사식 $r(x)$를 비교하기 위해서, 다음 MATLAB프로그램 Taylor2Pade101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename Taylor2Pade101.m
```

```
% Taylor vs Pade
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
xx = 0:0.0002:4;
yy = xx.^(1/4);
tt = 1 + 1/4*(xx-1) - 3/32*(xx-1).^2 + 7/128*(xx-1).^3;
pp = (21 + 70*xx + 5*xx.^2 )./(40 + 56* xx) ;
% plot
plot(xx,yy,'r',xx,tt,'b--',xx,pp,'k-.','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('function','Taylor','Pade','location','SE')
xlabel('{\bf x}','fontsize',12)
ylabel('\bf y','FontSize',12,'rotation',0)
saveas(gcf,'Taylor2Pade101','jpg')
% end of program
% ------------------------------------------------------------------------
```

이 MATLAB 프로그램 Taylor2Pade101.m을 실행하면, 그림 1.2.7이 그려진다. 그림 1.2.7에서 알 수 있듯이, (청색) 긴점선으로 그려진 3차 Taylor근사식 $t(x)$보다 (검정색) 반점선으로 그려진 $(2,1)$차 Padé근사식 $r(x)$가 (적색) 실선으로 그려진 함수 $f(x)$에 훨씬 더 가까움을 알 수 있다. 특히, 점 $x = 1$에서 멀어질수록 그러한 현상은 더욱 뚜렷하다. 이 예제와 같이, Padé근사식은 놀라울만큼 근사정도가 높은 경우가 많다. ■

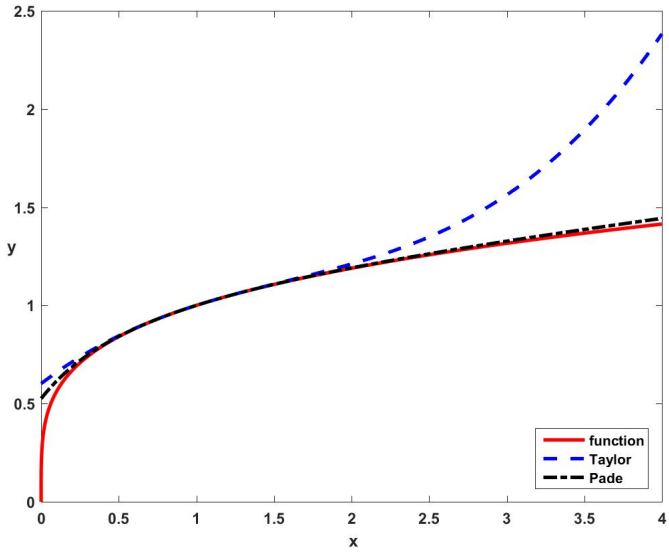


그림 1.2.7. Taylor근사식과 Padé근사식

예제 1.2.19 Python을 사용해서 예제 1.2.18을 다시 다루기 위해서, 다음 Python프로그램 taylor2Pade101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 11:15:14 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

xx = np.linspace(0, 4, 20000, endpoint=True)
yy = xx**(1/4);
tt = 1 + 1/4*(xx-1) - 3/32*(xx-1)**2 + 7/128*(xx-1)**3
pp = (21 + 70*xx + 5*xx**2 )/(40 + 56*xx)

# Plotting

fig, axes = plt.subplots(nrows=1, ncols=1)
plt.plot(xx,yy,'r', lw=3, label='function')
plt.plot(xx,tt,'b--', lw=3, label='Taylor')
plt.plot(xx,pp,'k-.', lw=3, label='Pade')
plt.legend(loc='lower right', numpoints = 1 )
plt.xlabel('x'), plt.ylabel('y')
plt.show()
fig.savefig('TaylorPade101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.18의 결과와 같다. ■

일반적인 Padé근사를 살펴보기 위해서, 다음 함수들을 생각해보자.

$$p(x) = p_0 + p_1[x - x_0] + \cdots + p_n[x - x_0]^n \tag{1.2.17}$$

$$q(x) = 1 + q_1[x - x_0] + \cdots + q_n[x - x_0]^n \tag{1.2.18}$$

또한, 다음 점들을 생각해보자.

$$x_1 < x_2 < \cdots < x_{2n-1} < x_{2n} \tag{1.2.19}$$

다음 값들을 정의하자.

$$f_0 \doteq f(x_0), \quad f_i \doteq f(x_i), \quad (i = 1, 2, \cdots, 2n-1, 2n) \tag{1.2.20}$$

함수 f 의 Padé근사식 $r(x) = p(x)/q(x)$가 다음 식들을 만족한다고 하자.

$$r_0 = f_0, \quad r(x_i) = f_i, \quad (i = 1, 2, \cdots, 2n-1, 2n) \tag{1.2.21}$$

식 (1.2.21)에서 알 수 있듯이, 다음 연립방정식이 성립한다.

$$\begin{aligned} & p_0 + p_1[x_i - x_0] + \cdots + p_n[x_i - x_0]^n \\ & = f_i\{1 + q_1[x_i - x_0] + \cdots + q_n[x_i - x_0]^n\}, \quad (i = 1, 2, \cdots, 2n-1, 2n) \end{aligned} \tag{1.2.22}$$

다음 식이 성립함은 명백하다.

$$p_0 = f_0 \tag{1.2.23}$$

다음 행렬들과 벡터들을 정의하자.

$$A_1 \doteq \begin{bmatrix} 1 & [x_1 - x_0]^1 & [x_1 - x_0]^2 & \cdots & [x_1 - x_0]^{n-1} \\ 1 & [x_2 - x_0]^1 & [x_2 - x_0]^2 & \cdots & [x_2 - x_0]^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & [x_{2n-1} - x_0]^1 & [x_{2n-1} - x_0]^2 & \cdots & [x_{2n-1} - x_0]^{n-1} \\ 1 & [x_{2n} - x_0]^1 & [x_{2n} - x_0]^2 & \cdots & [x_{2n} - x_0]^{n-1} \end{bmatrix} \tag{1.2.24}$$

$$A_2 \doteq - \begin{bmatrix} f_1 & f_1[x_1 - x_0] & \cdots & [f_1 x_1 - x_0]^{n-1} \\ f_2 & f_2[x_2 - x_0] & \cdots & f_2[x_2 - x_0]^{n-1} \\ \vdots & \vdots & & \vdots \\ f_{2n-1} & f_{2n-1}[x_{2n-1} - x_0] & \cdots & f_{2n-1}[x_{2n-1} - x_0]^{n-1} \\ f_{2n} & f_{2n}[x_{2n} - x_0] & \cdots & f_{2n}[x_{2n} - x_0]^{n-1} \end{bmatrix} \tag{1.2.25}$$

$$\boldsymbol{p} \doteq [p_1,\ p_2,\ \cdots,\ p_n]^t \tag{1.2.26}$$

$$\boldsymbol{q} \doteq [q_1,\ q_2,\ \cdots,\ q_n]^t \tag{1.2.27}$$

$$\boldsymbol{c} \doteq \left[\frac{f_1 - f_0}{x_1 - x_0},\ \frac{f_2 - f_0}{x_2 - x_0},\ \cdots,\ \frac{f_{2n} - f_0}{x_{2n} - x_0}\right]^t \tag{1.2.28}$$

연립방정식 (1.2.22)을 다음과 같이 쓸 수 있다.

$$[A_1\ A_2] \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{q} \end{bmatrix} = \boldsymbol{c} \tag{1.2.29}$$

예제 1.2.20 점 $x = 0$에서 함수 $f(x) = \ln(1 + x)$의 $(2, 2)$차 Padé근사를 구하기 위해서,

다음 멱함수들을 정의하자.

$$p(x) \doteq p_0 + p_1 x + p_2 x^2 \tag{1}$$

$$q(x) \doteq 1 + q_1 x + q_2 x^2 \tag{2}$$

또한, 다음 값들을 선택하자.

$$n = 2, \quad x_1 = \frac{1}{2}, \quad x_2 = 1, \quad x_3 = \frac{3}{2}, \quad x_4 = 2 \tag{3}$$

다음 식들이 성립함은 자명하다.

$$x_0 = 0, \quad p_0 = f_0 = 0 \tag{4}$$

또한, 다음 식들이 성립한다.

$$A_1 = \begin{bmatrix} 1 & \frac{1}{2} \\ 1 & 1 \\ 1 & \frac{3}{2} \\ 1 & 2 \end{bmatrix}, \quad A_2 = -\begin{bmatrix} \ln\frac{3}{2} & \frac{1}{2}\ln\frac{3}{2} \\ \ln 2 & \ln 2 \\ \ln\frac{5}{2} & \frac{3}{2}\ln\frac{5}{2} \\ \ln 3 & 2\ln 3 \end{bmatrix}, \quad \boldsymbol{c} = -\begin{bmatrix} 2\ln\frac{3}{2} \\ \ln 2 \\ \frac{2}{3}\ln\frac{5}{2} \\ \frac{1}{2}\ln 3 \end{bmatrix} \tag{5}$$

식 (1.2.29)에서 알 수 있듯이, 우리의 문제는 다음 연립방정식을 푸는 것으로 귀착된다.

$$[A_1 \; A_2] \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{q} \end{bmatrix} = \boldsymbol{c} \tag{6}$$

이 Padé근사를 구하기 위해서, 다음 MATLAB프로그램 PadeLogApproxi101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename PadeLogApproxi101.m
% Graph of Pade approximation of log(1+x)
% Programmed by CBS
% ------------------------------------------------------------------------
function PadeLogApproxi101
[p,q] = PadeCoefficient(@(x) log(1+x),0,2,2)
xx = -0.98:0.001:2;
yy = log(1+xx);
fa = log(1+0);
tt = xx.*(1 - xx.*( 1/2 - xx.*( 1/3 - xx.*(1/4 - 1/5*xx))));
PP = ( fa + p(1).*xx + p(2)*xx.^2 )./(1 + q(1)*xx + q(2)*xx.^2 ) ;
```

```
% plot
plot(xx,yy,'r',xx,tt,'b--',xx,PP,'k-.','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('log(1+x)','Taylor','Pade','location','NW')
xlabel('{\bf x}','fontsize',12)
ylabel('\bf y','FontSize',12,'rotation',0)
saveas(gcf,'PadeLogApproxi101','jpg')
end
% End of program
% ------------------------------------------------------------------------
% Filename PadeCoefficient.m
% Pade approximation of f(x) on [ x0,xb]
% Programmed by CBS
% ------------------------------------------------------------------------
function [p,q] = PadeCoefficient(fun,x0,xb,nterm)
deltax = (xb - x0)/(2*nterm);
x = [ (x0+deltax):deltax:xb ]';
f = fun(x);  f0 = fun(x0);
xx = x - x0;
cc = (f - f0)./xx;
A = zeros(2*nterm,2*nterm);
for ii=1:nterm
    A(:,ii) = xx.^(ii-1);
    A(:,nterm+ii) = - f.*xx.^(ii-1);
end
Ainv = inv(A);
p = Ainv(1:nterm,:)*cc;
q = Ainv( (nterm+1):(2*nterm),: )*cc;
% Use a generalized inverse
% pq = A\cc
% p = pq(1:nterm,:)
% q = pq( (nterm+1):(2*nterm),: )
if abs(det(A)) < 1.e-10
    disp('Warning: Matrix is near singular')
    disp('determinant is'), disp(det(A))
end
end
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 PadeLogApproxi101.m을 실행되면, 이 연립방정식의 해가 다음과 같음을 알 수 있다.

$$\boldsymbol{p} = \begin{bmatrix} 0.998518 \\ 0.303034 \end{bmatrix}, \quad \boldsymbol{q} = \begin{bmatrix} 0.794928 \\ 0.082814 \end{bmatrix} \tag{7}$$

즉, 함수 $f(x) = \ln(1+x)$의 $(2,2)$차 Padé근사식은 다음과 같다.

$$r(x) = \frac{0.998518x + 0.303034x^2}{1 + 0.794928x + 0.082814x^2} \tag{8}$$

또한, 이 MATLAB프로그램이 실행되면, 함수 $f(x)$, Taylor근사식 $t(x)$ 그리고 Padé근사식 $r(x)$를 비교하기 위한 그림 1.2.8이 그려진다. 그림 1.2.8에서 알 수 있듯이, 구간 $[-0.5, 2]$에서 함수 $f(x)$와 Padé근사식 $r(x)$의 차이는 거의 없다. 반면에, x가 커질수록 5차 Taylor

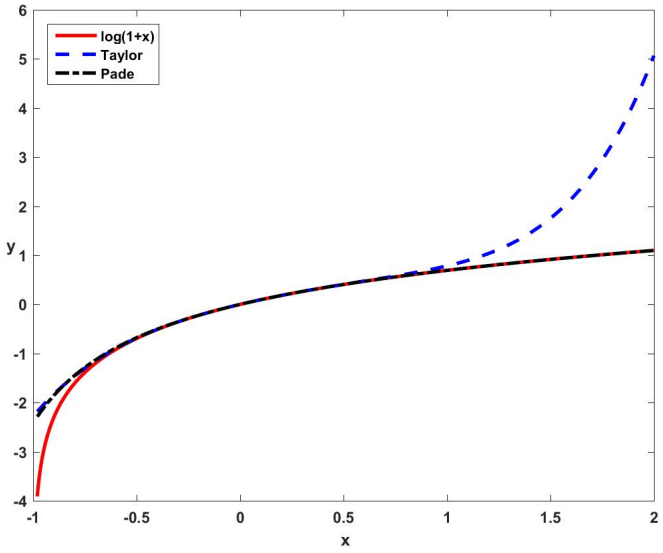


그림 1.2.8. Padé근사식 1

근사식 $t(x)$는 함수 $f(x)$로부터 멀어진다. ■

예제 1.2.21 Python을 사용해서 예제 1.2.20을 다시 다루기 위해서, 다음 Python프로그램 PadeLogApproxi101.Py를 실행해 보자.

```
"""
Created on Thu Jan 12 20:13:46 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

def PadeCoefficient(x0,xb,nterm):
    deltax = (xb - x0)/(2*nterm)
    x = np.arange((x0+deltax), xb+deltax, deltax)
    f = np.log(1+x)
    f0 = np.log(1+x0)
    xx = x - x0;
    cc = (f - f0)/xx
    A = np.zeros((2*nterm,2*nterm),float)
    for ii in range(1,(2*nterm+1)):
        for jj in range(1,(nterm+1)):
            A[ii-1,jj-1] = xx[ii-1]**(jj-1)
            A[ii-1,nterm+jj-1] = - f[ii-1]*xx[ii-1]**(jj-1)
    Ainv = np.linalg.inv(A)
    p = np.dot(Ainv[0:(nterm),], cc)
    q = np.dot(Ainv[ (nterm):(2*nterm),], cc)
#    ppqq = np.dot(Ainv,cc)
#    pp = ppqq[0:nterm]          # which equals p
#    qq = ppqq[nterm:(2*nterm)] # which equals q
#    print(pp)
#    print(qq)
    if abs(np.linalg.det(A)) < 10**(-10):
```

```
        print('Warning: Matrix is near singular')
        print('determinant is'), print(np.linalg.det(A))
    return(p,q)

[p,q] = PadeCoefficient(0,2,2)
xx = np.arange(-0.98, 2.001, 0.001)
yy = np.log(1+xx)
fa = np.log(1+0)
tt = xx*(1 - xx*( 1/2 - xx*( 1/3 - xx*(1/4 - 1/5*xx))))
PPnum = fa + p[0]*xx + p[1]*xx**2
PPdenom = 1 + q[0]*xx + q[1]*xx**2
PP = PPnum/PPdenom
# PP = np.divide(( fa + p[0]*xx + p[1]*xx**2 ),(1 + q[0]*xx + q[1]*xx**2 ))

# Plotting
fig, axes = plt.subplots(nrows=1, ncols=1)
plt.plot(xx,yy,'r', lw=3, label='log(1+x)')
plt.plot(xx,tt,'b--', lw=3, label='Taylor')
plt.plot(xx,PP,'k-.', lw=3, label='Pade')
plt.axis([-1, 2, -4, 6])
plt.legend(loc='upper left', numpoints = 1 )
plt.show()
fig.savefig('PadeLogApproxi101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.20의 결과와 같다. ■

일반적으로 말하면, Taylor근사보다 Padé근사가 더 좋은 근사식을 제공한다. 다음 명제는 Cuyt & Wuytack [15, pp. 97-99]에서 인용한 것이다.

명제 1.2.3

정의역이 집합 $B = \{z \,|\, |z| \le r\} (\subset C)$인 유리형함수(meromorphic funtion) f의 (m,n)차 Padé근사식을 $P_{m,n}$이라면, 임의의 $\delta(>0)$와 $\epsilon(>0)$에 대해서 다음 명제를 만족하는 자연수 k가 존재한다.

$$j > k \;\Rightarrow\; m(\{|\, f(z) - P_{j,1}(z) \,| > \epsilon\}) < \delta$$

여기서 $m(\cdot)$은 Lebesgue측도이다. 또한, 만약 함수 f가 유한개 극점(pole)들을 갖고 이들의 중복도들의 합이 n이면, Padé근사식 $\lim_{j\to\infty} P_{j,n}$은 극점을 포함하지 않는 임의의 컴팩트집합 $E(\subset B)$에서 일양수렴한다.

예제 1.2.22 함수 $f(x) = x^{1/4}$의 (m,n)차 Padé근사식을 $P_{m,n}(x)$라 하고, 다음 집합을

정의하자.

$$E(r_1, r_2) \doteq \{z \mid |z| \leq r_2\} \backslash \{z \mid |z| \leq r_1\} \tag{1}$$

함수 f의 특이점은 $x = 0$뿐이고, 또한 이 극점의 중복도는 1이다. 명제 1.2.3에서 알 수 있듯이, 각 $r_1 < r_2 (< \infty)$에 대해서 다음 명제가 성립한다.

$$z \in E(r_1, r_2) \;\Rightarrow\; \lim_{j \to \infty} | f(z) - P_{j,1}(z) | = 0 \tag{2}$$

즉, 임의의 점 $x (\in (0, \infty))$에 대해서, 다음 식이 성립한다.

$$\lim_{j \to \infty} P_{j,1}(x) = f(x) \tag{3}$$

또한, 임의의 유한구간 $[a, b]$에서 식 (3)의 극한은 일양수렴한다. ■

예제 1.2.23 Padé근사를 사용해서 예제 1.2.14의 문제를 풀어보자. 즉, 다음 집합에 속한 5개 점들을 지나는 $(2, 2)$차 Padé근사식을 구해보자.

$$I \doteq \{(10^{-1}, 10^{-1/4}), (10^0, 10^0), (10^1, 10^{1/4}), (10^2, 10^{1/2}), (10^3, 10^{3/4})\} \tag{1}$$

다음 함수들을 정의하자.

$$p(x) \doteq p_0 + p_1[x - 1] + p_2[x - 1]^2 \tag{2}$$

$$q(x) \doteq 1 + q_1[x - 1] + q_2[x - 1]^2 \tag{3}$$

다음 식들이 성립함은 자명하다.

$$p_0 = f_0 = 1, \quad x_0 = 1 \tag{4}$$

또한, 다음 식들이 성립한다.

$$A_1 = \begin{bmatrix} 1 & [10^{-1}-1] \\ 1 & [10-1] \\ 1 & [10^2-1] \\ 1 & [10^3-1] \end{bmatrix}, \quad A_2 = - \begin{bmatrix} 10^{-1/4} & 10^{-1/4}[10^{-1}-1] \\ 10^{1/4} & 10^{1/4}[10-1] \\ 10^{2/4} & 10^{2/4}[10^2-1] \\ 10^{3/4} & 10^{3/4}[10^3-1] \end{bmatrix} \tag{5}$$

$$\boldsymbol{c} = \left[\frac{10^{-1/4}-1}{10^{-1}-1}, \frac{10^{1/4}-1}{10-1}, \frac{10^{1/2}-1}{10-1}, \frac{10^{1/2}-1}{10^2-1}, \frac{10^{3/4}-1}{10^3-1} \right]^t \tag{6}$$

식 (1.2.29)에서 알 수 있듯이, 우리의 문제는 다음 연립방정식을 푸는 것으로 귀착된다.

$$[A_1 \; A_2] \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{q} \end{bmatrix} = \boldsymbol{c} \tag{7}$$

연립방정식 (7)을 풀기 위해서, 다음 MATLAB프로그램 PadeInter101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename PadeInter101.m
% Pade interpolation 1 based on  given dataset
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
% Pade coefficients
nterm = 2;
x0 = 1; f0 = 1;
x = [ 10^-1 10^1 10^2 10^3 ]';
f = [ 10^(-1/4) 10^(1/4) 10^(2/4) 10^(3/4) ]';
xx = x - x0;
cc = (f - f0)./xx;
A = zeros(2*nterm,2*nterm);
for ii=1:nterm
    A(:,ii) = xx.^(ii-1);
    A(:,nterm+ii) = - f.*xx.^(ii-1);
end
format long
A
Ainv = inv(A);
p = Ainv(1:nterm,:)*cc
q = Ainv( (nterm+1):(2*nterm),: )*cc
if abs(det(A)) < 1.e-10
    disp('Warning: Matrix is near singular')
    disp('determinant is'), disp(det(A))
end
% plot Pade approximation
xdum = logspace(-2,3,1000);
ydum = xdum.^(1/4);
xxdum = xdum - x0;
PP = ( f0 + p(1).*xxdum + p(2)*xxdum.^2 )...
      ./(1 + q(1)*xxdum + q(2)*xxdum.^2 ) ;
plot(xdum,ydum,'b-.',x,f,'rd',xdum,PP,'k-','LineWidth',2.5)
```

```
set(gca,'fontsize',11,'fontweigh','bold')
legend('function','Data Point','Pade','location','SE')
xlabel('{\bf x}','fontsize',12)
ylabel('\bf y','FontSize',12,'rotation',0)
saveas(gcf,'PadeInter101','jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 PadeInter101.m이 실행되면, 연립방정식의 해가 다음과 같음을 알 수 있다.

$$\boldsymbol{p} = \begin{bmatrix} 0.716904 \\ 0.010800 \end{bmatrix}, \quad \boldsymbol{q} = \begin{bmatrix} 0.394303 \\ 0.001653 \end{bmatrix} \tag{8}$$

즉, 함수 $f(x) = x^{1/4}$ 의 $(2,2)$차 Padé근사식은 다음과 같다.

$$P_{2,2}(x) = \frac{1 + 0.716904x + 0.010800x^2}{1 + 0.394303x + 0.001653x^2} \tag{9}$$

또한, 이 MATLAB명령문이 실행되면, 함수 $f(x)$와 Taylor근사식 $t(x)$을 비교하기 위한 그림 1.2.9가 그려진다. 그림 1.2.5의 Taylor근사식과는 달리, 그림 1.2.9의 Padé근사식 $P_{2,2}(x)$는 함수 $f(x)$에 아주 가깝다. ■

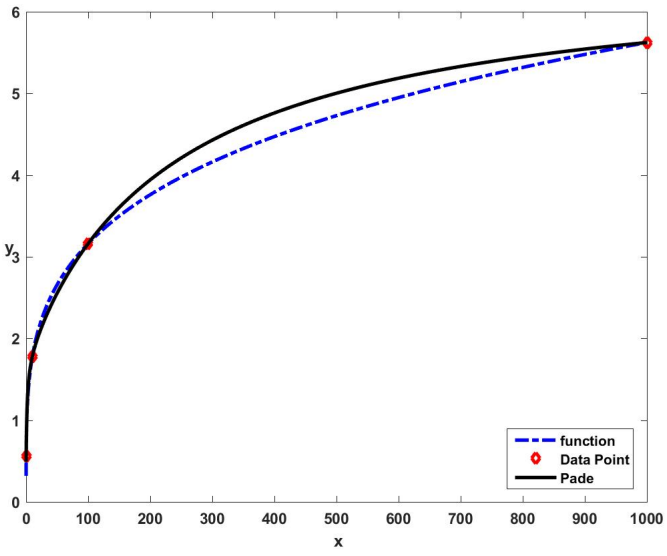


그림 1.2.9. Padé근사식 2

예제 1.2.24 Python을 사용해서 예제 1.2.23을 다시 다루기 위해서, 다음 Python프로그램

PadeInter101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

nterm = 2
x0 = 1
f0 = 1
x = np.array([ 10**(-1), 10**(1), 10**(2), 10**(3) ])
f = np.array([ 10**(-1/4), 10**(1/4), 10**(2/4), 10**(3/4) ])
xx = x - x0
cc = (f - f0)/xx

A = np.zeros((2*nterm,2*nterm),float)
for ii in range(1,(2*nterm+1)):
    for jj in range(1,(nterm+1)):
        A[ii-1,jj-1] = xx[ii-1]**(jj-1)
        A[ii-1,nterm+jj-1] = - f[ii-1]*xx[ii-1]**(jj-1)
Ainv = np.linalg.inv(A)
p = np.dot(Ainv[0:(nterm),], cc)
q = np.dot(Ainv[ (nterm):(2*nterm),], cc)
if abs(np.linalg.det(A)) < 10**(-10):
   print('Warning: Matrix is near singular')
   print('determinant is'), print(np.linalg.det(A))

# Plotting
xdum = np.logspace(-2, 3, num=1000)
ydum = xdum**(1/4)
xxdum = xdum-x0
PP = ( f0 + p[0]*xxdum + p[1]*xxdum**2 )/(1 + q[0]*xxdum + q[1]*xxdum**2 )
fig, axes = plt.subplots(nrows=1, ncols=1)
plt.plot(xdum,ydum,'b-.', label='function')
plt.plot(x,f,'rd', label='Data Point')
plt.plot(xdum,PP,'k-', label='Pade')
plt.axis([ 0, 1000, 0, 6 ])
plt.legend(loc='lower right', numpoints = 1 )
plt.show()
fig.savefig('PadeInter101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.23의 결과와 같다. ■

예제 1.2.25 앞에서 언급했듯이, 주어진 데이터세트에 적합한 근사식을 구할 때는 주어진 데이터세트에 적합한 변수변환을 해야 한다. 예제 1.2.16에서 변수 x를 $\log_{10} x$로 변환해서, 훨씬 더 적합한 근사식을 구할 수 있었다. 변수 x를 $\log_{10} x$로 그리고 변수 y를 $\log_{10} y$로 변환한 다음 Padé근사식을 구하기 위해서, 다음 MATLAB프로그램 PadeInter102.m을 실행하자.

```
% --------------------------------------------------------------------
% Filename: PadeInter102.m
```

```
% Pade interpolation 2 based on  given dataset w/ Log-Log transform
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
% Pade coefficients
nterm = 2;
x0 = 1; f0 = 1;
logx0 = log10(x0); logf0 = log10(f0);
x = [ 10^-1 10^1 10^2 10^3 ]';
logx = log10(x);
f = [ 10^(-1/4) 10^(1/4) 10^(2/4) 10^(3/4) ]';
logf = log10(f);
xx = logx - logx0;
cc = (logf - logf0)./xx;
A = zeros(2*nterm,2*nterm);
for ii=1:nterm
    A(:,ii) = xx.^(ii-1);
    A(:,nterm+ii) = - f.*xx.^(ii-1);
end
Ainv = inv(A);
p = Ainv(1:nterm,:)*cc
q = Ainv( (nterm+1):(2*nterm),: )*cc
if abs(det(A)) < 1.e-10
    disp('Warning: Matrix is near singular')
    disp('determinant is'), disp(det(A))
end
% plot Pade approximation
xdum =  linspace(-2/log(10),4/log(10),1000)*log(10);
ydum = 1/4*xdum;
xxdum = xdum - logx0;
PP = ( logf0 + p(1).*xxdum + p(2)*xxdum.^2 )...
      ./(1 + q(1)*xxdum + q(2)*xxdum.^2 ) ;
plot(xdum,ydum,'b-.',logx,logf,'rd',xdum,PP,'k-','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('function','Data Point','Pade','location','SouthEast')
xlabel('{\bf x}','fontsize',12)
ylabel('\bf y','FontSize',12,'rotation',0)
saveas(gcf,'PadeInter102','jpg')
% End of program
% ------------------------------------------------------------------------
```

MATLAB프로그램 PadeInter102.m이 실행하면, 다음과 같은 Padé계수벡터들를 얻는다.

$$\boldsymbol{p} = \begin{bmatrix} 0.250000 \\ -0.000000 \end{bmatrix}, \quad \boldsymbol{q} = \begin{bmatrix} -0.0000000 \\ 0.000000 \end{bmatrix} \tag{1}$$

즉, 함수 $f(x) = x^{1/4}$ 의 $(2,2)$차 Padé근사식은 다음과 같다.

$$\log_{10} P_{2,2}(x) = \frac{1 + 0.250000 \log_{10} x - 0.000000[\log_{10} x]^2}{1 - 0.000000 \log_{10} x + 0.000000[\log_{10} x]^2} \tag{2}$$

이 MATLAB프로그램이 실행되면, 함수 $\log_{10} f(x)$와 근사식 $\log_{10} P_{2,2}(x)$를 비교하기 위한 그림 1.2.10이 그려진다. 그림 1.2.10에서 알 수 있듯이, 근사식 $\log_{10} P_{2,2}(x)$는 함수 $\log_{10} f(x)$

와 거의 같다. 할인곡선 대신 일드곡선을 사용하는 것은 이러한 로그변환을 이용하는 것임을 기억하라. ■

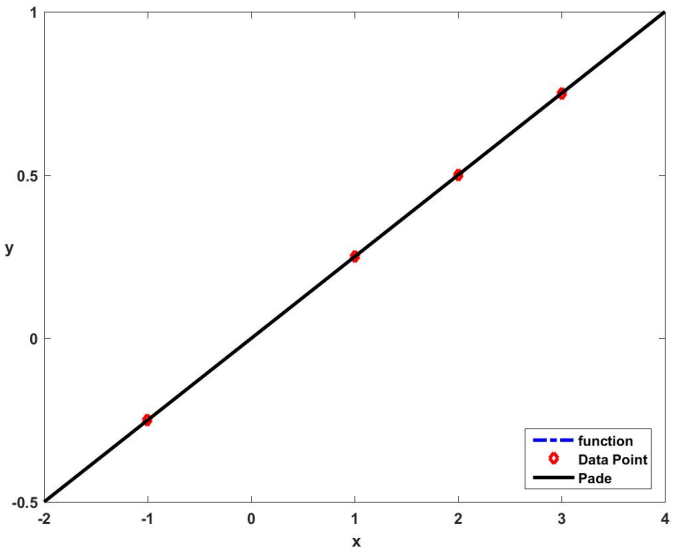


그림 1.2.10. Padé근사식 3

예제 1.2.26 Python을 사용해서 예제 1.2.25을 다시 다루기 위해서, 다음 Python프로그램 PadeInter102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

nterm = 2
x0 = 1
f0 = 1
logx0 = np.log10(x0)
logf0 = np.log10(f0)
x = np.array([ 10**(-1), 10**(1), 10**(2), 10**(3) ])
logx = np.log10(x)
f = np.array([ 10**(-1/4), 10**(1/4), 10**(2/4), 10**(3/4) ])
logf = np.log10(f)
xx = logx - logx0
cc = (logf - logf0)/xx
A = np.zeros((2*nterm,2*nterm),float)
for ii in range(1,(2*nterm+1)):
    for jj in range(1,(nterm+1)):
        A[ii-1,jj-1] = xx[ii-1]**(jj-1)
        A[ii-1,nterm+jj-1] = - f[ii-1]*xx[ii-1]**(jj-1)
Ainv = np.linalg.inv(A)
p = np.dot(Ainv[0:(nterm),], cc)
q = np.dot(Ainv[ (nterm):(2*nterm),], cc)
```

```
if abs(np.linalg.det(A)) < 10**(-10):
   print('Warning: Matrix is near singular')
   print('determinant is'), print(np.linalg.det(A))

# Plotting
xdum = np.linspace(-2/np.log(10),4/np.log(10),num=1000)*np.log(10)
ydum = 0.25*xdum
xxdum = xdum - logx0
PP = ( logf0 + p[0]*xxdum + p[1]*xxdum**2 )/(1 + q[0]*xxdum + q[1]*xxdum**2
    )
fig, axes = plt.subplots(nrows=1, ncols=1)
plt.plot(xdum,ydum,'b-.', lw=3, label='function')
plt.plot(logx,logf,'rd', lw=3, label='Data Point')
plt.plot(xdum,PP,'k-', label='Pade')
plt.axis([ -2, 4, -0.5, 1 ])
plt.xlabel('x'), plt.ylabel('y')
plt.legend(loc='lower right', numpoints = 1 )
plt.show()
fig.savefig('PadeInter102Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.25의 결과와 같다. ■

1.2.4 다변수함수의 Taylor전개

벡터 $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]^t$에 대한 함수 $f(\boldsymbol{x}) = f(x_1, x_2, \cdots, x_n)$의 점 $\boldsymbol{a} = [a_1, a_2, \cdots, a_n]^t$ 주변에서 Taylor전개는 다음과 같다.

$$f(\boldsymbol{x}) = f(\boldsymbol{a}) + \sum_{j=1}^{n} f_j(\boldsymbol{a})[x_j - a_j] + \frac{1}{2}\sum_{j=1}^{n}\sum_{k=1}^{n} f_{jk}(\boldsymbol{a})[x_j - a_j][x_k - a_k] + \cdots \tag{1.2.30}$$

함수 $f(\boldsymbol{x})$의 벡터 $\boldsymbol{a}$에서 기울기벡터(gradient) $\nabla f(\boldsymbol{a})$와 Hessian행렬 $H(\boldsymbol{a})$를 각각 다음과 같이 표기하자.

$$\nabla f \doteq \nabla f(\boldsymbol{a}) \doteq \begin{bmatrix} f_1(\boldsymbol{a}) \\ f_2(\boldsymbol{a}) \\ \vdots \\ f_n(\boldsymbol{a}) \end{bmatrix}, \quad H \doteq H(\boldsymbol{a}) \doteq \begin{bmatrix} f_{11}(\boldsymbol{a}) & f_{12}(\boldsymbol{a}) & \cdots & f_{1n}(\boldsymbol{a}) \\ f_{21}(\boldsymbol{a}) & f_{22}(\boldsymbol{a}) & \cdots & f_{2n}(\boldsymbol{a}) \\ \vdots & \vdots & & \vdots \\ f_{n1}(\boldsymbol{a}) & f_{n2}(\boldsymbol{a}) & \cdots & f_{nn}(\boldsymbol{a}) \end{bmatrix} \tag{1.2.31}$$

식 (1.2.30)의 Taylor전개를 다음과 같이 쓸 수 있다.

$$f(\boldsymbol{x}) = f(\boldsymbol{a}) + \nabla f(\boldsymbol{a})^t[\boldsymbol{x} - \boldsymbol{a}] + \frac{1}{2}[\boldsymbol{x} - \boldsymbol{a}]^t H(\boldsymbol{a})[\boldsymbol{x} - \boldsymbol{a}] + \cdots \tag{1.2.32}$$

만약 노름(norm $||\boldsymbol{x}-\boldsymbol{a}||$가 충분히 작다면, 식 (1.2.32)에서 알 수 있듯이 함수 $f(\boldsymbol{x})$의 벡터 $\boldsymbol{a}$에서 1차 근사식은 다음과 같다.

$$f(\boldsymbol{x}) \approx f(\boldsymbol{a}) + \nabla f(\boldsymbol{a})^t[\boldsymbol{x}-\boldsymbol{a}] \tag{1.2.33}$$

식 (1.2.33)로부터 다변수함수의 근을 구하는 Newton-Raphson법을 유도할 수 있다. 또한, 함수 $f(\boldsymbol{x})$의 벡터 $\boldsymbol{a}$에서 2차 근사식은 다음과 같다.

$$f(\boldsymbol{x}) \approx f(\boldsymbol{a}) + \nabla f(\boldsymbol{a})^t[\boldsymbol{x}-\boldsymbol{a}] + \frac{1}{2}[\boldsymbol{x}-\boldsymbol{a}]^t H(\boldsymbol{a})[\boldsymbol{x}-\boldsymbol{a}] \tag{1.2.34}$$

예제 1.2.27 MATLAB의 Symbolic Math Toolbox의 함수들을 사용해서 2변수함수의 근사식을 구하기 위해, 다음 MATLAB프로그램 MultiApproxi101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename MultiApproxi101.m
% Approximation of a bivariate function sqrt(x^2 + y^2)
% Programmed by CBS
% ------------------------------------------------------------------------
syms x y
S = sqrt(x^2 + y^2);
% Partial derivatives
Sx = diff(S,x), Sy = diff(S,y)
Sxx = diff(Sx,x), Syy = diff(Sy,y), Sxy = diff(Sx,y)
a = 3; b = 4;
Sab = subs(subs(S,x,a),y,b)
grad(1) = subs(subs(Sx,x,a),y,b);
grad(2) = subs(subs(Sy,x,a),y,b);
H(1,1) = subs(subs(Sxx,x,a),y,b);
H(2,2) = subs(subs(Syy,x,a),y,b);
H(1,2) = subs(subs(Sxy,x,a),y,b);
H(2,1) = H(1,2);
grad, H
% 1st-order approximation at (a,b)
Fapproxi = Sab + grad(1)*(x-a) + grad(2)*(y-b)
% 2nd-order approximation at (a,b)
dum2 = 1/2*H(1,1)*(x-a)^2 + H(1,2)*(x-a)*(y-b) + 1/2*H(2,2)*(y-b)^2
Sapproxi = Fapproxi + dum2
Sapproxi = simplify(Sapproxi)
% Approximate values at (a+eps1,b+eps1)
eps1 = 0.1, eps2 = -0.01
ApproxiF =  subs(subs(Fapproxi,x,a+eps1),y,b+eps2)
ApproxiS =  subs(subs(Sapproxi,x,a+eps1),y,b+eps2)
TrueValue = subs(subs(S,x,a+eps1),y,b+eps2)
errorF = TrueValue - ApproxiF
errorS = TrueValue - ApproxiS
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 MultiApproxi1011.m은 다음 2변수함수의 점 $[a,b]=[3,4]$에서 1

차 근사식과 2차 근사식을 구하기 위한 것이다.

$$S = \sqrt{x^2 + y^2} \tag{1}$$

이 MATLAB프로그램을 실행하면, 편도함수들이 다음과 같음을 알 수 있다.

$$S_x = \frac{x}{\sqrt{x^2 + y^2}}, \quad S_y = \frac{y}{\sqrt{x^2 + y^2}} \tag{2}$$

$$S_{xx} = \frac{1}{[x^2 + y^2]^{1/2}} - \frac{x^2}{[x^2 + y^2]^{3/2}} \tag{3}$$

$$S_{yy} = \frac{1}{[x^2 + y^2]^{1/2}} - \frac{y^2}{[x^2 + y^2]^{3/2}} \tag{4}$$

$$S_{xy} = -\frac{xy}{[x^2 + y^2]^{3/2}} \tag{5}$$

식 (1)∼식 (5)에 점 $[a, b] = [3, 4]$을 대입하면, 다음 식들이 성립함을 알 수 있다.

$$S(3, 4) = 5, \quad \nabla S = \begin{bmatrix} 0.6000 \\ 0.8000 \end{bmatrix}, \quad H = \begin{bmatrix} 0.1280 & -0.0960 \\ -0.0960 & 0.0720 \end{bmatrix} \tag{6}$$

따라서, 2변수함수 S의 점 $[a, b] = [3, 4]$에서 1차 근사식과 2차 근사식은 각각 다음과 같다.

$$S^F(x, y) \approx \frac{3}{5}x + \frac{4}{5}y \tag{7}$$

$$S^S(x, y) \approx \frac{3}{5}x + \frac{4}{5}y + \frac{8}{125}[x - 3]^2 - \frac{12}{125}[x - 3][y - 4] + \frac{9}{250}[y - 4]^2 \tag{8}$$

점 $(3.1, 3.99)$에서 2변수함수 S의 진짜값과 근사값들은 각각 다음과 같다.

$$S(3.1, 3.99) = 5.052737934310389 \tag{9}$$

$$S^F(3.1, 3.99) \approx 5.0520000 \tag{10}$$

$$S^S(3.1, 3.99) \approx 5.0527396 \tag{11}$$

따라서, 1차 근사식의 오차는 $7.3 \cdot 10^{-4}$이고 2차 근사식의 오차는 $-7.7 \cdot 10^{-6}$이다. ■

예제 1.2.28 Python을 사용해서 예제 1.2.27을 다시 다루기 위해서, 다음 Python프로그램 MultiApproxi101.Py를 실행해 보자.

```
"""
Created on Wed Jan 11 22:13:11 2017
@author: CBS
"""
import numpy as np
from sympy import *
x = Symbol('x')
y = Symbol('y')
S = sqrt(x**2 + y**2)
# Partial derivatives
Sx = diff(S,x); Sy = diff(S,y)
Sxx = diff(Sx,x); Syy = diff(Sy,y); Sxy = diff(Sx,y)
a = 3; b = 4
Sa = S.subs(x,a); Sab = Sa.subs(y,b)
grad = np.zeros((2,1))
g0 = Sx.subs(x,a); grad[0] = g0.subs(y,b)
g1 = Sy.subs(y,b); grad[1] = g1.subs(x,a)
H = np.zeros((2,2))
h00 = Sxx.subs(x,a); H[0,0] = h00.subs(y,b)
h01 = Sxy.subs(x,a); H[0,1] = h01.subs(y,b)
h11 = Syy.subs(x,a); H[1,1] = h11.subs(y,b)
H[1,0] = H[0,1]

#  1st-order approximation at (a,b)
Fapproxi = Sab + grad[0]*(x-a) + grad[1]*(y-b)
Fapproxi = np.take(Fapproxi,0)  # Make a scalar
dum2 = 1/2*H[0,0]*(x-a)**2 + H[0,1]*(x-a)*(y-b) + 1/2*H[1,1]*(y-b)**2
Sapproxi = Fapproxi + dum2
Sapproxi = simplify(Sapproxi)
Sapproxi = np.take(Sapproxi,0) # Make a scalar

# Approximate values at (a+eps1,b+eps1)
eps1 = 0.1; eps2 = -0.01
TrueValue0 = S.subs(x,a+eps1); TrueValue = TrueValue0.subs(y,b+eps2)
AF1 = Fapproxi.subs(x,a+eps1); ApproxiF = AF1.subs(y,b+eps2)
AS1 = Sapproxi.subs(x,a+eps1); ApproxiS = AS1.subs(y,b+eps2)
errorF = TrueValue - ApproxiF
errorS = TrueValue - ApproxiS

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.2.27의 결과와 같다. ■

제1.3절 컴퓨터의 함정

컴퓨터는 아주 유용한 도구이지만, 정밀성을 요구하는 문제를 푸는 과정에서 일반인이 생각하는 정도로 적당히 컴퓨터를 사용하면 매우 이상한 결과를 얻을 수도 있다. 다음 MATLAB 프로그램 ComputerError101.m을 실행해보자.

```
% --------------------------------------------------------------------
% Filename ComputerError101.m
% Strange results due to Computer Structure
```

```
% Programmed by CBS
% ------------------------------------------------------------------------
% Case 1
falsetrue = {'false', 'true'}
P11 = (0.1 + 0.1 + 0.1 + 0.1) - 0.4
P12 = (0.1 + 0.1 + 0.1 + 0.1) == 0.4
P12string = falsetrue(P12+1)
P13 = (0.1 + 0.1 + 0.1) == 0.3
P13string = falsetrue(P13+1)
P14 = (0.1 + 0.1 + 0.1) - 0.3
P15 = 0.1 + 0.1 + (0.1 - 0.3)
P16 = 0.1 + (0.1 + 0.1) - 0.3
P17 = (0.1 + 0.1) + 0.1 - 0.3
P18 = 3*0.1 - 0.3
P21 = 0.1*0.1 - 0.01
P22 = 7/3 - 4/3 - 1
P23 = (7/3 -4/3 - 1) == 0
P23string = falsetrue(P23+1)
% Case 2
a = 12345.678, b = a*10^(-8), c = 1/3
P31 = (a+b)*c - (a*c+b*c)
P32 = (a+b)*c == (a*c+b*c)
P32string = falsetrue(P32+1)
% Case 3
a = 1; b = 2*10^8; c = 1;
x1 = (-b-sqrt(b^2-4*a*c))/2/a
A1 = (a*x1^2+b*x1+c == 0)
A1string = falsetrue(A1+1)
A2 = a*x1^2+b*x1+c
x2 = (-b+sqrt(b^2-4*a*c))/2/a
B1 = (a*x2^2+b*x2+c == 0)
B1string = falsetrue(B1+1)
B2 = a*x2^2+b*x2+c
x3 = 1/x1
C1 = (a*x3^2+b*x3+c == 0)
C1string = falsetrue(C1+1)
C2 = a*x3^2+b*x3+c
x4 = b*( -1/2*(4*a*c/b^2)-1/8*(4*a*c/b^2)^2 )/2/a
D1 = (a*x4^2+b*x4+c == 0)
D1string = falsetrue(D1+1)
D2 = a*x4^2+b*x4+c
% Case 4
a = 10^8
z1 = log(a-sqrt(a^2-1))
z2 = -log(a+sqrt(a^2-1))
z3 = log(1/2/a+1/8/a^3)
% Case 5
theta1 = pi/2*(0:22); tann = tan(theta1)
subplot(2,1,1)
plot(theta1,tann,'kd','linewidth',2)
hold on
set(gca,'fontsize',11,'fontweigh','bold')
legend('tan \theta','location','SouthWest')
plot(theta1,tann,'g-',theta1,tann,'kd','linewidth',2)
hold off
theta2 = pi*(0:22); sinn = sin(theta2)
subplot(2,1,2)
plot(theta2,sinn,'r*','linewidth',2)
hold on
set(gca,'fontsize',11,'fontweigh','bold')
legend('sin \theta','location','SouthWest')
```

```
plot(theta2,sinn,'g-',theta2,sinn,'r*','linewidth',2)
hold off
saveas(gcf,'ComputerError101.jpg')
% End of Program
% --------------------------------------------------------------------
```

첫 번째 경우에서 변수 P11은 다음 식을 계산하기 위한 것이다.

$$\mathrm{P11} = (0.1 + 0.1 + 0.1 + 0.1) - 0.4 \tag{1.3.1}$$

답은 당연히 0이다. 변수 P12는 다음 비교연산을 하기 위한 것이다.

$$\mathrm{P12} = (0.1 + 0.1 + 0.1 + 0.1) == 0.4 \tag{1.3.2}$$

이 식을 다음과 같이 쓰면 이해가 더 빠를 것 같다.

$$\mathrm{P12} = ((0.1 + 0.1 + 0.1 + 0.1) == 0.4) \tag{1.3.3}$$

이 식은 $0.1 + 0.1 + 0.1 + 0.1$가 0.4와 같으면 변수 P12에 1을 할당하고, 그렇지 않으면 0을 할당하라는 것이다. 당연히 P12는 1이다. 변수 P13은 다음 비교연산을 하기 위한 것이다.

$$\mathrm{P13} = (0.1 + 0.1 + 0.1) == 0.3 \tag{1.3.4}$$

이 식은 $0.1 + 0.1 + 0.1$가 0.3과 같으면 변수 P13에 1을 할당하고, 그렇지 않으면 0을 할당하라는 것이다. 당연히 P13은 1일 것 같다. 그러나 MATLAB이 제시하는 P13은 0이다. 즉, $0.1 + 0.1 + 0.1$은 0.3과 같지 않다. "우째 이런 일이?" 그렇다면 컴퓨터의 산수 실력이 초등학교생만도 못하다는 말인가? 변수 P13에 1이 아닌 0이 할당되는 이유는 컴퓨터에서 0.1을 정확히 나타낼 수 없기때문이다. 변수 P14는 다음 식을 계산하는 것이다.

$$\mathrm{P14} = (\ 0.1 + 0.1 + 0.1) - 0.3 \tag{1.3.5}$$

MATLAB이 제시하는 답은 0이 아닌 $5.5511 \cdot 10^{-17}$이다. 변수 P15는 다음 식을 계산하는 것이다.

$$\mathrm{P15} = 0.1 + 0.1 + (0.1 - 0.3) \tag{1.3.6}$$

MATLAB이 제시하는 답은 0이 아닌 $2.7756 \cdot 10^{-17}$이다. 변수 P16은 다음 식을 계산하는

것이다.

$$\text{P16} = 0.1 + (0.1 + 0.1) - 0.3 \tag{1.3.7}$$

MATLAB이 제시하는 답은 0이 아닌 $5.5511 \cdot 10^{-17}$이다. 변수 P17은 다음 식을 계산하는 것이다.

$$\text{P17} = (\ 0.1 + 0.1) + 0.1 - 0.3 \tag{1.3.8}$$

MATLAB이 제시하는 답은 0이 아닌 $5.5511 \cdot 10^{-17}$이다. 지금까지 결과에서 알 수 있듯이, 컴퓨터연산에서는 결합법칙이 성립하지 않는다. 변수 P18은 다음 식을 계산하는 것이다.

$$\text{P18} = 3*0.1 - 0.3 \tag{1.3.9}$$

이 경우에도 MATLAB이 제시하는 답은 0이 아닌 $5.5511 \cdot 10^{-17}$이다. 변수 P21은 다음 식을 계산하는 것이다.

$$\text{P21} = 0.1*0.1 - 0.01 \tag{1.3.10}$$

MATLAB이 제시하는 답은 0이 아닌 $1.7347 \cdot 10^{-18}$이다. 변수 P22는 다음 식을 계산하는 것이다.

$$\text{P22} = 7/3 - 4/3 - 1 \tag{1.3.11}$$

MATLAB이 제시하는 답은 0이 아닌 $2.2204 \cdot 10^{-16}$이다. 변수 P23은 다음 비교연산을 하기 위한 것이다.

$$\text{P23} = 7/3 - 4/3 - 1 == 0 \tag{1.3.12}$$

변수 P23에는 1이 아닌 0이 할당된다. 즉, MATLAB은 이 등식이 성립하지 않는다는 결론을 출력한다.

두 번째 경우에서 P31은 다음 식을 계산하는 것이다.

$$\text{P31} = (a+b)*c-(a*c+b*c) \tag{1.3.13}$$

MATLAB이 제시하는 답은 0이 아닌 $9.0949 \cdot 10^{-13}$이다. 즉, 분배법칙이 성립하지 않는다. 변수 P32는 다음 비교연산을 하기 위한 것이다.

$$\text{P32} = (a + b) * c == (a * c + b * c) \tag{1.3.14}$$

MATLAB이 제시하는 답은 0이다. 즉, 분배법칙이 성립하지 않음을 확인할 수 있다.

세 번째 경우는 다음 2차방정식의 근들을 구하는 것이다.

$$ax^2 + bx + c = 0 \tag{1.3.15}$$

여기서 $a = 1$, $b = 2 \cdot 10^8$, 그리고 $c = 1$이다. 중학교 때 배운 근의 공식을 사용하면, 이 2차방정식의 근들이 다음과 같음을 알 수 있다.

$$x_1 = -10^8 - \sqrt{10^{16} - 1}, \quad x_2 = -10^8 + \sqrt{10^{16} - 1} \tag{1.3.16}$$

MATLAB이 제시하는 x_1은 $-200,000,000 = -2 \cdot 10^8$이다. 변수 A1은 다음 식이 성립하는지 여부를 조사하기 위한 것이다.

$$ax_1^2 + bx_1 + c = 0 \tag{1.3.17}$$

변수 A1에는 0이 출력된다. 즉, 식 (1.3.17)이 성립하지 않는다. 변수 A2는 식 (1.3.17)의 좌변을 계산한 값으로 1이다. 그러나, 근 x_1이 $-2 \cdot 10^8$인 점을 고려하면, 이 정도의 오차는 작은 것이라고 느껴진다. MATLAB이 제시하는 x_2는 0이다. 변수 B1은 다음 식이 성립하는지 여부를 조사하기 위한 것이다.

$$ax_2^2 + bx_2 + c = 0 \tag{1.3.18}$$

변수 B1에는 0이 출력된다. 즉, 식 (1.3.18)이 성립하지 않는다. 변수 B2는 식 (1.3.18)의 좌변을 계산한 값으로 1이다. 그러나, 근 x_2가 0인 점을 고려하면, 이 정도의 오차를 받아들일 수 없다. 그렇다면, 우리는 좀 더 정교한 x_2를 구할 필요가 있다. 근과 계수의 관계, 즉 Vieta정리에서 알 수 있듯이, x_2는 $1/x_1$이다. 이렇게 계산된 새로운 근은 $x_3 = -5.000000000000000 \cdot 10^{-9}$이다. 변수 C1은 다음 식이 성립하는지 여부를 조사하기 위한 것이다.

$$ax_3^2 + bx_3 + c = 0 \tag{1.3.19}$$

변수 C1에는 1이 출력된다. 즉, 식 (1.3.19)이 성립한다. 변수 C2는 식 (1.3.19)의 좌변을 계산한 값으로 0이다. 다음 Taylor전개가 성립한다.

$$\sqrt{1-x} = 1 - \frac{1}{2}x - \frac{1}{8}x^2 - \cdots \tag{1.3.20}$$

식 (1.3.20)를 이용해서, 다음과 같이 x_2의 정교한 근사값인 x_4를 구할 수 있다.

$$x_4 \doteq \frac{b}{2a}\left\{-\frac{1}{2}\left[\frac{4ac}{b^2}\right]-\frac{1}{8}\left[\frac{4ac}{b^2}\right]^2\right\} = -5.000000000000000\cdot 10^{-9} \tag{1.3.21}$$

변수 D1은 다음 식이 성립하는지 여부를 조사하기 위한 것이다.

$$ax_4^2+bx_4+c=0 \tag{1.3.22}$$

변수 D1에는 1이 출력된다. 즉, 식 (1.3.22)이 성립한다. 변수 D2는 식 (1.3.22)의 좌변을 계산한 값으로 0이다.

네 번째 경우에서 변수 z1은 다음 식을 계산하는 것이다.

$$z_1 = \ln\left(a-\sqrt{a^2-1}\right) \tag{1.3.23}$$

여기서 $a=10^8$이다. MATLAB이 제시하는 z_1은 $-\infty$이다. 다음 식이 성립한다.

$$\ln\left(a-\sqrt{a^2-1}\right) = -\ln\left(a+\sqrt{a^2-1}\right) \tag{1.3.24}$$

변수 z2는 식 (1.3.24)의 우변을 계산한 값으로서, MATLAB이 제시하는 값은 -19.1138이다. 변수 z3는 Taylor급수 (1.3.20)를 사용해서 계산한 값으로서, MATLAB이 제시하는 값은 -19.1138이다.

다섯 번째 경우에서 변수 tann은 $\{\tan(\pi k/2)|k=0,1,\cdots,22\}$를 계산하는 것이다. 만약 k가 홀수이면, $\tan(\pi k/2)$는 오버플로우(overflow)이어야 한다. 그러나, MATLAB이 제시하는 답은 오버플로우가 아니라 $1.6331\cdot 10^{16}$로부터 점점 작아지는 값을 출력한다. 변수 sinn은 $\{\sin(\pi k)|k=0,1,\cdots,22\}$를 계산하는 것이다. 정의에 의해서 $\sin(\pi k)=0$이다. 그러나, MATLAB이 제시하는 답은 0이 아니고 $1.2246\cdot 10^{-16}$에서 절대값이 점점 커지는 값을 출력한다. 이 그래프들이 그림 1.3.1에 그려져 있다.

예제 1.3.1 Python을 사용해서 MATLAB프로그램 ComputerError101.m을 다시 다루기 위해서, 다음 Python프로그램 ComputerError101.Py를 실행해 보자.

```
"""
Created on Thu Jan 12 08:59:40 2017
@author: CBS
"""
import numpy as np
```

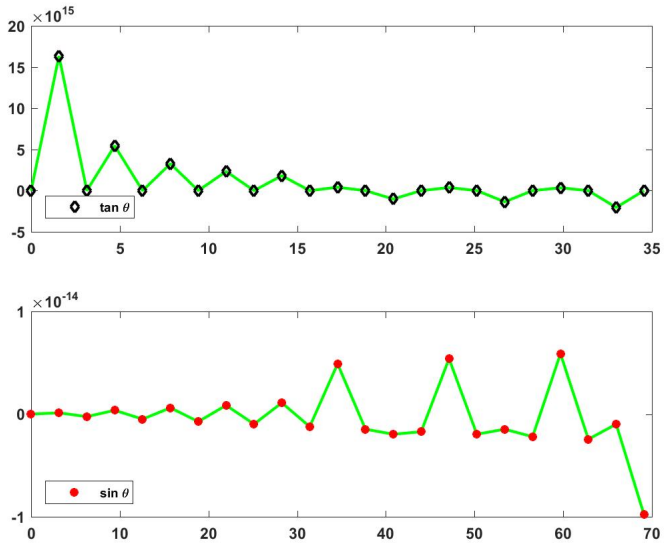


그림 1.3.1. 탄젠트값과 사인값

```
import matplotlib.pyplot as plt

# Case 1
P11 = (0.1 + 0.1 + 0.1 + 0.1) - 0.4
P12 = (0.1 + 0.1 + 0.1 + 0.1) == 0.4
P13 = (0.1 + 0.1 + 0.1) == 0.3
P14 = (0.1 + 0.1 + 0.1) - 0.3
P15 = 0.1 + 0.1 + (0.1 - 0.3)
P16 = 0.1 + (0.1 + 0.1) - 0.3
P17 = (0.1 + 0.1) + 0.1 - 0.3
P18 = 3*0.1 - 0.3
P21 = 0.1*0.1 - 0.01

# Case 2
a = 12345.678; b = a*10**(-8); c = 1/3
P31 = (a+b)*c - (a*c+b*c)
P32 = (a+b)*c == (a*c+b*c)

# Case 3
a = 1; b = 2*10**8; c = 1;
x1 = (-b-np.sqrt(b**2-4*a*c))/2/a; x1f = float(x1)
A1 = (a*x1f**2+b*x1f+c == 0)
A2 = a*x1f**2+b*x1f+c;             A2f = float(A2)
x2 = (-b+np.sqrt(b**2-4*a*c))/2/a; x2f = float(x2)
B1 = (a*x2f**2+b*x2f+c == 0)
B2 = a*x2f**2+b*x2f+c;             B2f = float(B2)
x3 = 1/x1;                         x3f = float(x3)
C1 = (a*x3f**2+b*x3f+c == 0)
C2 = a*x3f**2+b*x3f+c;             C2f = float(C2)
x4 = b*( -1/2*(4*a*c/b**2)-1/8*(4*a*c/b**2)**2 )/2/a
D1 = (a*x4**2+b*x4+c == 0)
D2 = a*x4**2+b*x4+c

# Case 4
a = 10**8
z1 = np.log(a-np.sqrt(a**2-1))
z2 = -np.log(a+np.sqrt(a**2-1))
```

```
z3 = np.log(1/2/a+1/8/a**3)

# Case 5: Plotting
fig, axes = plt.subplots(nrows=1, ncols=2)
ax1 = plt.subplot(211)
theta1 = np.pi/2*np.linspace(0, 22, num=23)
tann = np.tan(theta1)
plt.plot(theta1, tann,'k-d', label=r'$\tan \theta$')
plt.legend(loc=1)
plt.xlabel('\theta')
ax2 = plt.subplot(212)
theta2 = np.pi*np.linspace(0, 22, num=23)
sinn = np.sin(theta2)
plt.plot(theta2, sinn,'r-*', label=r'$\sin \theta$')
plt.legend(loc=3)
plt.show()
fig.savefig('ComputorError101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 MATLAB프로그램 ComputerError101.m의 결과와 같다. 여기서 유의할 점은 Python프로그램에서는 컴퓨터의 유효숫자문제를 해결하기 위해서 식으로 답을 표기하는 경우가 있다. 이렇게 표현된 식을 숫자로 표기하기 위해서 float함수를 사용하였다. ■

지금까지 설명한 내용에서 알 수 있듯이, 무심하게 컴퓨터를 사용하면 이상한 답을 얻을 수도 있다. 적어도 본서를 쓰는 현재 시점에서 컴퓨터는 스스로 새로운 수학문제를 풀지는 못한다. 단지 프로그래머가 시키는 수치계산 또는 논리계산을 할 따름이다. 따라서, 컴퓨터를 사용해서 수리적 문제를 풀기 위해서는 수치해석적 능력 뿐 아니라 컴퓨터를 잘 이해하고 다룰 수 있어야 한다. 이러한 과학적 컴퓨팅 실력을 배양시키는 것이 본서의 목표이다.

제1.4절 부동소수점

십진수 123.456를 다음과 같이 쓸 수 있다.

$$1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2} + 6 \cdot 10^{-3} \tag{1.4.1}$$

이 수를 다음과 같은 형태로 표현할 수 있다.

$$10^2 \cdot \left[1 \cdot 10^0 + 2 \cdot 10^{-1} + 3 \cdot 10^{-2} + 4 \cdot 10^{-3} + 5 \cdot 10^{-4} + 6 \cdot 10^{-5}\right] \tag{1.4.2}$$

식 (1.4.2)와 같은 표현법을 부동소수점표현(floating point format) 또는 떠돌이소수점형이라 한다. 부동소수점표현은 실수를 표현할 때 소수점의 위치를 고정하지 않고 소수점의 위치를 나타내는 지수(exponent, 指數)를 따로 적는 것이다. 이때 유효숫자를 가수(mantissa, 假數)라 한다. 십진수는 밑수(base)를 10으로 하는 수이고, 컴퓨터에서는 밑수를 2로 하는 이진수를 사용한다.

컴퓨터메모리의 기본단위는 비트(bit)로서 0 또는 1을 나타낸다. 컴퓨터에 따라 다르지만 일반적으로 이러한 비트들이 8개 모인 것을 바이트(byte)라 부른다. 이러한 바이트가 2개 또는 4개가 모인 것을 워드(word)라고 부른다. 예전에는 바이트가 2개 모인, 즉 16비트를 워드라고 했으나 Intel386 프로세서 이후에는 32비트를 1 워드라고 하는 것이 마이크로프로세서를 설명하는데 더 타당한 것 같다. 그리고, 컴퓨터 하드웨어의 발전에 따라 워드의 크기는 점점 더 커져갈 것이다. 그러나, 본서를 읽는 수준에서는 워드라는 단어를 구태여 이해하려고 노력할 필요는 없다고 생각한다. 워드에 대해 관심이 있는 독자는 컴퓨터 전문서적을 참조하기 바란다.

다음 예제에서는 십진수를 이진수로 바꾸는 방법을 예를 들어 설명한다.

예제 1.4.1 십진수 $[11]_{10}$을 이진수 $[d_5d_4d_3d_2d_1d_0]_2$로 바꾸기 위해서 다음 식을 살펴보자.

$$11 = d_4 \cdot 2^4 + d_3 \cdot 2^3 + d_2 \cdot 2^2 + d_1 \cdot 2^1 + d_0 \cdot 2^0 \tag{1}$$

여기서 d_i는 0 또는 1이다. 식 (1)을 다음과 같이 쓸 수 있다.

$$11 = 2\left[2\left\{2\left(2d_4 + d_3\right) + d_2\right\} + d_1\right] + d_0 \cdot 2^0 \tag{2}$$

식 (2)에서 알 수 있듯이, 11을 2로 나눈 나머지가 d_0이고 몫을 다시 2로 나눈 나머지가 d_1이다. 이러한 과정을 반복하면, 다음 식을 얻는다.

$$11 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \tag{3}$$

즉, 십진수 $[11]_{10}$은 이진수로 $[1011]_2$이다.

십진수 $[0.1]_{10}$을 이진수로 바꾸기 위해서 다음 식을 살펴보자

$$0.1 = \frac{d_{-1}}{2^1} + \frac{d_{-2}}{2^2} + \frac{d_{-3}}{2^3} + \cdots \tag{4}$$

여기서 d_{-i}는 0 또는 1이다. Horner형식을 사용해서 식 (4)을 다음과 같이 쓸 수 있다.

$$0.1 = \frac{1}{2}\left(d_{-1} + \frac{1}{2}\left\{d_{-2} + \frac{1}{2}\left[d_{-3} + \frac{1}{2}\left(d_{-4} + \frac{1}{2}\{\cdots\}\right)\right]\right\}\right) \tag{5}$$

식 (5)에서 알 수 있듯이, 0.1에 2를 곱해서 얻은 정수부분이 d_{-1}이고 소수부분을 다시 2로 곱해서 얻은 정수부분이 d_{-2}이다. 이러한 과정을 반복하면, 다음 식을 얻는다.

$$[0.1]_{10} = [0.0001100110011001100110011001\cdots]_2 \tag{6}$$

다음 식들로부터 식 (6)이 성립함을 확인할 수 있다.

$$[0.1]_{10} = \sum_{k=1}^{\infty}\left[\frac{1}{2^4} + \frac{1}{2^5}\right]\left[\frac{1}{2^4}\right]^k = \frac{3}{32}\cdot\frac{1}{1-\frac{1}{16}} = \frac{1}{10} \tag{7}$$

다음 MATLAB프로그램 Decimal2Binary101.m은 십진수를 이진수로 바꾸기 위한 것이다.

```
% ------------------------------------------------------------------
%  Filename: Decimal2Binary101.m
%  Convert a decimal to a binary of a positive number
%  Programmed by CBS
%-------------------------------------------------------------------
clear all, close all
d(1) = 11     % decimal numbr to convert; First example
% d(1) = 0.1  % decimal numbr to convert; Second example
if d(1) > 1
   for ii=1:100
       d(ii+1) = floor(d(ii)/2);
       if d(ii) == 0,
           break
       end
       b(ii) = d(ii)-2*d(ii+1);
   end
   b          % binomial number
else
    for ii=1:100
       b(ii) = floor(2*d(ii));
       d(ii+1) = 2*d(ii)-b(ii);
       if d(ii) == 0,
           break
       end
    end
    b         % binomial number
end
% End of Program
%-------------------------------------------------------------------
```

■

예제 1.4.2 Python을 사용해서 예제 1.4.1을 다시 다루기 위해서, 다음 Python프로그램 Decimal2Binary101.Py를 실행해 보자.

```
"""
Created on Thu Jan 12 08:59:40 2017
@author: CBS
"""
# Use Python function bin to convert decimal into binary
def dec_to_bin(x):
    return int(bin(x)[2:])

b11 = dec_to_bin(11)    # Decimal 11 = Binary bin(11)
# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.4.1의 결과와 같다. ■

컴퓨터시대의 초기는 혼돈의 시대였다. 동일한 프로그램을 다양한 컴퓨터기종들에서 실행하면 기종마다 다른 결과가 나오거나 프로그램이 돌아가지도 않는 경우가 많았다. 즉, 소프트웨어와 하드웨어를 분리해서 생각하기가 어려웠다. 이러한 컴퓨터세계의 혼란을 극복하기 위해서, 1990년 IEEE에서 표준(standard)을 제시했고, 2008년 이 표준에 약간의 수정을 가했다. 오늘날에는 특별한 용도의 컴퓨터나 탁상용계산기를 제외한 일반적인 하드웨어업체나 소프트웨어업체들은 이 IEEE표준을 따르고 있다. 이 IEEE 부동소수점표준(IEEE Standard for Floating-Point Arithmetic)을 IEEE 754라고 부르는데, IEEE 754에서 실수 x의 부동소수점표현 $\mathrm{fl}(x)$는 다음과 같다.

$$\mathrm{fl}(x) \doteq \pm \left[1 + \frac{d_{-1}}{2^1} + \frac{d_{-2}}{2^2} + \cdots + \frac{d_{-t}}{2^t}\right] \times 2^e \tag{1.4.3}$$

여기서 d_{-i}는 0 또는 1이다. 또한, 밑수는 2이고, 지수는 e, 그리고 가수는 $[1d_{-1}d_{-2}\cdots d_{-t}]_2$이다. 컴퓨터에서 사용할 수 있는 유효숫자는 유한하다. 즉, t는 유한값이다. 식 (1.4.3)의 부동소수점표현으로 0을 나타낼 수는 없다. 따라서, 0, $+\infty$, 그리고 $-\infty$는 비트들의 특별한 결합으로 나타낸다.

예제 1.4.3 식 (1.4.3)의 부동소수점표현에 의한 점들은 직선 상에서 균등하게 나타나지 않는다. 이 점들을 시각적으로 파악하기 위해서 다음 MATLAB프로그램 FloatingPointDispersion101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename FloatingPointdispersion101.m
% Floating Point dispersion
% Programmed by CBS
```

```
% ------------------------------------------------------------------------
clear all, close all
beta = 2, t = 3, L = -2, U = 3
point = [];
tDum = 2^(1-t);
tMax= 2*(1-2^(-t));
for ii = 1:tDum:tMax
    for jj = L:U
        point = [ point ii*beta^jj];
    end
end
point = [ -point 0 point ];
point = sort(point);
ptNum = length(point);
ptDum = ones(ptNum,1);
set(gca,'fontsize',11,'fontweigh','bold','xtick',[],'ytick',[])
hold on
for jj=1:ptNum
    plot(point(jj)*ptDum,point,'k.','LineWidth',2)
end
axis square
box on
hold off
saveas(gcf,'FloatingPointDispersion101.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 그림 1.4.1이 그려진다. 시각적 효과를 돕기 위해서, 이 그림에서는 1D 그래프가 아닌 2D 그래프를 출력했음에 유의하라.

그림 1.4.1에서 알 수 있듯이, 이웃 점들 사이의 거리는 절대적인 관점에서는 지수적으로 벌어지나 상대적인 관점에서는 상수를 유지한다. 특히 유의할 점은 0의 아주 가까운 이웃에는 도리어 점들이 존재하지 않는다는 것이다. 즉, 식 (1.4.3)은 절대값이 아주 작은 점들을 잘 나타내지 못한다. 부동소수점표현의 정의를 고려하면, 이러한 현상은 당연한 것이다. 이러한 문제점을 해결하기 위해서 0 주변에 식 (1.4.3)으로 나타낼 수 없는 점들, 전문용어로는 부정규수(subnormal numbers)를 추가하는 것이 일반적이다. ■

예제 1.4.4 Python을 사용해서 예제 1.4.3을 다시 다루기 위해서, 다음 Python프로그램 FloatingPointDispersion101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

beta = 2; t = 3; L = -2; U = 3
point1 = [];
tDum = 2**(1-t);
tMax= 2*(1-2**(-t));
```

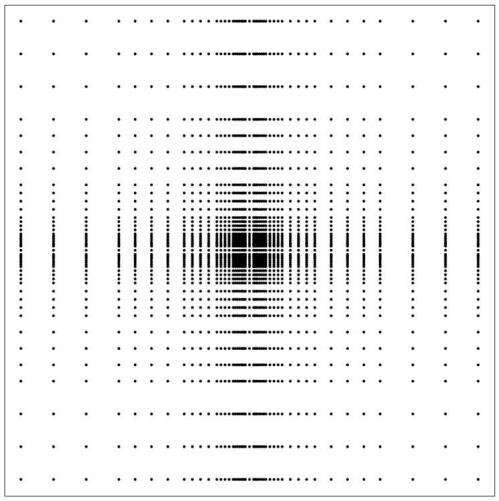


그림 1.4.1. 부동소수점들의 분포

```
nDum = int(tMax/tDum)
for ii in range(1,nDum+1):
    for jj in range(L, U+1):
        point1.append(float(ii*tDum)*beta**jj)
point2 = np.array(point1)
point = np.append(-point2, [0])
point = np.append(point, point2)
point = np.sort(point)
ptNum = len(point)
ptDum = np.ones((ptNum, 1))

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
for jj in range(1,ptNum):
    plt.plot(point[jj-1]*ptDum,point,'k.')
plt.axis([-15, 15, -15, 15])
ax.set_aspect('equal')
frame = plt.gca()
frame.axes.get_xaxis().set_ticks([])
frame.axes.get_yaxis().set_ticks([])
plt.show()
fig.savefig('FloatingPointDispersion101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.4.3의 결과와 같다. ■

MATLAB에서는 특별히 다른 명령이 없는 한 배정밀도(double precision 또는 long precision)로 계산을 수행한다. 이는 64비트인 워드(64-bit word)를 사용한다는 뜻이다. 즉, 그림 1.4.2에 묘사된 부동소수점표현을 사용한다. 이 배정밀도를 사용하는 부동소수점표현에

서는 첫 번째 비트에 부호(sign)를 표시한다. 즉, 실수 x의 부동소수점표현 $\mathrm{fl}(x)$의 첫 번째 비트가 0이면 $\mathrm{fl}(x)$는 양수이고, 첫 번째 비트가 1이면 $\mathrm{fl}(x)$는 음수이다. 그 다음 11비트는 부호를 포함한 지수(signed exponent)를 표시한다. 따라서, 나타낼 수 있는 지수는 모두 $2^{11} = 2048$이다. 이를 사용해서 -1022부터 1023까지 지수를 표시한다. 나머지 52비트는 가수(mantissa)를 표시한다. 식 (1.4.3)의 첫 번째 자리는 1이므로 가수는 $53(= 52 + 1)$개 비트로 나타내진다. 따라서, 이 부동소수점표현으로 나타낼 수 있는 숫자의 최대값은 다음과 같다.

$$2^{1023} \cdot \left[1 + \sum_{i=1}^{52} \frac{1}{2^i}\right] = 2^{1023}\left[2 - 2^{-52}\right] \approx 2^{1024} \approx 1.797693134862328 \cdot 10^{308} \tag{1.4.4}$$

MATLAB함수 realmax를 사용하면, MATLAB에서 사용할 수 있는 최대값이 $1.7977 \cdot 10^{308}$임을 알 수 있다. 이보다 큰 수가 나타나면 오버플로우(overflow) 되었다고 하고 MATLAB은 inf를 할당한다. 이 부동소수점표현으로 나타낼 수 있는 양수의 최소값은 다음과 같다.

$$2^{-1022} \cdot [1 + 0] = 2^{-1022} \approx 2.225073858507187 \cdot 10^{-308} \tag{1.4.5}$$

MATLAB함수 realmin를 사용하면, MATLAB에서 사용할 수 있는 최소 양수가 $2.2251 \cdot 10^{-308}$임을 알 수 있다. 이보다 작은 양수가 나타나면 언더플로우(underflow) 되었다고 하고 MATLAB은 0을 할당한다.

그림 1.4.2. 부동소수점

MATLAB에서 단정밀도(single precision 또는 short precision)를 사용한다는 것은 32비트인 워드(32-bit word)를 사용한다는 뜻이다. 단정밀도를 사용하는 부동소수점표현에서도 첫 번째 비트에 부호를 표시한다. 즉, 실수 x의 부동소수점표현 $\mathrm{fl}(x)$의 첫 번째 비트가 0이면 $\mathrm{fl}(x)$는 양수이고, 첫 번째 비트가 1이면 $\mathrm{fl}(x)$는 음수이다. 그 다음 8비트는 부호를 포함한 지수를 표시한다. 따라서, 나타낼 수 있는 지수는 모두 $2^8 = 256$이다. 이를 사용해서 -126부터 127까지 지수를 표시한다. 나머지 23비트는 가수(mantissa)를 표시한다. 식 (1.4.3)의 첫 번째 자리는 1이므로 가수는 $24(= 23 + 1)$개 비트로 나타내진다. 따라서, 이 부동소수점

표현으로 나타낼 수 있는 숫자의 최대값은 다음과 같다.

$$2^{127} \cdot \left[1 + \sum_{i=1}^{23} \frac{1}{2^i}\right] = 2^{1023}\left[2 - 2^{-23}\right] \approx 2^{128} \approx 3.4028237 \cdot 10^{38} \tag{1.4.6}$$

MATLAB함수 realmax('single')을 사용해서 MATLAB에서 사용할 수 있는 최대값이 $3.4028 \cdot 10^{38}$임을 알 수 있다. 이보다 큰 수가 나타나면 오버플로우 되었다고 하고 MATLAB은 inf를 할당한다. 이 부동소수점표현으로 나타낼 수 있는 양수의 최소값은 다음과 같다.

$$2^{-126} \cdot [1 + 0] = 2^{-126} \approx 1.1754944 \cdot 10^{-38} \tag{1.4.7}$$

MATLAB함수 realmin('single')을 사용해서 MATLAB에서 사용할 수 있는 최소 양수가 $1.1755 \cdot 10^{-38}$임을 알 수 있다. 이보다 작은 양수가 나타나면 언더플로우 되었다고 하고 MATLAB은 0을 할당한다. MATLAB에서 단정밀도를 사용하는 보편적 이유는 메모리공간 때문이다. 그러나, 오늘날 컴퓨터의 메모리크기는 크게 문제가 되지 않으므로 실제로 단정밀도를 사용해야하는 경우는 별로 없을 것 같다. MATLAB에서 단정밀도로 수를 표현하기 위해서는 MATLAB함수 single을 사용한다.

예제 1.4.5 배정밀도와 단정밀도의 차이는 반올림오차(roundoff error)로 나타나는데, 이 차이는 매우 산만하게(erratic) 나타난다. 이러한 성질을 살펴보기 위해서 다음 MATLAB 프로그램 DoubleSingle101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename DoubleSingle101.m
% Difference between Double Precision and Single Precision
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
xx = 0/3:0.001:pi/2;
fDouble = sin(xx);
fSingle = single(fDouble);
fDifference = fDouble-fSingle;
% Plotting
plot(xx,fDifference,'k-','LineWidth',1)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0 pi/2])
legend('\bf Difference between Double and Single Precisions', ...
       'location','SW')
saveas(gcf,'DoubleSingle101.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 그림 1.4.3이 그려진다. 사인함수는 구간 $[\pi/3, \pi/2]$에서 매우 평활하다. 그러나, 그림 1.4.3에서 알 수 있듯이, 이 구간에서 사인값을 배정밀도로

표현한 값과 단정밀도로 표현한 값의 차이는 매우 변화가 심하다. 즉, 함수값이 커지면 배정밀도와 단정밀도의 차이도 커진다. 뒤에서 알게 되겠지만, 상대적 차이를 그리면 이러한 현상은 사라진다. ■

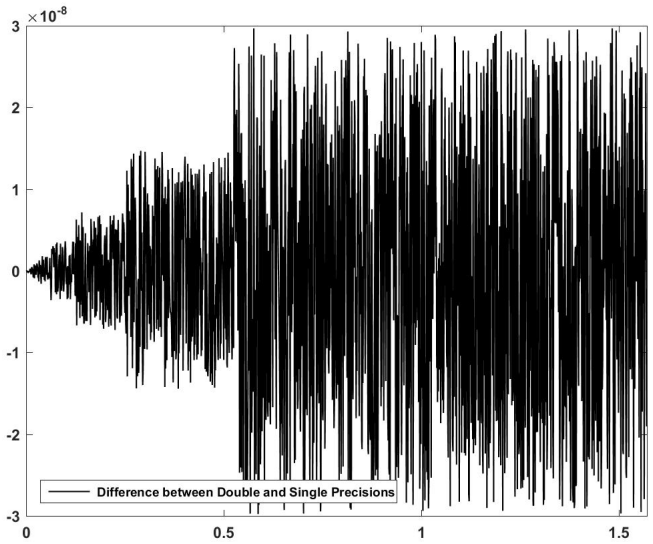


그림 1.4.3. 배정밀도와 단정밀도

예제 1.4.6 Python을 사용해서 예제 1.4.5을 다시 다루기 위해서, 다음 Python프로그램 DoubleSingle101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

xx = np.linspace(0/3, np.pi/2, num=1000)
fDouble = np.sin(xx)
fSingle = np.float32(fDouble)
fDifference = fDouble-fSingle

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(xx,fDifference,'k-',
         label='Difference between Double and Single Precisions')
plt.legend(loc='lower left', numpoints = 1 )
plt.show()
fig.savefig('DoubleSingle101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.4.5의 결과와 같다. ■

앞 절에서 언급한 컴퓨터를 사용할 때 얻어지는 이상한 결과들은 부동소수점을 사용함으로써 발생하는 것이다. 이러한 문제점들 중에서 가장 심각한 것은 크기가 비슷한 두 값들의 차이를 구할 때 발생한다. 제1.3절에서 다룬 세 번째 경우와 네 번째 경우가 대표적인 예이다. 앞에서 언급했듯이, 부동소수점 계산에서는 교환법칙은 성립하지만 결합법칙이나 분배법칙이 항상 성립하는 것은 아니다. 따라서, 컴퓨터를 사용해서 어떤 문제를 풀고자 할 때는, 컴퓨터가 수를 표현하는 법칙, 즉 부동소수점표현을 고려해서 신중하게 프로그램을 해야 할 것이다.

제1.5절 오차

디지털컴퓨터를 사용해서 계산을 한 결과에는 반드시 오차(error)가 따르기 마련이다. 본저자는 어떤 문제의 답이 수치적으로 제시되는 경우 원래 정확한 답이 유리수가 아닌 경우에 우리가 답이라고 주장하는 값에는 반드시 오차가 존재한다고 생각한다. 확대해서 해석하면 우리의 삶 자체가 때로는 오차를 무시하면서 때로는 달래가면서 사는 것이라고 생각한다. 이러한 말을 하는 이유는 오차를 단순히 무시하지도 말고 또 너무 두려워하지도 말아야 한다는 것이다. 과학적 컴퓨팅에서는 오차를 우리가 무시해도 될 정도로 작게 만드는 것이 가능한 범위 내에서 문제를 정확하게 푸는 핵심이다. 물론 이 과정에서 컴퓨팅의 효율성을 고려해야 한다.

1.5.1 절대오차와 상대오차

측정오차(measured error)에는 절대오차(absolute error)와 상대오차가 있다. 상수 c를 근사값 a로 나타내면, 절대오차는 $|c-a|$이다. 만약 $c \neq 0$이면, 상대오차는 $|c-a|/|c|$이다. 일반적으로 절대오차보다는 상대오차가 더 의미가 있는 측도(meaningful measure)이다. 특히, 부동소수점표현으로 수를 나타낼 때는 절대오차보다는 상대오차가 더 의미가 있다. 또한, 절대값 $|c|$가 큰 경우에는 절대오차보다 상대오차보다 더 유용하다. 그러나, 절대값 $|c|$가 0에 가까운 경우에는 상대오차에 큰 의미를 두지 않는 것이 좋다. 그 이유는 $\frac{0}{0}$은 부정(indefinite)이고 아주 작은 양수는, 예를 들어 MATLAB에서는 $2.2251 \cdot 10^{-308}$보다 작은 양수는 0으로 간주되기 때문이다. 만약 $c \approx a$이면, 다음 식들이 성립한다.

$$\left| \ln \frac{a}{c} \right| = \left| \ln \left(1 - \frac{c-a}{c} \right) \right| \approx \left| \frac{c-a}{c} \right| \tag{1.5.1}$$

즉, 상대오차는 로그값들의 차이인 $|\ln c - \ln a|$와 비슷하다. 절대값 $|c|$가 0에 가깝다면 $|\ln c|$와 $|\ln a|$는 아주 큰 값들이 될 것이고 그 차이 $|\ln c - \ln a|$도 큰 값이 될 수 있다. 따라서, 절대값 $|c|$가 0에 가까운 경우에는 상대오차를 사용하는 것이 별 도움이 되지 않을 수도 있다. 다음 예제들은 절대오차와 상대오차를 비교하기 위한 것들이다.

예제 1.5.1 Taylor전개를 사용해서 계산하면, 식 $\sin(0.499\pi) = 1.0000$과 식 $\cos(0.499\pi) = 0.0031$이 성립한다. 이들의 절대오차와 상대오차를 살펴보기 위해서, 다음 MATLAB프로그램 ComputerError102.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename ComputerError102.m
% Absoute Error and Relative Error
% Taylor Exapnsions of Sin and Cos near zero
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
x = 0.499*pi
% Initialization
Nmax = 20; NN = 1:Nmax;
SinTrue = sin(x)
CosTrue = cos(x)
SinSol(1) = x;
CosSol(1) = 1;
% Iteration
for ii = 2:Nmax
    SinSol(ii) = SinSol(ii-1) + (-1)^(ii-1)/factorial(2*ii-1)*x^(2*ii-1);
    CosSol(ii) = CosSol(ii-1) + (-1)^(ii-1)/factorial(2*ii-2)*x^(2*ii-2);
end
SinAbsErr = abs(SinSol - SinTrue);
SinRelErr =  abs(SinSol - SinTrue)./abs(SinTrue);
CosAbsErr = abs(CosSol - CosTrue);
CosRelErr =  abs(CosSol - CosTrue)./abs(CosTrue);
format short g
OutP = [ NN ; SinAbsErr; SinRelErr; CosAbsErr; CosRelErr ]'
% Plotting
subplot(2,1,1)
plot(NN,SinAbsErr,'r-',NN,CosAbsErr,'k--','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold','ylim',[-0.1 3])
legend('\bf Abs Error of Sin','\bf Abs Error of Cos')
axis([ 0 20 0 1 ])
subplot(2,1,2)
plot(NN,SinRelErr,'r-',NN,CosRelErr,'k--','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold','ylim',[-0.1 3])
legend('\bf Rel Error of Sin','\bf Rel Error of Cos')
axis([ 0 20 0 1 ])
saveas(gcf,'ComputerError102.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 다음 값들이 출력된다.

```
-------------------------------------------------------

   n  Sin Abs Err     Rel Err     Cos Abs Err     Rel Err
```

```
-------------------------------------------------------
 1      0.56766      0.56766      0.99686       317.31
 2     0.074436     0.074437      0.23191        73.82
 3    0.0044625    0.0044625     0.019734       6.2815
 4  0.00015411   0.00015411   0.00088041      0.28024
 5  3.4656e-06   3.4657e-06   2.4249e-05    0.0077186
 6  5.4816e-08   5.4816e-08   4.5376e-07   0.00014444
 7   6.432e-10    6.432e-10    6.147e-09   1.9567e-06
 8   5.822e-12    5.822e-12   6.3082e-11    2.008e-08
 9  4.1966e-14   4.1967e-14   5.0744e-13   1.6152e-10
10  2.2204e-16   2.2205e-16   3.2608e-15    1.038e-12
11           0            0   4.2067e-17    1.339e-14
12           0            0   2.4286e-17   7.7305e-15
13           0            0   2.4286e-17   7.7305e-15
14           0            0   2.4286e-17   7.7305e-15
15           0            0   2.4286e-17   7.7305e-15
16           0            0   2.4286e-17   7.7305e-15
17           0            0   2.4286e-17   7.7305e-15
18           0            0   2.4286e-17   7.7305e-15
19           0            0   2.4286e-17   7.7305e-15
20           0            0   2.4286e-17   7.7305e-15
-------------------------------------------------------
```

또한, 그림 1.5.1이 출력된다. 이 값들과 그림 1.5.1에서 알 수 있듯이, 사인값의 절대오차나 코사인값의 절대오차는 제5번째 반복(iteration)에서 10^{-4}보다 작아진다. 그러나, 사인값은 1에 가까우나 코사인값은 0에 가까우므로 사인값의 상대오차가 코사인값의 상대오차보다 빨리 0에 수렴한다. ■

예제 1.5.2 Python을 사용해서 예제 1.5.1을 다시 다루기 위해서, 다음 Python프로그램 ComputerError102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
import math

x = 0.499*np.pi
```

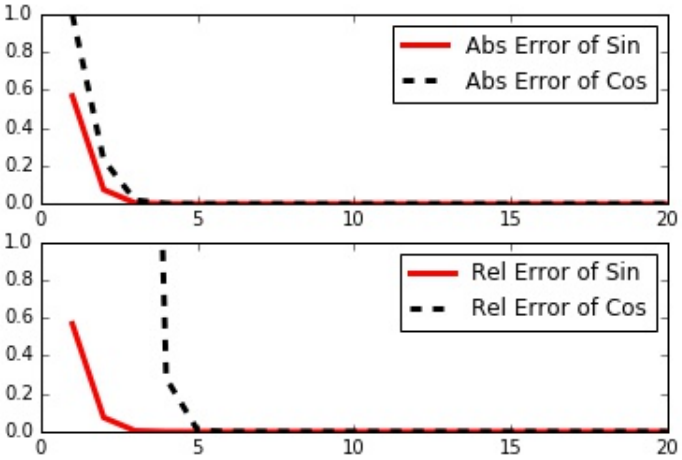


그림 1.5.1. 절대오차와 상대오차 1

```
Nmax = 20;
NN = np.linspace(1, Nmax, num=20); NN=np.reshape(NN, (Nmax,1))
SinSol = np.zeros((Nmax,1),float)
CosSol = np.zeros((Nmax,1),float)

# Initialization
SinTrue = np.sin(x)
CosTrue = np.cos(x)
SinSol[0] = x
CosSol[0] = 1
# Iteration
for ii in range(1,Nmax):
    SinSol[ii] = SinSol[ii-1] + (-1)**(ii)/math.factorial(2*ii+1)*x**(2*ii
    +1)
    CosSol[ii] = CosSol[ii-1] + (-1)**(ii)/math.factorial(2*ii)*x**(2*ii)
SinAbsErr = np.abs(SinSol - SinTrue)
SinRelErr =  np.abs(SinSol - SinTrue)/np.abs(SinTrue)
CosAbsErr = np.abs(CosSol - CosTrue)
CosRelErr =  np.abs(CosSol - CosTrue)/np.abs(CosTrue)
OutP = np.concatenate((NN, SinAbsErr, SinRelErr,CosAbsErr, CosRelErr), axis
    =1)

# Plotting
fig = plt.figure()
ax1 = fig.add_subplot(211)
plt.plot(NN,SinAbsErr,'r-', lw=3, label='Abs Error of Sin')
plt.plot(NN,CosAbsErr,'k--', lw=3, label='Abs Error of Cos')
plt.axis([ 0, 20, 0, 1 ])
plt.legend(loc='upper right', numpoints = 1 )
ax2 = fig.add_subplot(212)
plt.plot(NN,SinRelErr,'r-', lw=3, label='Rel Error of Sin')
plt.plot(NN,CosRelErr,'k--', lw=3, label='Rel Error of Cos')
plt.axis([ 0, 20, 0, 1 ])
plt.legend(loc='upper right', numpoints = 1 )
fig.savefig('ComputerError102.jpg')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.5.1의 결과와 같다. ■

예제 1.5.3 예제 1.2.5에서 언급했듯이, 다음 Stirling근사식은 $n!$의 근사를 계산하는데 유용하다.

$$n! \approx \sqrt{2\pi n} n^n e^{-n}, \quad (n = 1, 2, \cdots) \tag{1}$$

좀 더 정교한 근사는 다음과 같다.

$$n! \approx \sqrt{2\pi n} n^n e^{-n} \left[1 + \frac{1}{12n}\right], \quad (n = 1, 2, \cdots) \tag{2}$$

이들의 절대오차와 상대오차를 살펴보기 위해서, 다음 MATLAB프로그램 Computer-Error103.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename ComputerError103.m
% Absoute Error and Relative Error of Stirling's formula
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
x = 1
% Initialization
Nmax = 20; nn = 1:Nmax;
ee = exp(1)
StirTrue = factorial(nn);
StirSol = sqrt(2*pi*nn).*((nn/ee).^nn).*(1+1/12./nn);
StirAbsErr = abs(StirSol - StirTrue);
StirRelErr =  abs(StirSol - StirTrue)./abs(StirTrue);
format short g
OutP = [ nn ; StirTrue; StirSol; StirAbsErr; StirRelErr ]'
% Plotting
subplot(2,1,1)
plot(nn,StirAbsErr,'r-','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf Absolute Error of Stiring Approximation','location','NW')
subplot(2,1,2)
plot(nn,StirRelErr,'k--','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf Relative Error of Stiring Approximation','location','NW')
saveas(gcf,'ComputerError103.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 다음 값들이 출력된다.

```
-----------------------------------------------------------
   n        n!        Stirling      Abs Err       Rel Err
-----------------------------------------------------------
   1           1       0.99898    0.0010182     0.0010182
```

```
 2            2         1.999     0.0010371    0.00051857
 3            6        5.9983     0.0016735    0.00027891
 4           24        23.996     0.0041129    0.00017137
 5          120        119.99      0.013846    0.00011538
 6          720        719.94      0.059618    8.2803e-05
 7         5040        5039.7       0.31374     6.225e-05
 8        40320         40318        1.9546    4.8477e-05
 9  3.6288e+05   3.6287e+05         14.082    3.8806e-05
10  3.6288e+06   3.6287e+06        115.25     3.176e-05
11  3.9917e+07   3.9916e+07        1056.6     2.647e-05
12   4.79e+08    4.7899e+08         10728    2.2397e-05
13   6.227e+09   6.2269e+09    1.1953e+05    1.9196e-05
14  8.7178e+10   8.7177e+10    1.4502e+06    1.6634e-05
15  1.3077e+12   1.3077e+12    1.9031e+07    1.4553e-05
16  2.0923e+13   2.0923e+13    2.6863e+08    1.2839e-05
17  3.5569e+14   3.5568e+14    4.0586e+09    1.1411e-05
18  6.4024e+15   6.4023e+15    6.5354e+10    1.0208e-05
19  1.2165e+17   1.2164e+17    1.1174e+12    9.1856e-06
20  2.4329e+18   2.4329e+18    2.0216e+13    8.3095e-06
------------------------------------------------------------
```

또한, 그림 1.5.2가 출력된다. 이 값들과 그림 1.5.2에서 알 수 있듯이, Stirling근사식에 의한 절대오차는 n이 증가함에 따라 증가하지만 상대오차는 감소한다. 따라서, 이 경우에는 근사 정도를 나타내는데 절대오차보다 상대오차가 더 유리하다. ■

예제 1.5.4 Python을 사용해서 예제 1.5.3을 다시 다루기 위해서, 다음 Python프로그램 ComputerError103.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
import math

x = 1.0
Nmax = 20;
NN = np.linspace(1, Nmax, num=20); nn = np.reshape(NN, (Nmax,1))
ee = np.e
StirTrue = np.zeros((Nmax,1),float)
StirSol = np.zeros((Nmax,1),float)
```

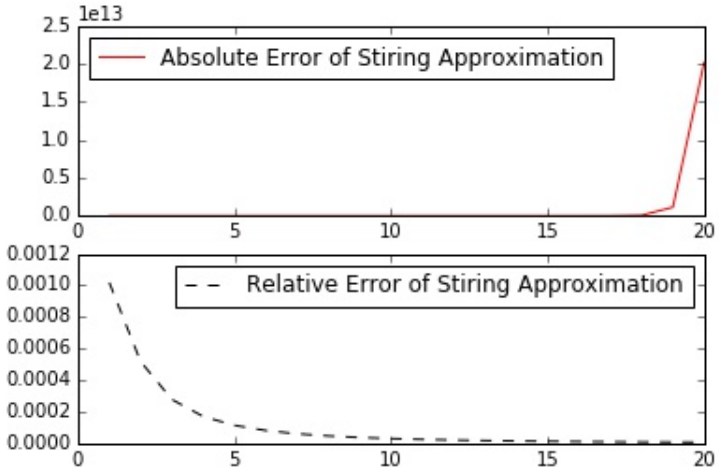


그림 1.5.2. 절대오차와 상대오차 2

```
StirAbsErr = np.zeros((Nmax,1),float)
StirRelErr = np.zeros((Nmax,1),float)

# Iteration
for ii in range(1,Nmax+1):
    StirTrue[ii-1] = math.factorial(ii)
    StirSol[ii-1] = np.sqrt(2*np.pi*ii)*((ii/ee)**ii)*(1+1/(12*ii))
    StirAbsErr[ii-1] = np.abs(StirSol[ii-1] - StirTrue[ii-1])
    StirRelErr[ii-1] = StirAbsErr[ii-1]/np.abs(StirTrue[ii-1])
OutP = np.concatenate((nn, StirTrue, StirSol, StirAbsErr, StirRelErr), axis
    =1)

# Plotting
fig = plt.figure()
ax1 = fig.add_subplot(211)
plt.plot(nn,StirAbsErr,'r-', label='Absolute Error of Stiring Approximation'
    )
plt.legend(loc='upper left', numpoints = 1 )
ax2 = fig.add_subplot(212)
plt.plot(nn,StirRelErr,'k--', label='Relative Error of Stiring Approximation
    ')
plt.legend(loc='upper right', numpoints = 1 )
fig.savefig('ComputerError103.jpg')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.5.3의 결과와 같다. ■

1.5.2 오차의 종류

과학적 컴퓨팅을 사용해서 수리적 모형을 다룰 때 발생할 수 있는 대표적인 오차는 분석하고자 하는 수리모형(mathematical model) 자체에 포함된 것이다. 예를 들어, 어떤 경제현상을 설명하기 위해서 고려해야 하는 수많은 변수들이 있으나, 이 변수들을 모두 모형에 반영할

수는 없다. 현실적으로 가능한 방법은 인과관계가 큰 변수들만을 포함하는 수리적 모형을 다루는 것이다. 따라서, 우리가 다루는 모형은 근사모형일 수 밖에 없다. 이는 수리적 모형 자체에 오차가 포함됐음을 의미한다. 또한, 입력데이터(input data)의 측정에 오류가 있어서 모형의 모수들이 주어진 현상을 나타내지 못하는 경우도 있다. 본서에서는 이렇게 분석하고자 하는 수리적 모형이나 입력데이터 자체에 포함된 오차는 다루지 않고, 주어진 수리적 모형이 완전하다고 가정한다. 그러나, 실제 문제를 풀 때는 과학적 컴퓨팅을 해서 얻은 결과가 실제 문제를 나타내는 현상을 잘 나타내는지를 반드시 조사해야 한다.

과학적 컴퓨팅을 적용해서 수리적 문제를 해결할 때 발생하는 오차를 크게 나누어 근사오차(approximation error)와 반올림오차(roundoff error)로 나눌 수 있다. 근사오차에는 연속함수를 이산함수로 바꾸는 이산화과정(discretization)에서 발생하는 이산화오차(discretization error)와 반복법(iterative method)을 적용할 때 무한 반복이 아니라 유한 반복함으로써 발생하는 수렴오차(convergence error)가 있다. 반올림오차는 컴퓨터에서 수를 표현하는 방식, 즉 부동소수점표현에 기인한 것이다. 이는 숫자를 유한 개의 비트로 표현하는 것과 이를 바탕으로 한 컴퓨터연산(computer arithematic)에 의한 것이다. 일반적으로 근사오차가 반올림오차보다 훨씬 크다고 가정하고, 근사오차를 감소시키는 알고리즘을 만들고 실장해서 정확성과 효율성을 갖는 과학적 컴퓨팅을 하고자 한다.

다음 예제들은 이산화오차에 관한 것들이다.

예제 1.5.5 함수 $\cos x$를 $x = 0.37$에서 미분하는 예를 이용해서 이산화오차에 대해서 살펴보자. 실수 x_0와 작은 양수 h에 대해서 다음 Taylor전개 성립한다.

$$f(x_0+h) = f(x_0) + \frac{1}{1!}f'(x_0)\,h + \frac{1}{2!}f''(x_0)\,h^2 + \frac{1}{3!}f'''(x_0)\,h^3 + \cdots \tag{1}$$

식 (1)에서 알 수 있듯이, 다음 식이 성립한다.

$$\left| f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h} \right| = \left| \frac{1}{2!}f''(x_0)\,h + \frac{1}{3!}f'''(x_0)\,h^2 + \cdots \right| \tag{2}$$

따라서, 다음 식이 성립한다.

$$\left| f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h} \right| \approx \frac{1}{2}\left| f''(x_0)\,h \right| = O(h) \tag{3}$$

즉, 미분값 $f'(x_0)$을 이산화값 $[f(x_0+h)-f(x_0)]/h$로 추정하면, 그 오차는 $|f''(x_0)\,h/2|$

이다. 다음 식들이 성립한다.

$$\left.\frac{d\cos x}{dx}\right|_{x=0.37} = -0.36162, \quad \left.\frac{1}{2}\frac{d^2\cos x}{dx^2}\right|_{x=0.37} = 0.46616 \tag{4}$$

이러한 이산화근사에 의한 절대오차와 상대오차를 살펴보기 위해서, 다음 MATLAB 프로그램 DiscretizationError101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename DiscretizationError101.m
% Discretization Error of cos(x)
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
x = 0.37
% Initialization
Nmax = 11; nn = 1:Nmax
h = 0.1.^(nn-1);
Value = cos(x)
DiffValue = -sin(x)
ErrBound = 1/2*cos(x)
DTrue = DiffValue*ones(1,Nmax);
DSol = (cos(x+h)-cos(x))./h;
DAbsErr = abs(DTrue-DSol);
DRelErr =  abs(DTrue-DSol)./abs(DTrue);
format short g
OutP = [ h ; DTrue; DSol; DAbsErr; DRelErr ]'
% Plotting
subplot(2,1,1)
loglog(h,DAbsErr,'r-','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold','xDir','reverse', ...
    'ylim',[ 10^(-9) 10^0 ],'ytick',[10^-8 10^-5 10^-2 10^1])
legend('\bf Absolute Error',1)
xlabel('\bf h')
subplot(2,1,2)
loglog(h,DRelErr,'k--','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold','xDir','reverse', ...
    'ylim',[ 10^(-8) 10^1 ],'ytick',[10^-8 10^-5 10^-2 10^1])
xlabel('\bf h')
legend('\bf  Relative Error',1)
saveas(gcf,'DiscretizationError101.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 다음과 같이 이산화근사에 의한 절대오차와 상대오차를 출력한다.

```
-----------------------------------------------------------
   h          True        Approx       Abs Err      Rel Err
-----------------------------------------------------------
   1      -0.36162       -0.73288       0.37126       1.0267
 0.1      -0.36162       -0.40759      0.045975      0.12714
```

```
    0.01     -0.36162     -0.36627     0.0046556      0.012874
   0.001     -0.36162     -0.36208     0.0004661     0.0012889
  0.0001     -0.36162     -0.36166     4.6616e-05    0.00012891
   1e-05     -0.36162     -0.36162     4.6616e-06    1.2891e-05
   1e-06     -0.36162     -0.36162     4.6615e-07    1.2891e-06
   1e-07     -0.36162     -0.36162     4.6707e-08    1.2916e-07
   1e-08     -0.36162     -0.36162     5.6291e-09    1.5567e-08
   1e-09     -0.36162     -0.36162     8.3345e-08    2.3048e-07
   1e-10     -0.36162     -0.36162      8.605e-07    2.3796e-06
-------------------------------------------------------------
```

또한, 이 값들의 그래프들이 그림 1.5.3에 그려져 있다. 이 그림에서는 X 축이 로그스케일(log scale)로 그려져 있음에 유의하라. 이 값들과 그림 1.5.3에서 알 수 있듯이, h가 0.1보다 작으면 절대오차는 $0.466h$와 비슷하다. ■

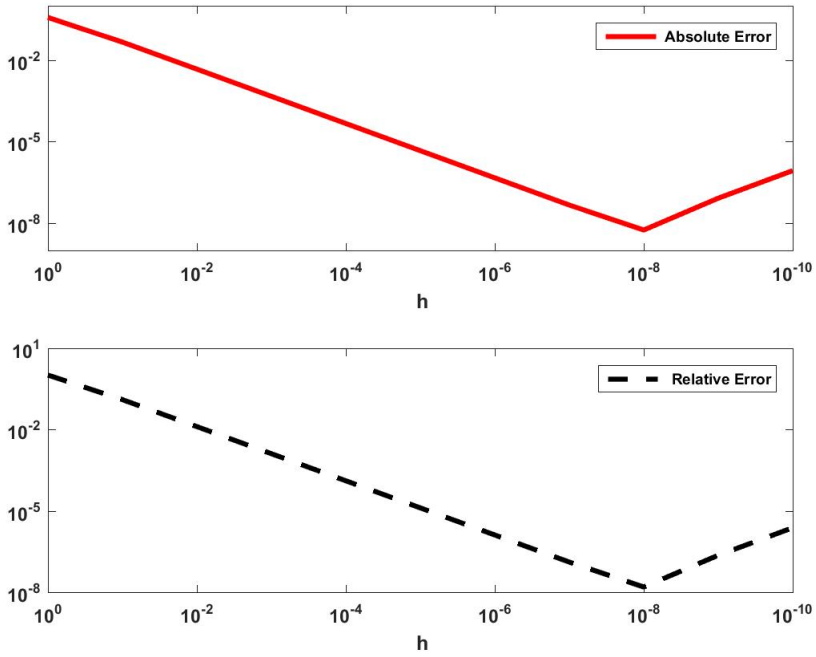


그림 1.5.3. 이산화오차 1

예제 1.5.6 Python을 사용해서 예제 1.5.5을 다시 다루기 위해서, 다음 Python프로그램 DiscretizationError101.Py를 실행해 보자.

```
1 """
```

```
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
import math

x = 0.37
# Initialization
Nmax = 11;
NN = np.linspace(1, Nmax, num=Nmax); nn = np.reshape(NN, (Nmax,1))
h = 0.1**(nn-1);
Value = np.cos(x)
DiffValue = -np.sin(x)
ErrBound = 1/2*np.cos(x)
DTrue = DiffValue*np.ones((Nmax,1), float)
# DSol = np.divide((np.cos(x+h)-np.cos(x)), h)
DSol = (np.cos(x+h)-np.cos(x))/h
DAbsErr = np.abs(DTrue-DSol)
DRelErr = np.abs(DTrue-DSol)/np.abs(DTrue)
OutP = np.concatenate((h, DTrue, DSol, DAbsErr, DRelErr), axis=1)

# Plotting
fig = plt.figure()
ax1 = fig.add_subplot(211)
ax1.invert_xaxis()
plt.loglog(h,DAbsErr,'r-', lw=3, label='Absolute Error')
plt.legend(loc='upper right', numpoints = 1 )
ax2 = fig.add_subplot(212)
ax2.invert_xaxis()
plt.loglog(h,DRelErr,'k--', lw=3, label='Relative Error')
plt.legend(loc='upper right', numpoints = 1 )
fig.savefig('DiscretizationError101.jpg')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.5.5의 결과와 같다. ■

예제 1.5.7 그림 1.5.3에서 h가 0.1보다 작으면 절대오차는 $0.466h$와 비슷하다. 이 규칙은 h가 10^{-8} 이하가 되면서 깨지는데 이러한 현상은 이 문제에서만 발생하는 것이 아니고 다른 문제에서도 자주 발생한다. 이러한 현상을 좀 더 자세히 살펴보기 위해서, 다음 MATLAB 프로그램 RoundoffError101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename RoundoffError101.m
% Roundoff Error and Discretization Error of cos(x)
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
x = 0.37
% Initialization
Nmax = 20; nn = 1:Nmax
h = 0.1.^(nn-1);
Value = cos(x)
DiffValue = -sin(x)
```

```
ErrBound = 1/2*cos(x)
DTrue = DiffValue*ones(1,length(nn));
DSol = (cos(x+h)-cos(x))./h;
DAbsErr = abs(DTrue-DSol);
DRelErr = abs(DTrue-DSol)./DTrue;
ErrLine = ErrBound*h;
format short g
OutP = [ h ; DTrue; DSol; DAbsErr; DRelErr ]'
% Plotting
loglog(h,DAbsErr,'r-o',h,ErrLine,'k-.','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold','xDir','reverse')
legend('\bf Absolute Error','\bf Discretization Error', ...
        'location','SW')
saveas(gcf,'RoundoffError101.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 다음과 같이 이산화근사에 의한 절대오차와 상대오차를 출력한다.

```
------------------------------------------------------------
     h         True       Approx      Abs Err      Rel Err
------------------------------------------------------------
     1     -0.36162     -0.73288      0.37126       1.0267
   0.1     -0.36162     -0.40759     0.045975      0.12714
  0.01     -0.36162     -0.36627    0.0046556     0.012874
 0.001     -0.36162     -0.36208    0.0004661    0.0012889
0.0001     -0.36162     -0.36166    4.6616e-05   0.00012891
 1e-05     -0.36162     -0.36162    4.6616e-06   1.2891e-05
 1e-06     -0.36162     -0.36162    4.6615e-07   1.2891e-06
 1e-07     -0.36162     -0.36162    4.6707e-08   1.2916e-07
 1e-08     -0.36162     -0.36162    5.6291e-09   1.5567e-08
 1e-09     -0.36162     -0.36162    8.3345e-08   2.3048e-07
 1e-10     -0.36162     -0.36162     8.605e-07   2.3796e-06
 1e-11     -0.36162     -0.36162    6.4116e-06    1.773e-05
 1e-12     -0.36162     -0.36171    9.5229e-05   0.00026334
 1e-13     -0.36162     -0.36193   0.00031727   0.00087738
 1e-14     -0.36162     -0.36637    0.0047582     0.013158
 1e-15     -0.36162     -0.44409     0.082474      0.22807
 1e-16     -0.36162      -1.1102      0.74861       2.0702
 1e-17     -0.36162            0      0.36162            1
 1e-18     -0.36162            0      0.36162            1
 1e-19     -0.36162            0      0.36162            1
------------------------------------------------------------
```

이 값들에서 알 수 있듯이, h가 감소함에 따라 절대오차나 상대오차가 감소하다가, h가 10^{-8} 보다 작아지면 절대오차나 상대오차는 증가한다.

이 MATLAB프로그램을 실행하면, 절대오차의 그래프를 출력한다. 이 그래프가 그림 1.5.4에 수록되어 있다. 이 그림에서도 X 축과 Y 축 모두가 로그스케일(log scale)로 그려져 있음에 유의하라. 그림 1.5.4에서도 $h = 10^{-8}$에서 절대오차가 최소임을 알 수 있다. 이 절대오차는 이산화오차와 반올림오차의 합이다. 그림 1.5.4에서 흑색 반점선은 이산화오차 $|f''(x_0)\,h/2|$를 로그-로그스케일로 나타낸 것이다. 그림 1.5.4에서 알 수 있듯이, h가 10^{-8} 이상인 경우에는 이산화오차가 반올림오차보다 훨씬 크고 h가 감소하면서 이산화오차도 감소한다. 그러나, h가 10^{-8} 미만인 경우에는 이산화오차는 작지만 반올림오차가 이산화오차보다 훨씬 커지고 또한 h가 감소하면서 반올림오차가 증가한다. 더구나, h가 10^{-8} 미만인 경우에 반올림오차는 추세선을 중심으로 위아래로 움직이는 경향을 보인다. 즉, 이산화오차에 비해 반올림오차는 움직임이 심한(erratic) 형태를 보인다. 이것이 미분방정식을 푸는 문제와 같이 미분이 포함된 문제를 수치적으로 풀 때 반올림오차보다는 이산화오차가 더 큰 방법을 선호하는 이유 중 하나이다. 한 가지 유의할 점은 $|h|$가 10^{-15}보다 작은 부분에서는 $\mathrm{fl}(x+h)$와 $\mathrm{fl}(x)$가 같다. 따라서, $|h|$가 아주 작은 부분에서는 미분의 근사값들이 변화하지 않는다. 즉, 미분값들은 h에 의존하지 않는다. ■

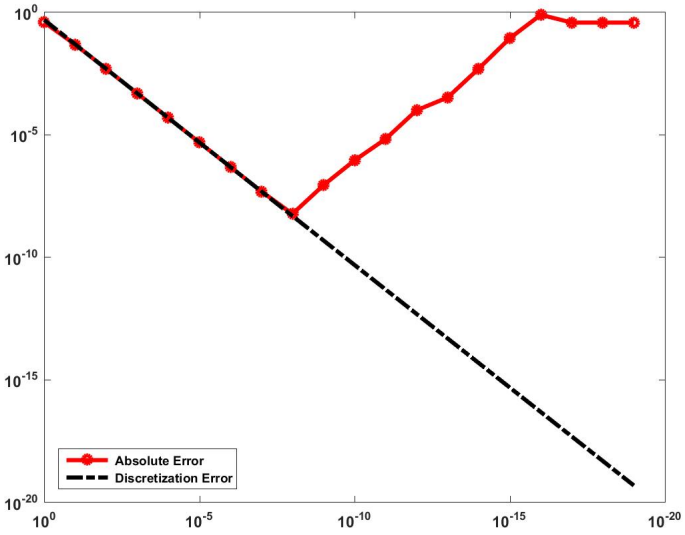


그림 1.5.4. 이산화오차 2

예제 1.5.8 Python을 사용해서 예제 1.5.7을 다시 다루기 위해서, 다음 Python프로그램

RoundoffError101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

x = 0.37
# Initialization
Nmax = 20; nn = np.linspace(1, Nmax, num=Nmax)
h = 0.1**(nn-1); h = h[:,None]
Value = np.cos(x)
DiffValue = -np.sin(x)
ErrBound = 1/2*np.cos(x)
DTrue = DiffValue*np.ones((len(nn),1), float)
DSol = (np.cos(x+h)-np.cos(x))/h
DAbsErr = np.abs(DTrue-DSol)
DRelErr = np.abs(DTrue-DSol)/DTrue
ErrLine = ErrBound*h
OutP = np.concatenate((h, DTrue, DSol, DAbsErr, DRelErr), axis=1)

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
ax.invert_xaxis()
plt.loglog(h,DAbsErr,'r-o', label='\bf Absolute Error')
plt.loglog(h,ErrLine,'k-.', lw=3, label='Discretization Error')
plt.show()
fig.savefig('RoundoffError101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.5.7의 결과와 같다. ■

예제 1.5.9 Taylor정리를 이용해서 $\exp(0.01)$을 계산할 때 발생하는 반올림오차에 대해 생각해보자. 다음 Taylor근사식이 성립한다.

$$e^x \approx 1 + \frac{1}{1!}x + \frac{1}{2!}x^2 + \cdots + \frac{1}{(p-1)!}x^{p-1} + \frac{1}{p!}x^p \tag{1}$$

MATLAB에서는 배정밀도 부동소수점표현을 사용하므로, 다음 식이 성립한다.

$$1 + 10^{-15} = 1 \tag{2}$$

즉, 다음 식이 성립한다.

$$1 + \frac{1}{6!}0.01^6 = 1 \tag{3}$$

따라서, MATLAB에서는 각 $p\,(>5)$에 대해서 다음 식이 성립한다.

$$\sum_{k=0}^{p} \frac{1}{k!} 0.01^k = \sum_{k=0}^{5} \frac{1}{k!} 0.01^k \tag{4}$$

즉, $\sum_{k=6}^{p} \frac{1}{k!} 0.01^k$는 반올림오차에 속한다. 앞에서 언급했듯이, 멱함수(polynomial)를 계산할 때는 Horner형식을 사용해서 반올림오차를 줄일 수 있다. 즉, 다음 식을 사용하면, 반올림오차와 계산량을 줄일 수 있다.

$$e^x \approx 1 + \frac{x}{1}\left[1 + \frac{x}{2}\left\{1 + \frac{x}{3}\left(1 + \frac{x}{4}[\cdots]\right)\right\}\right] \tag{5}$$

Taylor근사식에 의한 값과 Horner형식을 적용한 근사값의 오차를 살펴보기 위해서, 다음 MATLAB프로그램 HornerRule101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: HornerRule101.m
% Reducing Roundoff Errors and No of Operations by Horner Method
% Calculate exp(0.01)
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
x = 0.01;
Nmax = 15                    % p = Nmax
for nn=1:Nmax
    % Taylor Series
    T(nn) = 1+x;
    for jj = 2:nn
        T(nn) = T(nn)+1/factorial(jj)*x^(jj);
    end
    % Horner's Rule
    Hdum = 1+x/nn;
    for jj = nn-1:-1:1
        Hdum = 1 + x/jj*Hdum;
    end
    H(nn) = Hdum;
end
ErrT = [ abs(exp(x)-1), abs(exp(x)-T) ]
ErrH = [ abs(exp(x)-1), abs(exp(x)-H) ]
NT = (0:Nmax);
format long g
Outp = [ NT; ErrT; ErrH ]'
% Plotting
semilogy(NT, ErrT,'r-o', NT, ErrH,'k-.','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf Error by Taylor Series','Error by Horner Rule',1)
axis([-1, 16, 10^-16, 10^-1 ])
saveas(gcf,'HornerRule101.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 다음과 같은 Taylor근사식에 의한 값과 Horner형식을

적용한 절대오차를 출력한다.

```
-----------------------------------------------------------
  p      Taylor Approximation       Horner's rule
-----------------------------------------------------------
  1      5.01670841679402e-05      5.01670841679402e-05
  2       1.6708416783473e-07      1.67084168056775e-07
  3       4.1750114476713e-10      4.17501366811734e-10
  4      8.34443625308268e-13      8.34665669913193e-13
  5      1.11022302462516e-15      1.33226762955019e-15
  6      2.22044604925031e-16                         0
  7      2.22044604925031e-16                         0
  8      2.22044604925031e-16                         0
  9      2.22044604925031e-16                         0
 10      2.22044604925031e-16                         0
 11      2.22044604925031e-16                         0
 12      2.22044604925031e-16                         0
 13      2.22044604925031e-16                         0
 14      2.22044604925031e-16                         0
 15      2.22044604925031e-16                         0
-----------------------------------------------------------
```

또한, 이 MATLAB프로그램을 실행하면, Taylor근사식에 의한 오차와 Horner형식을 사용한 근사식의 오차를 그린다. 이 그래프들이 그림 1.5.5에 수록되어 있다. 이 그림에서는 y축이 로그스케일로 그려져 있음에 유의하라. 이 값들과 그림 1.5.5에서 알 수 있듯이, p가 5일 때까지는 Taylor근사식과 Horner형식에 의한 근사식은 동일한 결과를 보여준다. 그러나, p가 6이상에서 Taylor법에 의한 절대오차는 $2.2204 \cdot 10^{-16}$이나 Horner형식에 의한 절대오차는 0이다. 이 $2.2204 \cdot 10^{-16}$가 반올림오차이다. ■

예제 1.5.10 Python을 사용해서 예제 1.5.9을 다시 다루기 위해서, 다음 Python프로그램 HornerRule101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
import math

```

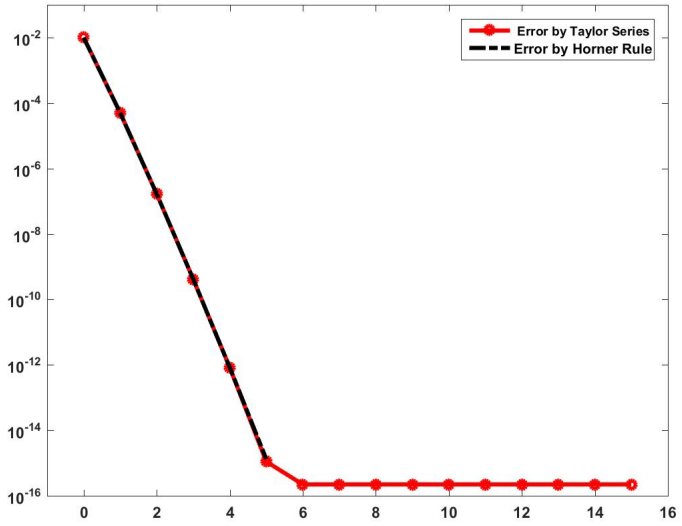


그림 1.5.5. Horner형식

```
x = 0.01
Nmax = 16
T = np.ones((Nmax,1))
H = np.zeros((Nmax,1))
NN = np.linspace(1, Nmax, num=Nmax); NT = np.reshape(NN, (Nmax,1))-1
for nn in range(1,Nmax):
    # Taylor Series
    for jj in range(1,nn+1):
        T[nn] = T[nn] + (1/math.factorial(jj))*x**jj
    # Horner's Rule
    Hdum = 1
    jjIndex = np.arange(nn-1,0,-1)
    for jj in jjIndex:
        Hdum = 1 + x/jj*Hdum
    H[nn-1] = Hdum
ErrT = np.abs(exp(x)-T)
ErrH = np.abs(exp(x)-H)
OutP = np.concatenate((NT, ErrT, ErrH), axis=1)

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
plt.semilogy(NT,ErrT,'r-o', lw=3, label='Error by Taylor Series')
plt.semilogy(NT,ErrH,'k-.', lw=3, label='Error by Horner Rule')
plt.legend(loc='upper right', numpoints = 1)
plt.xlabel('Iteration Number n')
plt.ylabel('x_n')
plt.axis( [-1, 16, 10**(-16), 10**(-1) ])
plt.show()
fig.savefig('HornerRule101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.5.9의 결과와 같다. ■

예제 1.5.11 함수 $f(x) = x^2 - 2$의 근 $\sqrt{2}$를 구하는 문제를 사용해서, 수렴오차에 대해 살펴보자. 이 근이 다음 방정식을 만족함은 자명하다.

$$x = \frac{1}{1.01}\left[0.01x + \frac{2}{x}\right] \tag{1}$$

따라서, 다음과 같은 점화식(recursive formula)을 생각해보자.

$$x_{n+1} = \frac{1}{1.01}\left[0.01x_n + \frac{2}{x_n}\right], \quad (n = 1, 2, \cdots) \tag{2}$$

초기값을 $x_1 = 1.6$으로 하는 수열 $\{x_n\}$의 그래프를 그리기 위해서, 다음 MATLAB 프로그램 ConvergenceError101.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: ConvergenceError101.m
%  Convergence Error to Calculate sqrt(2)
%  Programmed by CBS
% ------------------------------------------------------------------------
clear all, clf, format long
Nmax = 100;
x(1) = 1.6;
for n = 1:Nmax
        x(n+1) = 1/1.01*(0.01*x(n) + 2/x(n));
end
% Plotting
ii = (1:1:Nmax+1);
plot(ii,x(:),'k','LineWidth',2.5);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[1 100])
% legend('a=0','a=1','a=2','Newton-Raphson',4)
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\it x_{n}','fontsize',12,'rotation',0)
saveas(gcf,'ConvergenceError101.jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 초기값을 $x_1 = 1.6$으로 해서 계산한 $\{x_n\}$의 그래프를 출력한다. 이 그래프가 그림 1.5.6에 그려져 있다. 그림 1.5.6에서 알 수 있듯이, x_n이 $\sqrt{2}$로 수렴한다. 그러나, 반복횟수가 100이 되어도 오차가 여전히 크다. 즉, 이 알고리즘은 수렴오차를 내포하고 있다. ■

예제 1.5.12 Python을 사용해서 예제 1.5.11을 다시 다루기 위해서, 다음 Python프로그램 ConvergenceError101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
```

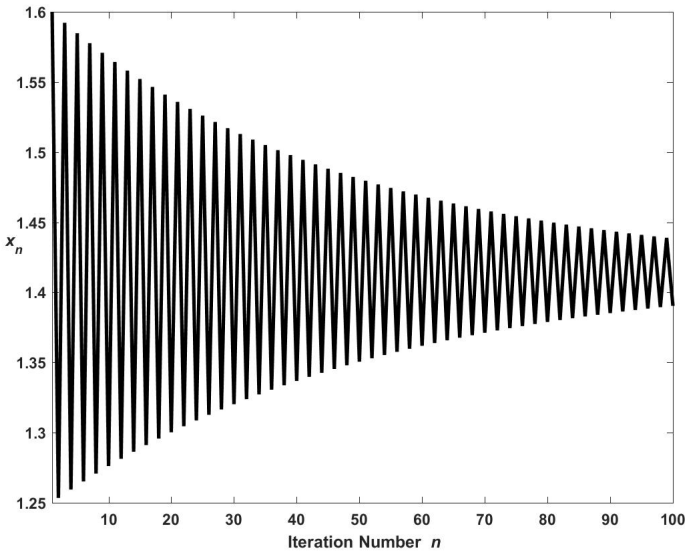


그림 1.5.6. 수렴오차

```
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

Nmax = 100
x = np.zeros((Nmax,1))
x[0] = 1.6
for n in range(1,Nmax):
      x[n] = 1/1.01*(0.01*x[n-1] + 2/x[n-1])

# Plotting
ii = np.linspace(0,Nmax-1,num=Nmax)
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(ii,x,'k', lw=2, label='\bf Absolute Error')
plt.xlabel('Iteration Number n')
plt.ylabel('x_n')
plt.show()
fig.savefig('ConvergenceError101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.5.11의 결과와 같다. ■

1.5.3 반올림오차

미국정부의 GAO(General Accounting Office)의 IMTD(Information Management and Technology Division)는 1992년 2월 4일 미하원의 Howard Wolpe 소위원장(Chairman of Subcommittee on Investigations and Oversight Committee on Science, Space, and Technol-

ogy)에게 *Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi*라는 제목의 보고서 GAO/IMTEC-92-26를 제출하였다. 이 보고서에 대해서는 다음 웹사이트들을 참조하라.

http://fas.org/spp/starwars/gao/im92026.htm

http://www.ima.umn.edu/ arnold/disasters/patriot.html

이 보고서의 앞 부분은 다음과 같다.

> 친애하는 의장님:
>
> 사막의 폭풍작전(Operation Desert Storm) 중이던 1991년 2월 25일에 사우디아라비아의 다란(Dhahran)에서 작동하던 패트리어트미사일방어시스템(Patriot missile defense system)은 상대방 스커드미사일(Iraqi Scud missile)을 추적하고 격추하는데 실패했습니다. 이 스커드미사일은 주둔하던 미육군부대를 공격해서 28명의 미국인을 죽였습니다. (참고: 또한 100여명의 부상자가 발생했음) 이 보고서는 이 사건에 컴퓨터소프트웨어 문제가 포함되었는지, 만약 그렇다면 어떤 소프트웨어에 문제가 있었는지, 그리고 그 문제점을 해결했는지에 대한 귀하의 질의에 답하는 것입니다.
>
> 요약하면 다란의 패트리오트미사일이 스커드미사일을 격추시키지 못한 이유는 시스템의 무기제어컴퓨터에 사용된 소프트웨어문제에 기인합니다. 이 소프트웨어문제로 인해서 미사일배터리의 추적계산(tracking calculation)에 오차가 있었고, 더구나 미사일배터리가 오래 작동할수록 이 오차가 커졌습니다. 스커드미사일의 공격이 있던 당시 미사일배터리는 이미 100시간 동안 연속적으로 작동하였고, 결과적으로 공격해오는 스커드미사일의 위치를 잘못 판단할 정도로 오차가 커졌습니다.
>
> 패트리어트미사일은 스커드미사일을 방어하는데 사용된 적이 없으며, 오랜기간 동안 연속적으로 작동시킬 것이라고 예상하지도 않았습니다. 스커드미사일의 공격을 받기 2주전 미육군은 이 시스템을 8시간 이상 작동시킬 경우 정확도에 문제가 생긴다는 이스라엘 데이터를 받았고, 이에 미육군은 이 소프트웨어를 수정해서 시스템의 정확성을 증가시켰습니다. 그러나 수정된 소프트웨어는 1991년 2월 26일까지 다란에 도착하지 않았습니다.
>
> ...

이 문제는 근본적으로 미사일배터리의 컴퓨터가 24비트 고정점표현(fixed point representation)을 사용해서 발생한 것이다. 이진법으로 0.1을 표현하면 다음과 같다.

$$[0.1]_{10} = [0.0001100110011001100110011001100110011\cdots]_2 \tag{1.5.2}$$

이 미사일배터리에서는 다음 식이 사용되었다.

$$[0.1]_P \doteq [0.00011001100110011001100]_2 \tag{1.5.3}$$

따라서, 절대오차는 $[0.1]_{10} - [0.1]_P = [0.000000095]_{10}$ 이다. 이 미사일배터리에서는 시간이 0.1초 단위로 계산되었고, 따라서 매초 마다 $10 \times [0.000000095]_{10}$ 초만큼 오차가 생겼다. 스커드공격을 받았을 때 이미 이 미사일배터리는 100시간 동안 작동했으므로, 이 미사일배터리 시간의 총오차는 다음과 같다.

$$100 \times 24 \times 60 \times 60 \times 10 * [0.000000095]_{10} = 0.34(sec) \tag{1.5.4}$$

스커드미사일의 속도가 $8,800km/hr$ 이므로 이 0.34초 동안 스커드미사일은 800미터 이상을 날아간다. 그러니 패트리오트미사일이 스커드미사일을 맞출 수가 없었다.

이 예에서도 알 수 있듯이, 각 연산단계에서 반올림오차는 작지만 이들이 쌓이면 결코 무시할 수 없는 결과가 발생할 수도 있다. 이 소절에서 반올림오차를 좀 더 자세히 살펴보자. 앞에서 정의했듯이, 실수 x 의 부동소수점표현 $\mathrm{fl}(x)$ 는 다음과 같다.

$$\mathrm{fl}(x) = \pm \left[1 + \frac{d_{-1}}{2^1} + \frac{d_{-2}}{2^2} + \cdots + \frac{d_{-t}}{2^t}\right] \times 2^t \tag{1.5.5}$$

MATLAB에서는 마지막 비트의 수 2^{-t} 를 eps라고 한다. 이 부동소수점표현 $\mathrm{fl}(x)$ 의 상대오차는 다음과 같다.

$$\text{relative error of } \mathrm{fl}(x) = \frac{|\mathrm{fl}(x) - x|}{|x|} \tag{1.5.6}$$

IEEE 754에 의하면, 다음 식이 성립한다.

$$\frac{|\mathrm{fl}(x) - x|}{|x|} \le \eta \doteq \frac{1}{2} \cdot 2^{-t} \tag{1.5.7}$$

즉, $\eta = \mathrm{eps}/2$ 이다. 이 η 를 반올림단위(unit roundoff, rounding unit), 기계정밀도(machine precision), 기계엡실론(machine epsilon), macheps 등으로 부른다. MATLAB에서 디폴트로 사용하는 부동소수점표현인 배정밀도(double precision)에서는 부동점워드(floating point word)가 64비트이고 이중 52비트가 가수(mantissa, fraction)로가수(mantissa) 사용되고

있다. 따라서, macheps는 다음과 같다.

$$\eta = \frac{1}{2} \cdot 2^{-52} = 2^{-53} \approx 0.1110 \cdot 10^{-15} \tag{1.5.8}$$

즉, 배정밀도에서는 52개 비트, 즉 십진법으로 15자리가 유의적(significant)이다. 단정밀도(single precision)인 경우에는 부동점워드가 32비트이고 이중 23비트가 가수로 사용되고 있다. 따라서, macheps는 다음과 같다.

$$\eta = \frac{1}{2} \cdot 2^{-23} = 2^{-24} \approx 0.5960 \cdot 10^{-7} \tag{1.5.9}$$

즉, 단정밀도에서는 23개 비트, 즉 십진법으로 7자리가 유의적이다.

부동소수점들 자체에 사칙연산(+,-,×,÷, arithematic operations)을 적용해도 반올림오차가 발생한다. 이 반올림오차는 상대적으로 상당히 클 수도 있다. IEEE 754에서는 계산의 중간에 가드디짓(guard digits)이라는 추가 디짓을 사용해서 이 반올림오차를 줄인다. 즉, IEEE 754에서는 정확한 반올림(exact rounding)을 요구하는데, 이 의미는 정확한 반올림을 행한 값이 먼저 해당하는 사칙연산을 정확하게 해서 얻은 다음 이 값을 가장 가까운 부동소수점으로 반올림한 값과 같아야 한다는 것이다. 즉, 각 사칙연산의 결과로 발생하는 상대오차가 macheps η보다 작도록하는 것이다. 따라서, 정확한 반올림을 하는 부동소수점연산은 다음 식들을 만족한다.

$$\mathrm{fl}(\mathrm{fl}(x) + \mathrm{fl}(y)) = [\mathrm{fl}(x) + \mathrm{fl}(y)]\,[1 + \epsilon_1] \tag{1.5.10}$$

$$\mathrm{fl}(\mathrm{fl}(x) - \mathrm{fl}(y)) = [\mathrm{fl}(x) - \mathrm{fl}(y)]\,[1 - \epsilon_2] \tag{1.5.11}$$

$$\mathrm{fl}(\mathrm{fl}(x) \times \mathrm{fl}(y)) = [\mathrm{fl}(x) \times \mathrm{fl}(y)]\,[1 + \epsilon_3] \tag{1.5.12}$$

$$\mathrm{fl}(\mathrm{fl}(x) \div \mathrm{fl}(y)) = [\mathrm{fl}(x) \div \mathrm{fl}(y)]\,[1 + \epsilon_4] \tag{1.5.13}$$

여기서 $|\epsilon_i| < \eta,\ (i = 1, 2, 3, 4)$이다. 하나의 사칙연산 결과 반올림오차가 작다고 반올림오차에 대해 조심을 하지 않아도 될까? 물론 그 답은 부정적이다. 앞에서도 간단히 언급했듯이, 반올림오차는 근사적으로 연산횟수의 선형함수이다. 즉, 반올림오차의 축적속도(accumulation speed)는 연산횟수에 비례한다. 일반적으로 우리가 다루는 문제의 연산횟수는 이러한 반올림오차의 축적량이 문제가 될 정도로 크지는 않다. 그래도 프로그래밍할 때 다음과 같은 점들을 고려해야 한다. 첫째, 만약 $|x|$와 $|y|$가 크게 다르면, $x + y$의 절대오차가 크다. 둘째, 만약 $|x| \ll 1$이면, y/x의 절대오차와 상대오차가 크다. 셋째, 만약 $|x| \gg 1$이면, xy의 절대오차와

상대오차가 크다. 넷째, 만약 x와 y가 비슷하면, $x-y$의 상대오차가 크다. 네 번째 경우를 소거오차(cancellation error)라고 부르는데, 프로그래밍할 때 이 경우를 특히 조심해야 한다.

예제 1.5.13 다음 식들을 만족하는 x와 y를 생각해보자.

$$x = 1 + \epsilon_1, \quad y = 1 + \epsilon_2, \quad \left|\epsilon_1\right| < \eta, \quad \left|\epsilon_2\right| < \eta \tag{1}$$

따라서, $\mathrm{fl}(x) = 1$이고 또한 $\mathrm{fl}(y) = 1$이다. 다음 식이 성립한다.

$$\mathrm{fl}(x) - \mathrm{fl}(y) = 0 \tag{2}$$

그림 1.5.7을 살펴보자. 그림 1.5.7에서 청색으로 표시된 마름모꼴 안에서는 식 $\left|\epsilon_1 - \epsilon_2\right| < \eta$이 성립한다. 즉, 다음 식들이 성립한다.

$$|x - y| = |\epsilon_1 - \epsilon_2| < \eta \tag{3}$$

따라서, 다음 식이 성립한다.

$$|\mathrm{fl}(x - y)| = 0 \tag{4}$$

즉, 반올림오차가 존재하지 않는다. 반면에, 마름모꼴과 정사각형 사이에서는 식 $|\epsilon_1 - \epsilon_2| \geq \eta$가 성립한다. 즉, 다음 식이 성립한다.

$$|x - y| = |\epsilon_1 - \epsilon_2| > \eta \tag{5}$$

따라서, 다음 식이 성립한다.

$$\mathrm{fl}(x - y) \neq 0 \tag{6}$$

■

예제 1.5.14 함수 $g(x)$가 어떤 구간 (a, b)에서 미분가능하다고 하면, 각 점 $x \in (a, b)$와 절대값이 작은 상수 h에 대해서 다음 근사식이 성립한다.

$$g(t + h) - g(t) \approx g'(t)h \tag{1}$$

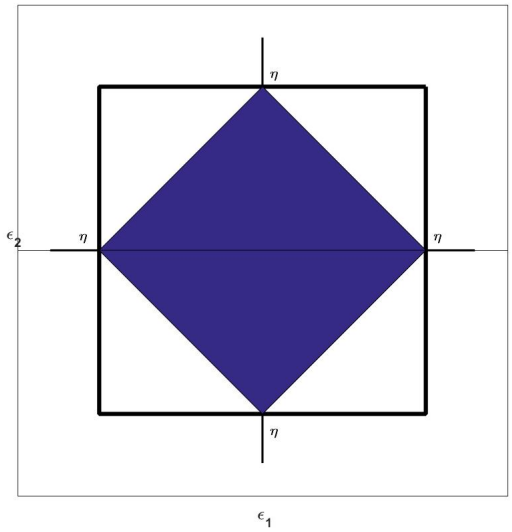


그림 1.5.7. 소거오차

함수 $g(x)$가 이 구간에서 평활하고 h는 고정되었으므로, 식 (1)에서 알 수 있듯이 $g(t+h)$와 $g(t)$는 비슷해야 한다. 그러나, $|\text{fl}(g(x)) - g(x)|$와 $|\text{fl}(g(x+h)) - g(x+h)|$는 별 상관이 없다. 즉, 집합 $\{|\text{fl}(g(x)) - g(x)| \mid x \in (a,b)\}$는 랜덤프로세스(random process)이다. 즉, $|\text{fl}(g(x)) - g(x)|$와 $|\text{fl}(g(x+h)) - g(x+h)|$ 각각은 작지만, 상대오차는 현저하게 크거나 작을 수 있다. 이는 $g(t+h)$와 $g(t)$가 비슷하기때문에 소거오차가 발생함을 의미한다.

소거오차를 살펴보기 위해서, 예제 1.4.5을 다시 생각해보자. 즉, 구간 $[0\ \pi/2]$에서 사인함수를 살펴보자. 이 구간에서 사인함수는 매우 평활하나, 사인값을 배정밀도로 표현한 값과 단정밀도로 표현한 값의 차이는 매우 변화가 심하다. 이 차이를 그리기 위해서, 다음 MATLAB프로그램 CancellationError101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename CancellationError101.m
% Cancellation Error
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
xx = 0:0.001:pi/2;
fDouble = sin(xx);
fSingle = single(fDouble);
fDifference = fDouble-fSingle;
fRelErr = fDifference./fDouble;
% Plotting
plot(xx,fRelErr,'k-','LineWidth',1)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0 pi/2])
xlabel('\bf x')
legend('\bf Relative Error between Dbl and Sgl Precisions', ...
       'location','SouthEast')
saveas(gcf,'CancellationError101.jpg')
% Test if \eta = eps('single')/2
```

```
fMaxRelErr = max(abs(fRelErr))
eta = eps('single')/2
Ratio1 = fMaxRelErr/eta
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 그림 1.5.8을 출력한다. 이 그림에서 알 수 있듯이 반올림오차는 변화가 심하다. 부동소수점표현의 상대오차의 최대값은 단정밀도의 η와 비슷해야 할 것이다. 이 예제에서는 $\max_x |\mathrm{fl}(g(x)) - g(x)|/|g(x)| = 5.7134 \cdot 10^{-8}$ 이고 단정밀도의 η는 $\mathrm{eps}/2 = 5.9605 \cdot 10^{-8}$ 이다. ■

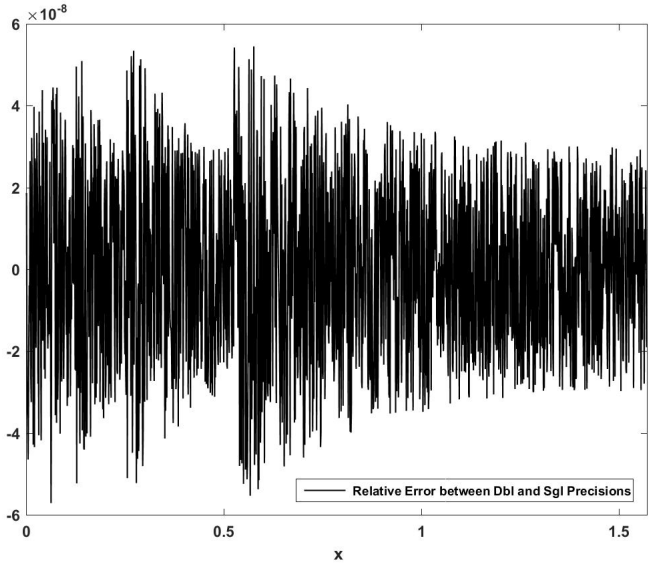


그림 1.5.8. 배정밀도와 단정밀도

예제 1.5.15 Python을 사용해서 예제 1.5.14을 다시 다루기 위해서, 다음 Python프로그램 CancellationError101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

xx = np.linspace(0, np.pi/2, num=1571)
fDouble = np.sin(xx)
fSingle = np.float32(fDouble)
fDifference = fDouble-fSingle
fRelErr = fDifference/fDouble

# Plotting
fig = plt.figure()
```

```
ax = fig.add_subplot(111)
plt.plot(xx,fRelErr,'k-',
         label='Relative Error between Dbl and Sgl Precisions')
plt.legend(loc='lower left', numpoints = 1 )
plt.xlabel('x'); plt.xlim(0, np.pi/2)
plt.show()
fig.savefig('CancellationError101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.5.14의 결과와 같다. ■

1.5.4 오차, 안정성과 조건수

컴퓨터를 사용해서 함수값 $f(x)$를 계산할 때 발생하는 다양한 오차를 다음과 같이 정의하자.

정의 1.5.1

입력값 x가 $\hat{x} \doteq x + \Delta x$로 컴퓨터에 입력되는 경우, 출력값이 $\hat{y} \doteq y + \Delta y$라고 하자. 또한, 다음 상수를 정의하자.

$$\delta \doteq \inf \{ |\Delta x| \mid \hat{y} = f(x + \Delta x) \}$$

이들에 대한 오차들은 다음과 같이 정의된다.

(a) 절대전향오차(absolute forward error): $|y - \hat{y}|$

(b) 상대전향오차(relative forward error): $\frac{|y-\hat{y}|}{|y|}$

(c) 절대후향오차(absolute backward error): δ

(d) 상대후향오차(relative backward error): $\frac{\delta}{|x|}$

(e) 조건수(condition number):

$$c(f, x) \doteq \frac{|x|}{|f(x)|} |f'(x)|$$

전향오차와 후향오차에 대해서는 그림 1.5.9를 참조하라.

상대후향오차가 $O(|\Delta x|/|x|)$인 알고리즘을 (수치적으로) 후향안정적(backward stable)이라 한다. 후향안정알고리즘(backward stable algorithm)이 생성하는 크기의 상대전향오차

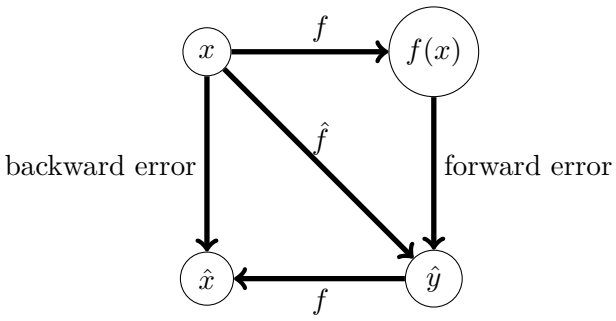


그림 1.5.9. 전향오차와 후향오차

(relative forward error)를 생성하는 알고리즘을 (수치적으로) 전향안정적(forward stable)이라 한다. 따라서, 후향안정적 알고리즘이면 전향안정적이다. 만약 $c(f,x)$가 1에 가까우면, 좋은 조건이라(well conditioned) 한다. 만약 $c(f,x)$가 1보다 아주 크면, 나쁜 조건이라(badly conditioned) 한다. 만약 $c(f,x)$가 1보다 아주 작으면, 정보손실(loss of information)에 의해서 나쁜 알고리즘이 될 수도 있다. 일반적으로 조건수에 후향오차를 곱한 값은 전향오차보다 작지 않다. 조건수가 작고 동시에 안정적인 알고리즘은 믿을 수 있는 결과를 낸다. 그러나, 조건수가 크거나 또는 안정적이 아니면, 그 알고리즘이 좋은 결과를 나타낸다고 보장할 수 없다.

예제 1.5.16 벡터 $\boldsymbol{x} = [x_1,\, x_2,\, \cdots x_n\,]^n\, (\in \mathbb{R}^n)$에 대해서 $l^\infty-$노름을 다음과 같이 정의하자.

$$\|\boldsymbol{x}\|_\infty \doteq \max_{1\le i\le n} |x_i\,| \tag{1}$$

만약 $\|\boldsymbol{x}\|_\infty = 0$이면, 식 $\max\limits_{1\le i\le n} |x_i\,| = 0$이 성립한다. 따라서, 다음 명제가 성립한다.

$$\|\boldsymbol{x}\|_\infty = 0 \quad \Rightarrow \quad \boldsymbol{x} = \boldsymbol{0} \tag{2}$$

임의의 실수 α에 대해서 다음 식들이 성립한다.

$$\|\alpha\,\boldsymbol{x}\|_\infty = \max_{1\le i\le n} |\alpha\, x_i\,| = |\alpha| \max_{1\le i\le n} |x_i\,| = |\alpha|\, \|\boldsymbol{x}\|_\infty \tag{3}$$

임의의 $\boldsymbol{x} = [x_1,\, x_2,\, \cdots x_n\,]^n\, (\in \mathbb{R}^n)$와 $\boldsymbol{y} = [y_1,\, y_2,\, \cdots y_n\,]^n\, (\in \mathbb{R}^n)$에 대해서 다음 식들이

성립한다.

$$\|\boldsymbol{x}+\boldsymbol{y}\|_\infty = \max_{1\le i\le n}|x_i+y_i| \le \max_{1\le i\le n}[|x_i|+|y_i|] \le \max_{1\le i\le n}|x_i| + \max_{1\le i\le n}|y_i| \tag{4}$$

따라서, 다음 식이 성립한다.

$$\|\boldsymbol{x}+\boldsymbol{y}\|_\infty \le \|\boldsymbol{x}\|_\infty + \|\boldsymbol{y}\|_\infty \tag{5}$$

식 (2), 식 (3) 그리고 식 (5)에서 알 수 있듯이, $\|\cdot\|_\infty$는 노름이다.

각 $i(=1, 2, \cdots, n)$에 대해서 식 $|x_i| \le \|\boldsymbol{x}\|$이 성립한다. 따라서 다음 식들이 성립한다.

$$\|\boldsymbol{x}\|_\infty \le \max_i |x_i| \le \|\boldsymbol{x}\| \tag{6}$$

다음 식들이 성립한다.

$$\|\boldsymbol{x}\| \le \sum_{i=1}^{n}|x_i|^2 \le \sum_{i=1}^{n}\|\boldsymbol{x}\|_\infty \le n\|\boldsymbol{x}\|_\infty \tag{7}$$

식 (6)과 식 (7)에서 알 수 있듯이, 다음 부등식들이 성립한다.

$$\|\boldsymbol{x}\|_\infty \le \|\boldsymbol{x}\| \le n\|\boldsymbol{x}\|_\infty \tag{8}$$

식 (8)에서 알 수 있듯이, $l^\infty-$노름은 l^2-노름보다 아주 크거나 아주 작지도 않다. 따라서 $l^\infty-$노름과 l^2-노름은 동치적(equivalent) 노름들이다.

벡터 $\boldsymbol{x}(\neq \boldsymbol{0})$와 근사벡터 $\hat{\boldsymbol{x}}$에 대해서, 상태오차들을 다음과 같이 정의하자.

$$\epsilon(\hat{\boldsymbol{x}}, \boldsymbol{x}) \doteq \frac{\|\hat{\boldsymbol{x}}-\boldsymbol{x}\|}{\|\boldsymbol{x}\|}, \quad \epsilon_\infty(\hat{\boldsymbol{x}}, \boldsymbol{x}) \doteq \frac{\|\hat{\boldsymbol{x}}-\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} \tag{9}$$

다음 식들이 성립한다.

$$\|\hat{\boldsymbol{x}}-\boldsymbol{x}\| \le \sqrt{n}\|\hat{\boldsymbol{x}}-\boldsymbol{x}\|_\infty, \quad \frac{1}{\|\boldsymbol{x}\|} \le \frac{1}{\|\boldsymbol{x}\|_\infty} \tag{10}$$

식 (10) 에서 알 수 있듯이, 다음 식들이 성립한다.

$$\epsilon(\hat{\boldsymbol{x}}, \boldsymbol{x}) = \frac{\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \sqrt{n} \frac{\|\hat{\boldsymbol{x}} - \boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} = \sqrt{n}\, \epsilon_\infty(\hat{\boldsymbol{x}}, \boldsymbol{x}) \tag{11}$$

■

제 1.6 절 예제들

1.6.1 Horner 형식과 반올림오차

이 소절의 내용은 Ascher & Grief [11, p. 59] 에서 인용한 것이다. 다음 함수를 살펴보자.

$$p(x) = [x - 2]^9 \tag{1.6.1}$$

이 함수를 전개하면, 다음과 같다.

$$\begin{aligned} p(x) = x^9 - 18x^8 + 144x^7 - 672x^6 + 2016x^5 - 4032x^4 \\ + 5376x^3 - 4608x^2 + 2304x - 512 \end{aligned} \tag{1.6.2}$$

이 함수를 다음과 같이 Horner 형식으로 쓸 수 있다.

$$\begin{aligned} p(x) = \Big\{ \Big(\Big[\Big\{ \big([\{(x - 18)x + 144\}x - 672]x + 2016 \big)x - 4032 \Big\}x \\ + 5376 \Big]x - 4608 \Big)x + 2304 \Big\}x - 512 \end{aligned} \tag{1.6.3}$$

일반적 멱함수 표기법인 식 (1.6.1) 에 의한 절단오차보다는 Horner 형식 (1.6.3) 에 의한 절단오차가 더 작다. 그러나, 반올림오차도 그러할까? 이에 대해 생각해보자.

예제 1.6.1 식 (1.6.1) 과 식 (1.6.3) 을 사용해서 함수 $\{p(x) \mid 1.92 \leq x \leq 2.08\}$ 를 그리기 위해서, 다음 MATLAB 프로그램 HornerRoundoffError101DY.m 을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: HornerRoundoffError101DY.m
% Regularized Regression with Newton-Raphson Method
% Programmed by CDY
% ------------------------------------------------------------------------
clear all, close all, format long g
```

```
% Making Problem

% Calculating Coefficients
nMax = 9
alpha = -2
for rr = 1:nMax
    a(rr) = alpha^(nMax-rr)*nchoosek(nMax,rr);
end
a0 = alpha^nMax
% Calculating function values
xNum = 8001;
x = linspace(1.92,2.08,xNum);
for ii=1:xNum
   b(nMax) = a(nMax);
   for kk = nMax-1:-1:1
       b(kk) = a(kk)+b(kk+1)*x(ii);
   end
   fHorner(ii) = a0+b(1)*x(ii);
   fDirect(ii) = (x(ii)-2).^9;
   Difference(ii) = fHorner(ii) - fDirect(ii);
end
a
b
% Plotting
subplot(1,2,1)
plot(x,fHorner,'r-','LineWidth',1)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[1.92 2.08])
hold on
plot(x,fDirect,'k-','LineWidth',2.5)
legend('\bf Horner Method','\bf Direct Calculation','location','SW')
hold off
% title('\bf Horner Method')
subplot(1,2,2)
plot(x,Difference,'r-',x,x-x,'k-','LineWidth',1)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[1.98 2.02])
legend('\bf Difference','location','SW')
saveas(gcf,'HornerRoundoffError101DY.jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 HornerRoundoffError101DY.m을 실행하면, 그림 1.6.1이 그려진다. 그림 1.6.1의 좌측 그래프에서 흑색 실선은 식 (1.6.1)을 이용해서 그린 함수 $p(x)$이고 적색 실선은 Horner형식 (1.6.3)을 이용해서 그린 함수 $p(x)$이다. 이 그래프에서 알 수 있듯이, 식 (1.6.1)을 이용한 흑색 실선에서는 절단오차(truncation error)나 반올림오차(roundoff error)가 나타나지 않는다. 반면에, Horner형식 (1.6.3)을 이용한 적색 실선의 급격한 변동은 반올림오차에 의한 것이다. Demmel [19, p. 7]은 이러한 반올림오차에 의한 현상을 잡음(noise)이라 불렀다. ■

예제 1.6.2 Python을 사용해서 예제 1.6.1을 다시 다루기 위해서, 다음 Python프로그램 HornerRoundoffError101.Py를 실행해 보자.

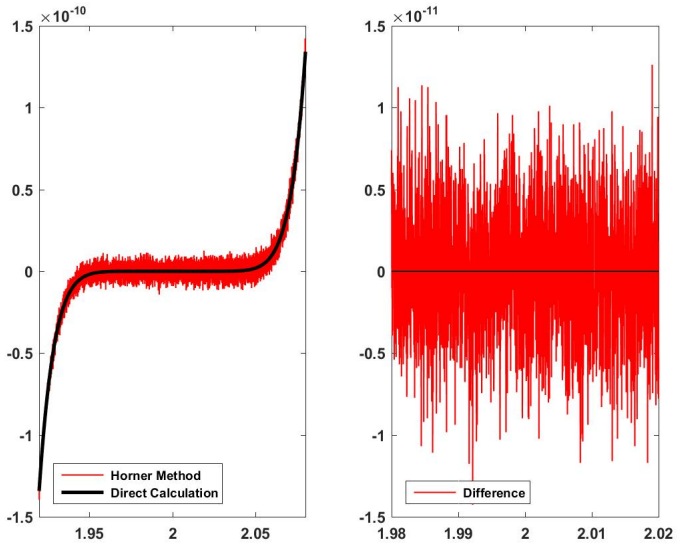


그림 1.6.1. Horner법과 반올림오차

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
import math

def combination(n,k):
    N1 = math.factorial(n);   # logN1 = np.log(N1)
    D1 = math.factorial(k);   # logD1 = np.log(D1)
    D2 = math.factorial(n-k); # logD2 = np.log(D2)
    Combi = N1/(D1*D2)        # A = exp(ln(A) makes error
    return Combi

# Calculating Coefficients
nMax = 9
alpha = -2
a = np.zeros((nMax,1),float)
for rr in range(1,nMax+1):
    a[rr-1] = alpha**(nMax-rr)*combination(nMax,rr)
a0 = alpha**nMax

# Calculating function values
xNum = 8001
x = np.linspace(1.92,2.08, num=xNum)
b = np.zeros((nMax,1),float)
fHorner = np.zeros((xNum,1),float)
fDirect = np.zeros((xNum,1),float)
Difference = np.zeros((xNum,1),float)

for ii in range(1,xNum+1):
   b[nMax-1] = a[nMax-1]
   for kk in range(nMax-1,0,-1):
       b[kk-1] =  b[kk]*x[ii-1] + a[kk-1]
   fHorner[ii-1] = b[0]*x[ii-1] + a0
```

```
   fDirect[ii-1] = (x[ii-1]-2)**9
Difference = fHorner - fDirect

# Plotting
fig = plt.figure()
ax1 = fig.add_subplot(121)
plt.plot(x,fHorner,'r-', lw=1, label='Horner Method')
plt.plot(x,fDirect,'k-', lw=2, label='Direct Calculation')
plt.xticks(np.arange(1.95, 2.06, 0.05 ))
plt.legend(loc='lower right', numpoints=1)
ax2 = fig.add_subplot(122)
plt.plot(x,Difference,'r-', lw=1, label='Difference')
plt.plot(x,x-x,'k-', lw=2)
plt.legend(loc='lower right', numpoints=1)
plt.xticks(np.arange(1.95, 2.06, 0.05 ))
fig.savefig('HornerRoundoffError101DYpy.jpg')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.6.1의 결과와 같다. ■

1.6.2 절단오차와 반올림오차

다음과 같은 2차도함수의 근사함수를 정의하자.

$$\frac{d^2f}{dx^2} \approx \frac{\delta^2 f}{\delta x^2} \doteq \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{1.6.4}$$

여기서 f는 C^3에 속하는 함수이고, h는 작은 양수이고, x는 고정된 상수이다. 식 (1.6.4)를 사용해서 2차도함수를 계산하는 경우에 발생하는 오차에 대해서 살펴보자.

다음 식들이 성립한다.

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2!}f''(x)h^2 + \frac{1}{3!}f'''(x)h^3 + O\left(h^4\right) \tag{1.6.5}$$

$$f(x-h) = f(x) - f'(x)h + \frac{1}{2!}f''(x)h^2 - \frac{1}{3!}f'''(x)h^3 + O\left(h^4\right) \tag{1.6.6}$$

식 (1.6.5)와 식 (1.6.6)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{f(x+h) - 2f(x) + f(x-h)}{h^2} = f''(x) + O\left(h^4\right) \tag{1.6.7}$$

식 (1.6.4)와 식 (1.6.7)에서 알 수 있듯이, 근사오차의 일종인 절단오차(truncation error)는

다음과 같다.

$$\left|\frac{\delta^2 f}{\delta x^2} - \frac{d^2 f}{dx^2}\right| = O\left(h^2\right) = c_1 h^2 \tag{1.6.8}$$

여기서 $c_1\,(>0)$은 상수이다.

식 (1.6.4)를 컴퓨터에 실장할(implementing) 때 발생하는 반올림오차를 살펴보자. 함수 f의 수치적 실장을 $\hat{f}$라고 하면, 2차도함수의 실장식(implementing equation)은 다음과 같다.

$$\frac{\delta^2 \hat{f}}{\delta x^2} = \frac{\hat{f}(x+h) - 2\hat{f}(x) + \hat{f}(x-h)}{h^2} \tag{1.6.9}$$

실장함수 $\hat{f}$가 다음 식을 만족한다고 하자.

$$\frac{\left|\hat{f}(x) - f(x)\right|}{|f(x)|} \le \eta \tag{1.6.10}$$

여기서 η는 머신엡실론(machine epsilon)이다. 다음 식들이 성립한다.

$$\left|\frac{\delta^2 \hat{f}}{\delta x^2} - \frac{\delta^2 f}{\delta x^2}\right| \tag{1.6.11}$$

$$= \left|\frac{\hat{f}(x+h) - 2\hat{f}(x) + \hat{f}(x-h)}{h^2} - \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}\right| \tag{1.6.12}$$

$$\le \frac{1}{h^2}\left\{\left|\hat{f}(x+h) - f(x+h)\right| + 2\left|\hat{f}(x) - f(x)\right| + \left|\hat{f}(x-h) - f(x-h)\right|\right\} \tag{1.6.13}$$

식 (1.6.10)을 식 (1.6.13)에 대입하면, 반올림오차가 다음과 같음을 알 수 있다.

$$\left|\frac{\delta^2 \hat{f}}{\delta x^2} - \frac{\delta^2 f}{\delta x^2}\right| \le \frac{\eta}{h^2}\left\{|f(x+h)| + 2\,|f(x)| + |f(x-h)|\right\} = c_2 \frac{\eta}{h^2} \tag{1.6.14}$$

여기서 $c_2\,(>0)$는 상수이다.

식 (1.6.8)과 식 (1.6.13)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\left|\frac{\delta^2 \hat{f}}{\delta x^2} - \frac{d^2 f}{dx^2}\right| \le \left|\frac{\delta^2 f}{\delta x^2} - \frac{d^2 f}{dx^2}\right| + \left|\frac{\delta^2 \hat{f}}{\delta x^2} - \frac{\delta^2 f}{\delta x^2}\right| \le c_1 h^2 + c_2 \frac{\eta}{h^2} \tag{1.6.15}$$

즉, 총오차의 상한은 $c_1h^2 + c_2\eta/h^2$ 이다. 다음 부등식이 성립한다.

$$c_1h^2 + c_2\frac{\eta}{h^2} \le 2\sqrt{c_1h^2 \cdot c_2\frac{\eta}{h^2}} = 2\sqrt{c_1\, c_2\, \eta} \tag{1.6.16}$$

여기서 등호는 $h = [c_2\eta/c_1\,]^{1/4}$ 에서 성립한다.

예제 1.6.3 식 (1.6.4)을 사용해서 함수 $f(x) = \exp(x)$ 의 점 $x = 1$ 에서 2차도함수값을 계산할 때 발생하는 오차를 계산하기 위해서, 다음 MATLAB프로그램 RoundoffError101DY.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename RoundoffError101DY.m
% Roundoff Error and Discretization Error of exp(x)
% Programmed by CDY
% ------------------------------------------------------------------------
clear all, close all
x = 1
% Initialization
Nmax = 16; nn = 1:Nmax;
h = 2.^(-nn);
deltaf2 = (exp(x+h)-2*exp(x)+exp(x-h))./h.^2;
df2 = exp(x)*ones(1,Nmax);
DiscretizationError = 1/12*exp(x).*h.^2;
TotalError = abs(deltaf2-df2)
format short g
OutP = [ h ; deltaf2; df2; TotalError ]'
% Minimum Total Error
c1 = 1/12*exp(1)      % c1 = |f^(4)(x)|/12
c2 = 4*exp(1)         % c2 = 4*|f(x)|
eta = eps(1)/2
hstar = (c2*eta/c1)^(1/4)
log10hstar = log10(hstar)
MinTotalError = 2*sqrt(c1*c2*eta)
log10MinTotalError = log10(MinTotalError)
% Plotting
loglog(h,TotalError,'r-o',h,DiscretizationError,'k-','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold','xDir','reverse')
legend('\bf Total Error','Discretization Error',3)
xlabel('h')
ylabel('Errors')
saveas(gcf,'RoundoffError101DY.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 RoundoffError101DY.m을 실행하면, 그림 1.6.2가 그려진다. 그림 1.6.2에서 알 수 있듯이, 총오차는 $h^* = \left[\frac{c_2\,\eta}{c_1}\right]^{1/4} = 10^{-3.568} = 0.00027$ 에서 최소값 $2\sqrt{c_1\, c_2\, \eta} = 3.307 \cdot 10^{-8}$ 을 갖는다. 따라서, 만약 부등식 $h < 0.00027$ 이 성립하면, 절단오차가 크다. 그렇지 않으면, 반올림오차가 크다. ■

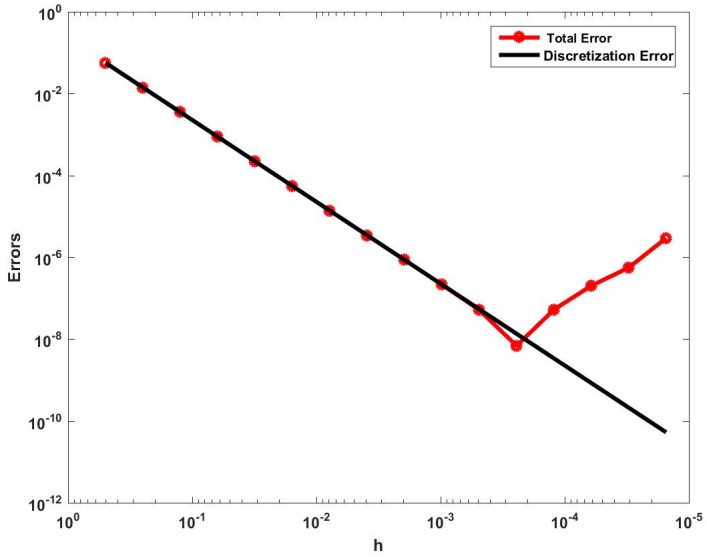


그림 1.6.2. 총오차의 최소값

예제 1.6.4 Python을 사용해서 예제 1.6.3을 다시 다루기 위해서, 다음 Python프로그램 RoundoffError101DY.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
# import math

x = 1.0
# Initialization
Nmax = 16;
nn = np.linspace(1, Nmax, num=Nmax)
h = 0.5**nn; h = h[:,None]
deltaf2 = (np.exp(x+h)-2*np.exp(x)+np.exp(x-h))/h**2;
df2 = np.exp(x)*np.ones((Nmax,1));
DiscretizationError = 1/12*np.exp(x)*h**2;
TotalError = abs(deltaf2-df2);
OutP = np.concatenate((h, deltaf2, df2, TotalError), axis=1)
# Minimum Total Error
c1 = 1/12*np.exp(1)       # c1 = |f**(4)(x)|/12
c2 = 4*np.exp(1)          # c2 = 4*|f(x)|
epss = 7/3 - 4/3 -1       # epss = Machine Eps
eta = epss/2
hstar = (c2*eta/c1)**(1/4)
log10hstar = np.log10(hstar)
MinTotalError = 2*np.sqrt(c1*c2*eta)
log10MinTotalError = np.log10(MinTotalError)

# Plotting
fig = plt.figure()
plt.loglog(h,TotalError,'r-o',label='Total Error',lw=2)
plt.loglog(h,DiscretizationError,'k-',label='Discretization Error',lw=2)
plt.gca().invert_xaxis()
```

```
plt.legend(loc='upper right', numpoints=1)
plt.xlabel('h')
plt.ylabel('Errors')
plt.show()
fig.savefig('RoundoffError101DYpy.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 1.6.3의 결과와 같다. ■

제 2 장

직교함수

제2.1절 직교함수계

연속함수들의 집합 C^0는 단항함수들 $\{x^n | n = 0, 1, 2, \cdots\}$에 의해서 생성된다. 따라서, 함수 공간 C^0의 기저(basis)로 멱함수들을 사용하는 것은 자연스러운 일이다.

2.1.1 직교기저

이 소절에서는 멱함수들로 이루어진 직교기저(orthogonal basis)에 대해서 살펴보자.

정의 2.1.1

다음과 같이 구간 $[a, b]$에서 제곱가적분인 함수들 f와 g를 살펴보자.

$$\int_a^b |f(x)|^2 dx < \infty, \quad \int_a^b |g(x)|^2 dx < \infty$$

이 함수들 f와 g의 내적(inner product)과 함수 f의 노름(norm)을 각각 다음과 같이 정의하자.

$$< f, g > \doteq \int_a^b f(x)g(x)dx, \quad \|f\| \doteq \sqrt{< f, f >}$$

또한, 함수들 f와 g의 거리(distance)를 다음과 같이 정의하자.

$$d(f, g) \doteq \|f - g\| \doteq \left[\int_a^b |f(x) - g(x)|^2 dx\right]^{1/2}$$

다음 식들을 만족하는 함수 w를 가중함수(weighting function)라 한다.

$$w(x) \geq 0, \quad (x \in [a,b]), \qquad \int_a^b w(x)dx < \infty$$

주어진 가중함수 w에 대해서, 다음과 같이 내적, 노름, 그리고 거리를 정의한다.

$$< f, g >_w \doteq \int_a^b w(x)f(x)g(x)dx$$
$$\|f\|_w \doteq \sqrt{< f, f >_w}$$
$$d_w(f,g) \doteq \|f-g\|_w = \left[\int_a^b w(x)|f(x)-g(x)|^2 dx\right]^{1/2}$$

정의 2.1.2

만약 다음 식들이 성립하면, 멱함수열 $\{\phi_n \,|\, n = 0,1,2,\cdots\}$이 가중함수 w에 대해서 직교라고(orthogonal) 한다.

$$< \phi_m, \phi_n >_w = 0, \quad (m \neq n)$$

또한, 함수 ϕ_n를 직교함수라 한다. 만약 다음 식들이 성립하면, 멱함수열 $\{\phi_n \mid n = 0,1,2,\cdots\}$이 가중함수 w에 대해서 정규직교라고(orthonormal) 한다.

$$< \phi_m, \phi_n >_w = \delta_{m,n}$$

여기서 $\delta_{m,n}$은 Kronecker의 델타함수이다.

함수의 집합 $\{f_j \mid j = 0,1,\cdots\}$에 속하는 임의의 두 함수들이 가중함수 w에 대해서 직교하면, $\{f_j\}$는 가중함수 w에 대해 직교계(orthogonal system)를 이룬다고 한다. 또한, 직교계 $\{f_j\}$의 각 함수가 가중함수 w에 대해서 정규화되어 있으면, 이 $\{f_j\}$를 가중함수 w에 대해 정규직교함수계 또는 정규직교계(orthonormal system)을 이룬다고 한다. 정규직교함수계에서 함수의 근사를 생각해보자. 직교함수에 대한 자세한 내용은 Sansone [52]과 Judd [31]를 참조하라. Gram-Schmidt정리를 사용해서 직교함수들 $\{\phi_n \mid n = 0,1,2\cdots\}$ 사이의 관계식을 보여주는 다음 명제를 증명할 수 있다. 이에 대한 자세한 내용은 Arfken & Weber

& Harris [9]를 참조하라.

명제 2.1.1

다음 함수들을 정의하자.

(a) $\phi_0(x) \doteq 1$

(b) $\phi_1(x) \doteq x - \frac{<x,1>_w}{<1,1>_w}$

(c) $\phi_{n+1}(x) \doteq [x - \alpha_n]\phi_n(x) - \beta_n\phi_{n-1}(x), \quad (n = 1, 2, \cdots)$

여기서 α_n과 β_n은 각각 다음과 같다.

$$\alpha_n \doteq \frac{< x\phi_n, \phi_n >_w}{< \phi_n, \phi_n >_w}, \quad \beta_n \doteq \frac{< \phi_n, \phi_n >_w}{< \phi_{n-1}, \phi_{n-1} >_w}$$

이 멱함수열 $\{\phi_n \mid n = 0, 1, 2, \cdots\}$은 가중함수 w에 대해서 직교한다. 또한, 각 멱함수 $\phi(x)$의 최고항의 계수는 1이다.

각 직교함수열에 대해서 명제 2.1.1에 해당하는 점화식을 구할 수 있다. 따라서, 각 직교함수의 정의로부터 계산을 하는 것보다는 이 점화식을 이용하는 것이 계산이 빠르고 안정적이다. 다양한 가중함수들에 대한 직교함수열들이 존재한다. 이 장에서는 이중에서 널리 사용되는 Legendre함수, Chebyshev함수, Laguerre함수 그리고 Hermite함수에 대해서 간단히 설명하고자 한다. 이 함수들에 대한 자세한 내용은 Abramowitz & Stegun [7, 제22장]을 참조하라.

2.1.2 Legendre함수

제n차 Legendre함수 $P_n(x)$를 다음과 같이 정의한다.

$$P_n(x) \doteq \frac{1}{2^n n!}\frac{d^n}{dx^n}[x^2 - 1]^n \tag{2.1.1}$$

명제 2.1.2

Legendre함수열 $\{P_n\}$은 다음 식들을 만족한다.

(a) $\int_{-1}^{1} x^k P_n(x)dx = 0, \quad (k = 0, 1, \cdots, n-1)$

(b) $\int_{-1}^{1} x^n P_n(x)dx = \frac{2^{n+1}n!n!}{(2n+1)!}$

(c) $\int_{-1}^{1} P_m(x)P_n(x)dx = \frac{2}{2m+1}\delta_{m,n}$

증명. **(a)** 부분적분을 k 번 적용하면, 다음 식들이 성립함을 알 수 있다.

$$\int_{-1}^{1} x^k \frac{d^n}{dx^n}[x^2-1]^n dx = [-1]^k k! \int_{-1}^{1} \frac{d^{n-k}}{dx^{n-k}}[x^2-1]^n dx = 0 \tag{1}$$

(b) 부분적분을 n 번 적용하면, 다음 식들이 성립함을 알 수 있다.

$$\begin{aligned}&\int_{-1}^{1} x^n \frac{d^n}{dx^n}[x^2-1]^n dx = [-1]^n n! \int_{-1}^{1} [x^2-1]^n dx \\ &= 2n! \int_{0}^{1} [1-x^2]^n dx = 2n! \int_{0}^{\pi/2} \cos^{2n+1} z dz = \frac{2^{n+1}n!n!}{(2n+1)!}\end{aligned} \tag{2}$$

(c) 함수 $P_m(x)$ 를 다음과 같이 쓸 수 있다.

$$P_m(x) \doteq \sum_{k=0}^{m} p_{m,k} x^k \tag{3}$$

식 (2.1.1) 에서 알 수 있듯이, $p_{m,m}$ 은 다음과 같다.

$$p_{m,m} = \frac{1}{2^m m!}\frac{(2m)!}{m!} \tag{4}$$

만약 $m < n$ 이면, 다음 식들이 성립한다.

$$\int_{-1}^{1} P_m(x)P_n(x)dx = \sum_{k=0}^{m} p_{m,k} \int_{-1}^{1} x^k P_n(x)dx = 0 \tag{5}$$

여기서 두 번째 등호는 성질 (a) 에 의해서 성립한다. 만약 $m = n$ 이면, 다음 식들이 성립한다.

$$\begin{aligned}&\int_{-1}^{1} P_n(x)P_n(x)dx = \sum_{k=0}^{n} p_{n,k} \int_{-1}^{1} x^k P_n(x)dx \\ &= p_{n,n} \int_{-1}^{1} x^n P_n(x)dx = \frac{2}{2n+1}\end{aligned} \tag{6}$$

여기서 두 번째 등호는 성질 (a) 에 의해서, 그리고 세 번째 등호는 식 (4) 와 성질 (b) 에 의해서 성립한다. ■

명제 2.1.2의 성질 (c)에서 알 수 있듯이, Legendre함수열 $\{P_n\}$은 가중함수가 다음과 같은 직교함수열이다.

$$w(x) = 1, \quad (x \in [-1, 1]) \tag{2.1.2}$$

명제 2.1.2를 사용해서, 다음 식들을 유도할 수 있다.

$$P_0(x) = 1, \quad P_1(x) = x, \quad P_2(x) = \frac{3}{2}\left[x^2 - \frac{1}{3}\right] \tag{2.1.3}$$

$$P_3(x) = \frac{5}{2}\left[x^3 - \frac{3}{5}x\right], \quad P_4(x) = \frac{35}{8}\left[x^4 - \frac{6}{7}x^2 + \frac{3}{35}\right] \tag{2.1.4}$$

$$P_5(x) = \frac{63}{8}\left[x^5 - \frac{10}{9}x^3 + \frac{5}{21}\right] \tag{2.1.5}$$

$$P_6(x) = \frac{231}{16}\left[x^6 - \frac{15}{11}x^4 + \frac{5}{11}x^2 - \frac{5}{231}\right] \tag{2.1.6}$$

명제 2.1.3

Legendre함수열 $\{P_n\}$은 다음 점화식을 만족한다.

$$P_{n+1}(x) = \frac{2n+1}{n+1}xP_n(x) - \frac{n}{n+1}P_{n-1}(x), \quad (n = 1, 2, \cdots)$$

증명. 멱함수 $xP_n(x)$는 $[n+1]$차 멱함수이다. 따라서, 다음과 같이 표기할 수 있다.

$$xP_n(x) = \sum_{k=0}^{n+1} c_k P_k(x) \tag{1}$$

각 $j(= 0, 1, \cdots, n-2)$에 대해서 다음 식들이 성립한다.

$$\begin{aligned} \frac{2}{2j+1}c_j &= c_j \int_{-1}^{1} P_j^2(x)dx = \sum_{k=0}^{n+1} c_k \int_{-1}^{1} P_k(x)P_j(x)dx \\ &= \int_{-1}^{1} xP_n(x)P_j(x)dx = \int_{-1}^{1} P_n(x)[xP_j(x)]dx = 0 \end{aligned} \tag{2}$$

여기서 첫 번째 등호와 두 번째 등호는 명제 2.1.2의 성질 (c)에 의해서, 세 번째 등호는 식 (1)에 의해서, 그리고 $xP_j(x)$가 최대 $[n-1]$차원이므로 명제 2.1.2에 의해서 다섯 번째 등호가

성립한다. 따라서, 다음 식이 성립한다.

$$xP_n(x) = c_{n+1}P_{n+1}(x) + c_nP_n(x) + c_{n-1}P_{n-1}(x) \tag{3}$$

명제 2.1.2의 식 (c)에서 알 수 있듯이, 식 (3)의 좌변의 멱함수 $xP_n(x)$의 x^{n+1}의 계수는 $p_{n,n}$이다. 또한, 우변의 x^{n+1}의 계수는 $c_{n+1}p_{n+1,n+1}$이다. 따라서, 다음 식들이 성립함을 알 수 있다.

$$\frac{1}{2^n n!}\frac{(2n)!}{n!} = p_{n,n} = c_{n+1}p_{n+1,n+1} = c_{n+1}\frac{1}{2^{n+1}(n+1)!}\frac{(2n+2)!}{(n+1)!} \tag{4}$$

여기서 첫 번째 등호와 세 번째 등호는 식 (2.1.1)에 의해서 성립한다. 식 (4)에서 알 수 있듯이, 다음 식이 성립한다.

$$c_{n+1} = \frac{n+1}{2n+1} \tag{5}$$

식 (2.1.1)에서 알 수 있듯이, Legendre함수는 짝수항들로만 이루어지거나 아니면 홀수항들로만 이루어진다. 따라서, 다음 식이 성립한다.

$$c_n = 0 \tag{6}$$

식 (5)와 식 (6)을 식 (3)에 대입하면, 다음 식이 성립함을 알 수 있다.

$$x^nP_n(x) = \frac{n+1}{2n+1}x^{n-1}P_{n+1}(x) + c_{n-1}x^{n-1}P_{n-1}(x) \tag{7}$$

식 (7)에서 알 수 있듯이, 다음 식이 성립한다.

$$\int_{-1}^{1} x^nP_n(x)dx = \frac{n+1}{2n+1}\int_{-1}^{1} x^{n-1}P_{n+1}(x)dx + c_{n-1}\int_{-1}^{1} x^{n-1}P_{n-1}(x)dx \tag{8}$$

명제 2.1.2를 식 (8)에 대입하면, 다음 식이 성립함을 알 수 있다.

$$\frac{2^{n+1}n!n!}{(2n+1)!} = c_{n-1}\frac{2^n(n-1)!(n-1)!}{(2n-1)!} \tag{9}$$

즉, 다음 식이 성립한다.

$$c_{n-1} = \frac{n}{2n+1} \tag{10}$$

■

명제 2.1.4

Legendre함수 $P_n(x)$는 구간 $(-1,1)$에서 n개 실근들을 갖는다.

증명. 함수 $f(x)=[x^2-1]^n$은 $x=+1$과 $x=-1$에서 각각 중복도 n인 근을 갖는다. 함수 $f'(x)$도 $x=+1$과 $x=-1$을 근으로 가지므로, Rolle정리에 함수 $f''(x)$는 구간 $(-1,1)$에서 실근 1개를 갖는다. 이 근을 $x_{2,1}$이라 하자. Rolle정리에서 알 수 있듯이, 함수 $f''(x)$는 구간 $(-1,x_{2,1})$에서 실근 1개 그리고 구간 $(x_{2,1},1)$에서 실근 1개를 갖는다. 함수 $f^{(n-1)}(x)$가 구간 $(-1,1)$에서 서로 다른 실근들 $x_{n-1,1},x_{n-1,2},\cdots,x_{n-1,n-2}$을 갖는다고 가정하자. 이 실근들이 다음 식들을 만족한다고 하자.

$$-1=x_{n-1,0}<x_{n-1,1}<x_{n-1,2}<\cdots<x_{n-1,n-2}<x_{n-1,n-1}=1 \tag{1}$$

함수 $f^{(n)}(x)$도 $x=+1$과 $x=-1$을 근으로 가지므로, Rolle정리에 의해 함수 $f^{(n)}(x)$는 각 $k(\in\{1,2,\cdots,n\})$에 대해서 구간 $(x_{n-1,k-1},x_{n-1,k})$에서 실근 1개를 갖는다. 즉, 함수 $f^{(n)}(x)$는 구간 $(-1,1)$에서 실근 n개를 갖는다. Legendre함수 $P_n(x)$는 n차 멱함수이므로, 방정식 $P_n(x)=0$는 복소수평면에서 근 n를 갖는다. 따라서, 방정식 $P_n(x)=0$의 각 근은 구간 $(-1,1)$에 속하는 실근이다. ■

예제 2.1.1 Legendre함수를 구하기 위해서, 다음 MATLAB프로그램 MakeLegendrePoly101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename MakeLegendrePoly101.m
% Make Legendre Polynomials
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
x = -1:0.03:1;
p0 = (x-x)+1;
p1 = x;
p2 = 1/2*( 3*x.*p1 - 1*p0 );
p3 = 1/3*( 5*x.*p2 - 2*p1 );
p4 = 1/4*( 7*x.*p3 - 3*p2 );
plot(x,p0,'k-.',x,p1,'r-',x,p2,'g:',x,p3,'b-.', ...
     x,p4,'m--','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
```

```
        '\it n=4','location','SouthEast')
xlabel('x','fontsize',12,'fontangle','italic')
ylabel('\it P_n','fontsize',12,'fontangle','italic','rotation',0)
axis([-1 1 -1.1 1.1])
saveas(gcf,'MakeLegendrePoly101','jpg')
save('MakeLegendrePoly101.txt','p0','p1','p2','p3','p4')
% End of program
% --------------------------------------------------------------------
```

이 MATLAB프로그램 MakeLegendrePoly101.m을 실행하면, Legendre함수들 $\{P_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.1을 출력한다. ■

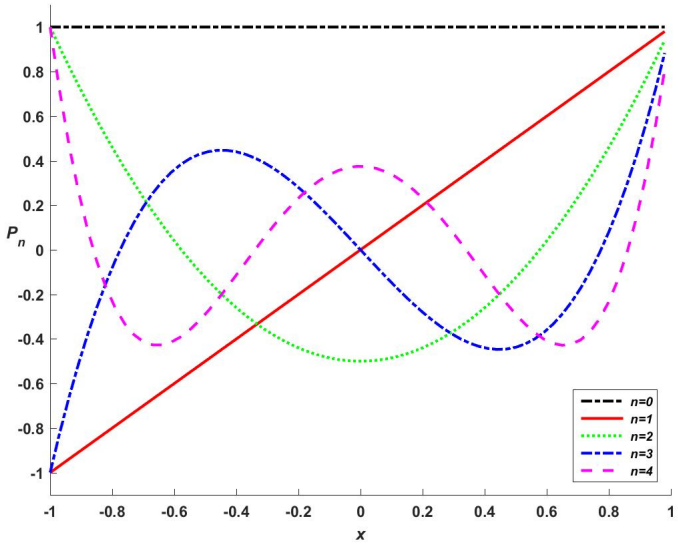


그림 2.1.1. Legendre함수들 I

예제 2.1.2 Python을 사용해서 예제 2.1.1을 다시 다루기 위해서, 다음 Python프로그램 MakeLegendrePoly101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1,1, num=201)
p0 = (x-x)+1.0;
p1 = x;
p2 = 1/2*( 3*x*p1 - 1*p0 );
p3 = 1/3*( 5*x*p2 - 2*p1 );
p4 = 1/4*( 7*x*p3 - 3*p2 );

# Plotting
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(x,p0,'k-.', lw=2, label='n=0')
plt.plot(x,p1,'r-',  lw=2, label='n=1')
plt.plot(x,p2,'g:',  lw=2, label='n=2')
plt.plot(x,p3,'b-.', lw=2, label='n=3')
plt.plot(x,p4,'m--', lw=2, label='n=4')
plt.xlabel('x'); plt.ylabel('P_n (x)')
plt.axis([-1, 1, -1.1, 1.1])
plt.show()
fig.savefig('MakeLegendrePoly101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.1.1의 결과와 같다. ■

예제 2.1.3 MATLAB함수 legendre.m을 사용해서, Legendre함수를 구할 수 있다. 이 MATLAB함수의 사용법은 다음과 같다.

```
>> lp = legendre(m,x)
```

이 MATLAB명령문을 실행하면, 다음 함수들을 출력한다.

$$P_n^m(x) \doteq [-1]^m[1-x^2]^{m/2}\frac{d^m}{dx^m}P_n(x), \quad (m=0,1,\cdots,n) \tag{1}$$

따라서, 다음 식이 성립한다.

$$P_n(x) = P_n^0(x) \tag{2}$$

즉, 출력변수 lp의 첫 번째 행이 Legendre함수이다. MATLAB함수 legendre.m를 사용해서 Legendre함수를 구하기 위해서, 다음 MATLAB프로그램 USElegendrePoly101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename USElegendrePoly101.m
% Legendre Polynomials by MATLAB function legendre.m
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
x = -1:0.03:1;
lp00 = legendre(0,x); lp0 = lp00(1,:);
lp11 = legendre(1,x); lp1 = lp11(1,:);
lp22 = legendre(2,x); lp2 = lp22(1,:);
lp33 = legendre(3,x); lp3 = lp33(1,:);
lp44 = legendre(4,x); lp4 = lp44(1,:);
plot(x,lp0,'k-.',x,lp1,'r-',x,lp2,'g:',x,lp3,'b-.', ...
     x,lp4,'m--','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
box off
```

```
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','SouthEast')
xlabel('x','fontsize',12,'fontangle','italic','rotation',0)
ylabel('L_n','fontsize',12,'fontangle','italic','rotation',0)
axis([-1 1 -1.1 1.1])
saveas(gcf,'USElegendrePoly101','jpg')
save('USElegendrePoly101.txt','lp00','lp11','lp22','lp33','lp44')
% End of lprogram
% ------------------------------------------------------------------
```

이 MATLAB프로그램 USElegendre.m을 실행하면, Legendre함수들 $\{P_n(x)|n=0,1,2,3,4\}$를 계산하고, 또한 그림 2.1.2를 출력한다. 그림 2.1.2은 그림 2.1.1과 동일하다. ■

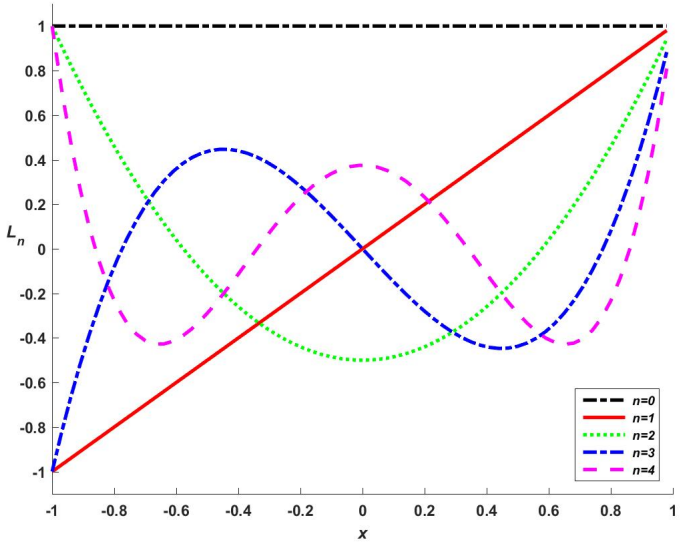


그림 2.1.2. Legendre함수 II

예제 2.1.4 MATLAB함수 legendreP.m을 사용해서, Legendre함수를 구할 수 있다. 이 MATLAB함수의 사용법은 다음과 같다.

```
>> lp = legendreP(m,x)
```

MATLAB함수 legendreP.m은 수치적으로 뿐아니라 심볼릭으로도 사용할 수 있다. MATLAB함수 legendreP.m를 사용해서 Legendre함수를 구하기 위해서, 다음 MATLAB프로그램 USElegendreP101.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename USElegendreP101.m
% legendreP Polynomials by MATLAB function legendreP.m
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
```

```
syms x
hold all
for n=0:4
    ezplot(legendreP(n,x))
end
axis([-1 1 -1.1 1.1])
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','best')
xlabel('x','fontsize',12,'fontangle','italic','rotation',0)
ylabel('P_n (x)','fontsize',12,'fontangle','italic')
title('\bf Legendre Polynomials')
hold off
saveas(gcf,'USElegendreP101','jpg')
% End of lprogram
% ------------------------------------------------------------------
```

이 MATLAB프로그램 USElegendreP.m을 실행하면, Legendre함수들 $\{P_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.3을 출력한다. 그림 2.1.3은 그림 2.1.1과 동일하다. ■

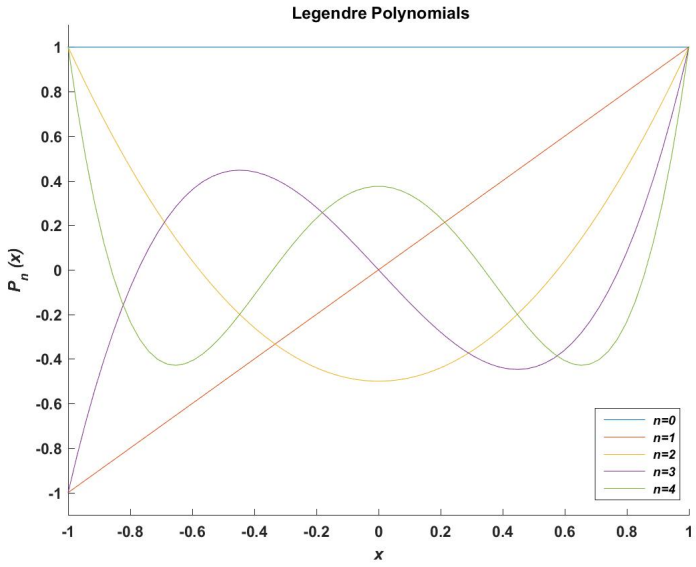


그림 2.1.3. Legendre함수 III

예제 2.1.5 Python을 사용해서 예제 2.1.1을 다시 다루기 위해서, 다음 Python프로그램 USElegendrePoly101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
```

```
8
9  x = np.linspace(-1,1, num=201)
10 lp0 = np.polynomial.legendre.Legendre.basis(0)(x)
11 lp1 = np.polynomial.legendre.Legendre.basis(1)(x)
12 lp2 = np.polynomial.legendre.Legendre.basis(2)(x)
13 lp3 = np.polynomial.legendre.Legendre.basis(3)(x)
14 lp4 = np.polynomial.legendre.Legendre.basis(4)(x)
15
16 # Plotting
17 fig = plt.figure()
18 ax = fig.add_subplot(111)
19 plt.plot(x,lp0,'k-.', lw=2, label='n=0')
20 plt.plot(x,lp1,'r-',  lw=2, label='n=1')
21 plt.plot(x,lp2,'g:',  lw=2, label='n=2')
22 plt.plot(x,lp3,'b-.', lw=2, label='n=3')
23 plt.plot(x,lp4,'m--', lw=2, label='n=4')
24 plt.xlabel('x'); plt.ylabel('P_n (x)')
25 plt.legend(loc='lower right', numpoints = 1)
26 plt.axis([-1, 1, -1.1, 1.1])
27 plt.show()
28 fig.savefig('USElegendrePoly101Py.png')
29
30 # End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.1.1의 결과와 같다. ■

2.1.3 Chebyshev함수

제n차 Chebyshev함수 $T_n(x)$를 다음과 같이 정의한다.

$$T_n(x) \doteq \cos(n \cos^{-1} x) \tag{2.1.7}$$

명제 2.1.5

Chebyshev함수열 $\{T_n\}$은 다음 식들을 만족한다.

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} T_m(x) T_n(x)\, dx = \begin{cases} 0, & (m \neq n) \\ \frac{\pi}{2}, & (m = n \neq 0) \\ \pi, & (m = n = 0) \end{cases}$$

증명. 다음 식들이 성립한다.

$$\begin{aligned}
&\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} T_m(x) T_n(x) dx \\
&= \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} \cos(m \cos^{-1} x) \cos(n \cos^{-1} x)\, dx \\
&= \int_{0}^{\pi} \cos mz \cos nz\, dz
\end{aligned} \tag{1}$$

여기서 두 번째 등호는 변수변환 $x = \cos z$에 의해서 성립한다. 그리고 다음 식이 성립한다.

$$\cos mz \cos nz = \frac{1}{2} \{\cos[m+n]x + \cos[m-n]x\} \tag{2}$$

식 (2)를 사용해서 식 (1)의 우변을 쉽게 적분할 수 있다. ■

명제 2.1.5에서 알 수 있듯이, Chebyshev함수열 $\{T_n\}$은 가중함수가 다음과 같은 직교함수열이다.

$$w(x) = \frac{1}{\sqrt{1-x^2}}, \quad (x \in [-1, 1]) \tag{2.1.8}$$

명제 2.1.5에서 알 수 있듯이, 다음 식들이 성립한다.

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_2(x) = 2x^2 - 1 \tag{2.1.9}$$

$$T_3(x) = 4x^3 - 3x, \quad T_4(x) = 8x^4 - 8x^2 + 1 \tag{2.1.10}$$

$$T_5(x) = 16x^5 - 20x^3 + 5x, \quad T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1 \tag{2.1.11}$$

$$T_7(x) = 64x^7 - 112x^5 + 56x^3 - 7x \tag{2.1.12}$$

명제 2.1.6

Chebyshev함수열 $\{T_n\}$은 다음 식들을 만족한다.

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x), \quad (n = 1, 2, \cdots)$$

증명.

$$\cos[n+1]z + \cos[n-1]z = 2 \cos z \cos nz \tag{1}$$

식 (1)에 변수변환 $x = \cos z$를 적용하면, 증명이 끝난다. ■

명제 2.1.7

Chebyshev함수 $T_n(x)$는 구간 $(-1, 1)$에서 n개 실근들을 갖는다.

증명. 변수변환 $x = \cos\theta$를 적용하면, 다음 식이 성립한다.

$$T_n(\cos\theta) = \cos n\theta \tag{1}$$

일반성을 잃지 않고, θ는 구간 $[0, \pi]$에 속한다고 할 수 있다. 다음 방정식을 살펴보자.

$$T_n(\cos\theta) = 0, \quad (0 \le \theta \le \pi) \tag{2}$$

다음 식이 성립함은 명백하다.

$$\theta = \frac{\pi}{2n}[2k+1], \quad (k = 0, 1, \cdots, n-1) \tag{3}$$

즉, 방정식 $T_n(x) = 0$ 의 근들은 다음과 같다.

$$x_k \doteq \cos\left(\frac{\pi}{2n}[2k+1]\right), \quad (k = 0, 1, \cdots, n-1) \tag{4}$$

즉, Chebyshev함수 $T_n(x)$는 구간 $(-1, 1)$에서 n개 실근들을 갖는다. ■

예제 2.1.6 Chebyshev함수를 구하기 위해서, 다음 MATLAB프로그램 MakeChebyshevPoly101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename MakeChebyshevPoly101.m
% Make Chebyshev Polynomials
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
x = -1:0.03:1;
T0 = x-x+1;
T1 = x;
T2 = 2*x.*T1 - T0;
T3 = 2*x.*T2 - T1;
T4 = 2*x.*T3 - T2;
plot(x,T0,'k-.',x,T1,'r-',x,T2,'g:',x,T3,'b-.', ...
```

```
     x,T4,'m--','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','Best')
xlabel('x','fontsize',12,'fontangle','italic')
ylabel('T_n','fontsize',12,'fontangle','italic','rotation',0)
axis([-1 1 -1.1 1.1])
saveas(gcf,'MakeChebyshevPoly101','jpg')
save('MakeChebyshevPoly101.txt','T0','T1','T2','T3','T4')
% End of program
% --------------------------------------------------------------------
```

이 MATLAB프로그램 MakeChebyshevPoly101.m을 실행하면, Chebyshev함수들 $\{P_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.4를 출력한다. 그림 2.1.4에서 알 수 있듯이, Chebyshev 함수열 $\{T_n\}$은 등진동성 (equal-ripple property)를 갖는다. ■

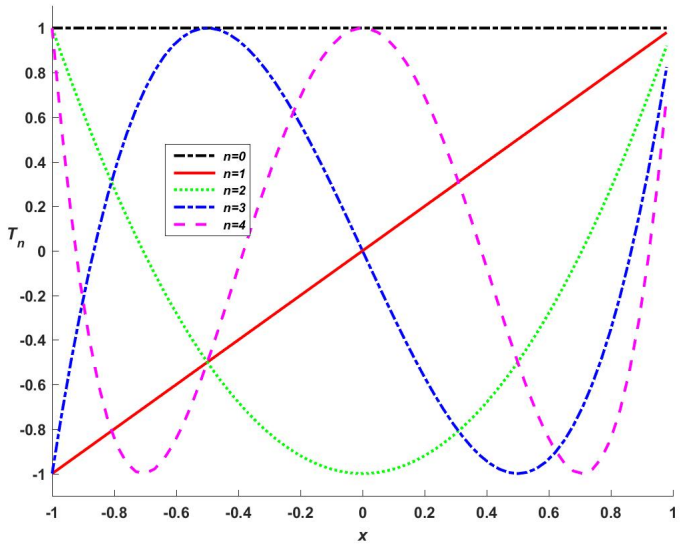


그림 2.1.4. Chebyshev함수 I

예제 2.1.7 Python을 사용해서 예제 2.1.6을 다시 다루기 위해서, 다음 Python프로그램 MakeChebyshevPoly101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1,1, num=201)
```

```
T0 = x-x+1.0;
T1 = x;
T2 = 2*x*T1 - T0;
T3 = 2*x*T2 - T1;
T4 = 2*x*T3 - T2;

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(x,T0,'k-.', lw=2, label='n=0')
plt.plot(x,T1,'r-',  lw=2, label='n=1')
plt.plot(x,T2,'g:',  lw=2, label='n=2')
plt.plot(x,T3,'b-.', lw=2, label='n=3')
plt.plot(x,T4,'m--', lw=2, label='n=4')
plt.xlabel('x'); plt.ylabel('T_n (x)')
plt.axis([-1, 1, -1.1, 1.1])
plt.show()
fig.savefig('MakeChebyshevPoly101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.1.6의 결과와 같다. ■

예제 2.1.8 MATLAB함수 chebyshevT.m을 사용해서, Chebyshev함수를 구할 수 있다. 이 MATLAB함수의 사용법은 다음과 같다.

```
>> cp = chebyshevT(m,x)
```

MATLAB함수 chebyshevPoly101.m를 사용해서 Chebyshev함수를 구하기 위해서, 다음 MATLAB프로그램 USEchebyshevPoly101.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename USEchebyshevPoly101.m
% chebyshevT Polynomials by MATLAB function chebyshevT.m
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
x = -1:0.03:1;
tp00 = chebyshevT(0,x); tp0 = tp00(1,:);
tp11 = chebyshevT(1,x); tp1 = tp11(1,:);
tp22 = chebyshevT(2,x); tp2 = tp22(1,:);
tp33 = chebyshevT(3,x); tp3 = tp33(1,:);
tp44 = chebyshevT(4,x); tp4 = tp44(1,:);
plot(x,tp0,'k-.',x,tp1,'r-',x,tp2,'g:',x,tp3,'b-.', ...
     x,tp4,'m--','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','Best')
xlabel('x','fontsize',12,'fontangle','italic')
ylabel('T_n','fontsize',12,'fontangle','italic','rotation',0)
axis([-1 1 -1.1 1.1])
saveas(gcf,'USEchebyshevPoly101','jpg')
save('USEchebyshevPoly101.txt','tp00','tp11','tp22','tp33','tp44')
% End of tprogram
```

```
% ------------------------------------------------------------
```

이 MATLAB프로그램 USEchebyshevPoly.m을 실행하면, Chebyshev함수들 $\{P_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.5를 출력한다. 그림 2.1.5는 그림 2.1.4와 동일하다. ■

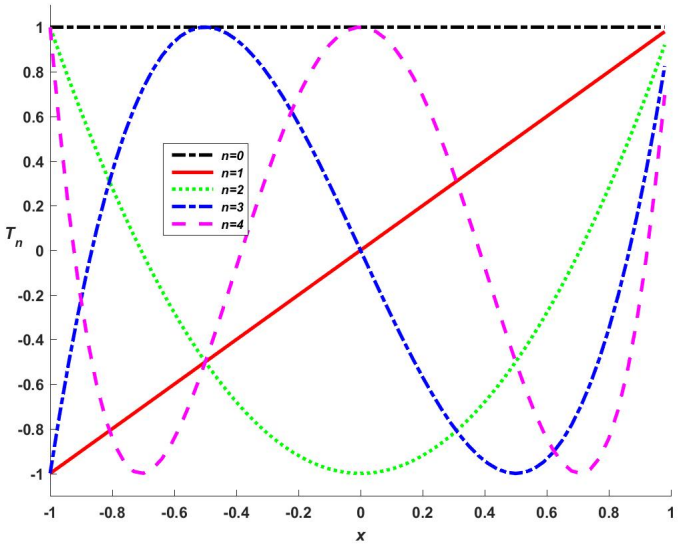


그림 2.1.5. Chebyshev함수 II

예제 2.1.9 MATLAB함수 chebyshevT.m은 수치적으로 뿐아니라 심볼릭으로도 사용할 수 있다. MATLAB함수 chebyshevT.m를 사용해서 심볼릭으로 Chebyshev함수를 구하기 위해서, 다음 MATLAB프로그램 USEchebyshevT101.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename USEchebyshevT101.m
% Chebyshev Polynomials by MATLAB function chebyshevT.m
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
syms x
hold all
for n=0:4
    ezplot(chebyshevT(n,x))
end
axis([-1 1 -1.1 1.1])
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','best')
xlabel('x','fontsize',12,'fontangle','italic')
ylabel('T_n','fontsize',12,'fontangle','italic','rotation',0)
title('\bf Chevyshev Polynomials')
hold off
saveas(gcf,'USEchebyshevT101','jpg')
```

```
% End of lprogram
% --------------------------------------------------------------
```

이 MATLAB 프로그램 USEchebyshevT.m을 실행하면, Chebyshev 함수들 $\{T_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.6을 출력한다. 그림 2.1.6은 그림 2.1.4와 동일하다. ■

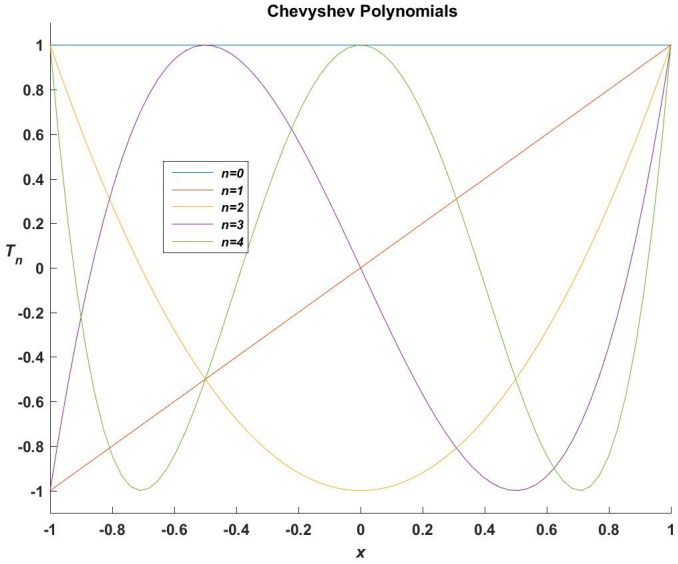


그림 2.1.6. Chebyshev 함수 III

예제 2.1.10 Python을 사용해서 예제 2.1.8을 다시 다루기 위해서, 다음 Python 프로그램 USEchebyshevPoly101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1,1, num=201)
tp0 = np.polynomial.chebyshev.Chebyshev.basis(0)(x)
tp1 = np.polynomial.chebyshev.Chebyshev.basis(1)(x)
tp2 = np.polynomial.chebyshev.Chebyshev.basis(2)(x)
tp3 = np.polynomial.chebyshev.Chebyshev.basis(3)(x)
tp4 = np.polynomial.chebyshev.Chebyshev.basis(4)(x)

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(x,tp0,'k-.', lw=2, label='n=0')
plt.plot(x,tp1,'r-',  lw=2, label='n=1')
plt.plot(x,tp2,'g:',  lw=2, label='n=2')
plt.plot(x,tp3,'b-.', lw=2, label='n=3')
```

```
plt.plot(x,tp4,'m--', lw=2, label='n=4')
plt.legend(loc='upper left', numpoints = 1)
plt.xlabel('x'); plt.ylabel('T_n (x)')
plt.legend(loc='lower right', numpoints = 1)
plt.axis([-1, 1, -1.1, 1.1])
plt.show()
fig.savefig('USEchebyshevPoly101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.1.8의 결과와 같다. ■

2.1.4 Laguerre함수

제n차 Laguerre함수 $L_n(x)$를 다음과 같이 정의한다.

$$L_n(X) \doteq \frac{1}{n!}e^x \frac{d^n}{dx^n}[e^{-x}x^n] \tag{2.1.13}$$

명제 2.1.8

Laguerre함수열 $\{L_n\}$은 다음 식들을 만족한다.

(a) $L_n(x) = \sum_{k=0}^{n}[-1]^k \binom{n}{k}\frac{1}{k!}x^k$

(b) $L_n'(x) = \sum_{k=1}^{n}[-1]^k \binom{n}{k}\frac{1}{(k-1)!}x^{k-1}$

(c) $xL_n'(x) = nL_n(x) - nL_{n-1}(x)$

증명. **(a)** 다음 식이 성립한다.

$$L_n(x) = \frac{1}{n!}e^x \frac{d^{n-1}}{dx^{n-1}}[-e^{-x}x^n + ne^{-x}x^{n-1}] \tag{1}$$

같은 방법으로 미분을 계속하면, 성질 (a)가 성립함을 알 수 있다. 관심있는 독자들은 수학적 귀납법을 적용해서 증명해보라.

(b) 식 (1)에서 성질 (b)가 성립함을 쉽게 알 수 있다.

(c) 다음 식들이 성립한다.

$$\begin{aligned}
&nL_n(x) - nL_{n-1}(x) \\
&= n\sum_{k=0}^{n}[-1]^k\binom{n}{k}\frac{1}{k!}x^k - n\sum_{k=0}^{n-1}[-1]^k\binom{n-1}{k}\frac{1}{k!}x^k \\
&= n\sum_{k=1}^{n-1}[-1]^k\left[\binom{n}{k} - \binom{n-1}{k}\right]\frac{1}{k!}x^k + n[-1]^n\frac{1}{n!}x^n \\
&= n\sum_{k=1}^{n-1}[-1]^k\binom{n-1}{k-1}\frac{1}{k!}x^k + n[-1]^n\frac{1}{n!}x^n \qquad (2)
\end{aligned}$$

여기서 첫 번째 등호는 성질 (a)에 의해서 성립한다. 따라서, 다음 식들이 성립한다.

$$\begin{aligned}
&nL_n(x) - nL_{n-1}(x) \\
&= \sum_{k=1}^{n-1}[-1]^k\binom{n}{k}\frac{1}{(k-1)!}x^k + [-1]^n\frac{1}{(n-1)!}x^n \\
&= x\sum_{k=1}^{n}[-1]^k\binom{n}{k}\frac{1}{(k-1)!}x^{k-1} \qquad (3)
\end{aligned}$$

식 (3)과 성질 (b)에서 알 수 있듯이, 다음 식이 성립한다.

$$xL_n'(x) = nL_n(x) - nL_{n-1}(x) \qquad (4)$$

■

명제 2.1.8에서 알 수 있듯이, 다음 식들이 성립한다.

$$L_0(x) = 1, \quad L_1(x) = -[x-1], \quad L_2(x) = \frac{1}{2!}[x^2 - 4x + 2] \qquad (2.1.14)$$

$$L_3(x) = -\frac{1}{3!}[x^3 - 9x^3 + 18x - 6] \qquad (2.1.15)$$

$$L_4(x) = \frac{1}{4!}[x^4 - 16x^3 + 72x^2 - 96x + 24] \qquad (2.1.16)$$

$$L_5(x) = -\frac{1}{5!}[x^5 - 25x^4 + 200x^3 - 600x^2 + 600x - 120] \qquad (2.1.17)$$

$$L_6(x) = \frac{1}{6!}[x^6 - 36x^5 + 450x^4 - 2400x^3 + 5400x^2 - 4320x + 720] \qquad (2.1.18)$$

명제 2.1.9

Laguerre함수열 $\{L_n\}$은 다음 식들을 만족한다.

(a) $\int_0^\infty e^{-x}x^m L_n(x)dx = \begin{cases} 0, & (m<n) \\ [-1]^n n!, & (m=n) \end{cases}$

(b) $\int_0^\infty e^{-x}L_m(x)L_n(x)dx = \delta_{m,n}$

증명. 임의의 자연수 l과 임의의 실수들 k와 n에 대해서 다음 식들이 성립함을 증명할 수 있다.

$$\lim_{x\to\infty}\left|x^k\frac{d^l}{dx^l}[e^{-x}x^n]\right| = \lim_{x\to\infty}\left|\frac{x^{k+n}+O(x^{k+n-l})}{e^x}\right| = 0 \tag{1}$$

여기서 두 번째 등호는 L'Hospital정리를 사용해서 증명할 수 있다.

(a) 다음 식들이 성립한다.

$$\begin{aligned} n!\int_0^\infty e^{-x}x^m L_n(x)dx &= \int_0^\infty x^m\frac{d^n}{dx^n}[e^{-x}x^n]dx \\ &= x^m\frac{d^{n-1}}{dx^{n-1}}[e^{-x}x^n]\Big|_0^\infty - m\int_0^\infty x^{m-1}\frac{d^{n-1}}{dx^{n-1}}[e^{-x}x^n]dx \\ &= -m\int_0^\infty x^{m-1}\frac{d^{n-1}}{dx^{n-1}}[e^{-x}x^n]dx \end{aligned} \tag{2}$$

여기서 두 번째 등호는 부분적분에 의해서, 그리고 세 번째 등호는 식 (1)에 의해서 성립한다. 식 (2)와 같은 방법을 반복적용하면, 각 $m(\le n)$에 다음 식들이 성립함을 알 수 있다.

$$\begin{aligned} n!\int_0^\infty e^{-x}x^m L_n(x)dx &= \int_0^\infty x^m\frac{d^n}{dx^n}[e^{-x}x^n]dx \\ &= [-1]^m m!\int_0^\infty \frac{d^{n-m}}{dx^{n-m}}[e^{-x}x^n]dx \end{aligned} \tag{3}$$

만약 $m=n$이면, 식 (3)을 다음과 같이 쓸 수 있다.

$$n!\int_0^\infty e^{-x}x^m L_n(x)dx = [-1]^n n!\int_0^\infty e^{-x}x^n dx = [-1]^n n!\Gamma(n+1) \tag{4}$$

즉, 다음 식이 성립한다.

$$\int_0^\infty e^{-x}x^m L_n(x)dx = [-1]^n n! \tag{5}$$

만약 $m < n$ 이면, 식 (3)을 다음과 같이 쓸 수 있다.

$$\begin{aligned} n!\int_0^\infty e^{-x}x^m L_n(x)dx &= [-1]^m m!\int_0^\infty \frac{d^{n-m}}{dx^{n-m}}[e^{-x}x^n]dx \\ &= [-1]^m m!\frac{d^{n-m-1}}{dx^{n-m-1}}[e^{-x}x^n]\Big|_0^\infty = 0 \end{aligned} \tag{6}$$

여기서 세 번째 등호는 식 (1)에 의해서 성립한다.

(b) 일반성을 잃지 않고, 식 $m \le n$ 이 성립한다고 할 수 있다. 다음 식들이 성립한다.

$$\begin{aligned} &\int_0^\infty e^{-x}L_m(x)L_n(x)dx \\ &= \sum_{k=0}^{m}[-1]^k\binom{m}{k}\frac{1}{k!}\int_0^\infty e^{-x}x^k L_n(x)dx \\ &= [-1]^m\frac{1}{m!}\int_0^\infty e^{-x}x^k L_n(x)dx\delta_{m,n} = \delta_{m,n} \end{aligned} \tag{7}$$

여기서 첫 번째 등호는 명제 2.1.8의 성질 (a)에 의해서 두 번째 등호는 성질 (a)에 의해서 성립한다. ■

명제 2.1.9의 성질 (b)에서 알 수 있듯이, Laguerre함수열 $\{L_n\}$은 가중함수가 다음과 같은 직교함수열이다.

$$w(x) = e^{-x}, \quad (x \in [0,\infty)) \tag{2.1.19}$$

명제 2.1.10

Laguerre함수열 $\{L_n\}$은 다음 식들을 만족한다.

$$L_{n+1}(x) = \frac{2n+1-x}{n+1}L_n(x) - \frac{n}{n+1}L_{n-1}(x), \quad (n = 1, 2, \cdots)$$

증명. 멱함수 $xL_n(x)$는 $[n+1]$차 멱함수이므로, 다음과 같이 표기할 수 있다.

$$xL_n(x) = \sum_{k=0}^{n+1} c_k L_k(x) \tag{1}$$

각 $j(=0,1,\cdots,n-2)$에 대해서 다음 식들이 성립한다.

$$\begin{aligned} c_j &= c_j \int_{-1}^{1} e^{-x} L_j^2(x)dx = \sum_{k=0}^{n+1} c_k \int_{-1}^{1} e^{-x} L_k(x) L_j(x)dx \\ &= \int_{-1}^{1} e^{-x} x L_n(x) L_j(x)dx = \int_{-1}^{1} e^{-x} L_n(x)[xL_j(x)]dx = 0 \end{aligned} \tag{2}$$

여기서 첫 번째와 두 번째 등호들은 명제 2.1.9의 성질 (a)에 의해서, 세 번째 등호는 식 (1)에 의해서, 그리고 다섯 번째 등호는 $xL_j(x)$가 최대 $[n-1]$차원이어서 성립한다. 따라서, 다음 식이 성립한다.

$$xL_n(x) = c_{n+1}L_{n+1}(x) + c_nL_n(x) + c_{n-1}L_{n-1}(x) \tag{3}$$

명제 2.1.8의 성질 (a)를 식 (3)에 대입한 다음 양변의 계수들을 비교하면, 증명이 끝난다. ■

명제 2.1.10에서 알 수 있듯이, 각 자연수 n에 대해서 다음 식이 성립한다.

$$L_{n+1}(x) = 2L_n(x) - L_{n-1}(x) - \frac{[1+x]L_n(x) - L_{n-1}(x)}{n+1} \tag{2.1.20}$$

수치적 안정성과 효율성 때문에 명제 2.1.10의 식 대신에 식 (2.1.19)를 사용한다.

명제 2.1.11

Laguerre함수 $L_n(x)$는 구간 $[0,\infty)$에서 n개 실근들을 갖는다.

증명. 명제 2.1.9의 증명의 식 (1)에서 알 수 있듯이, 각 자연수 k에 대해서 다음 식이 성립한다.

$$\lim_{x\to\infty} x^k \frac{d^l}{dx^l}[e^{-x}x^n] = 0 \tag{1}$$

함수 $f(x) = e^{-x}x^n$은 $x=0$과 $x=\infty$를 근으로 갖는다. 식 (1)에서 알 수 있듯이 함수 $f'(x)$도 $x=0$과 $x=\infty$에서 근을 갖으므로, Rolle정리에 의해 구간 $(0,\infty)$에서 실근 1개를 갖음을 알 수 있다. 식 (1)에서 알 수 있듯이 함수 $f''(x)$도 $x=0$과 $x=\infty$을 근으로 갖으므로, Rolle정리에 의해 구간 $(0,\infty)$에서 실근 2개를 갖음을 알 수 있다. 같은 방법을 반복적용하면, 함수 $f^{(n)}(x)$도 $x=0$ 과 $x=\infty$을 근으로 갖고, Rolle정리에 의해 구간 $(0,\infty)$에서 실근 n개를 갖음을 알 수 있다. 즉, Laguerre함수 $L_n(x)$는 구간 $(0,\infty)$에서 n개 실근들을 갖는다. ■

예제 2.1.11 Laguerre함수를 구하기 위해서, 다음 MATLAB프로그램 MakeLaguerrePoly101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename MakeLagurrePoly101.m
% Make Laguerre Polynomials
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
x = 0:0.1:7;
L0 = x-x+1;
L1 = 1-x;
L2 = 1/2*( (3-x).*L1 - 1*L0 );
L3 = 1/3*( (5-x).*L2 - 2*L1 );
L4 = 1/4*( (7-x).*L3 - 3*L2 );
plot(x,L0,'k-.',x,L1,'r-',x,L2,'g:',x,L3,'b-.', ...
     x,L4,'m--','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','Best')
xlabel('x','fontsize',12,'fontangle','italic')
ylabel('\it L_n','fontsize',12,'fontangle','italic','rotation',0)
axis([0 7 -10 15])
saveas(gcf,'MakeLaguerrePoly101','jpg')
save('MakeLaguerrePoly101.txt','L0','L1','L2','L3','L4')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 MakeLaguerrePoly101.m을 실행하면, Laguerre함수들 $\{L_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.7을 출력한다. ■

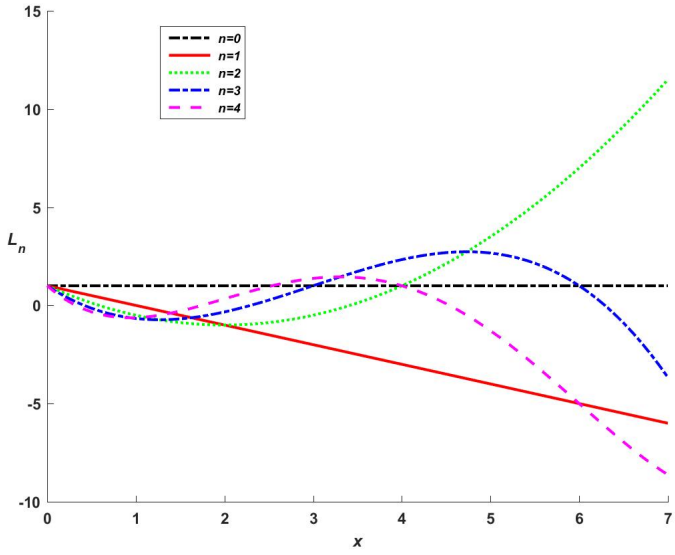


그림 2.1.7. Laguerre함수 I

예제 2.1.12 Python을 사용해서 예제 2.1.11을 다시 다루기 위해서, 다음 Python프로그램 MakeLaguerrePoly101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,7, num=201)
L0 = x-x+1.0;
L1 = 1.0-x;
L2 = 1/2*( (3-x)*L1 - 1*L0 );
L3 = 1/3*( (5-x)*L2 - 2*L1 );
L4 = 1/4*( (7-x)*L3 - 3*L2 );

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(x,L0,'k-.', lw=2, label='n=0')
plt.plot(x,L1,'r-',  lw=2, label='n=1')
plt.plot(x,L2,'g:',  lw=2, label='n=2')
plt.plot(x,L3,'b-.', lw=2, label='n=3')
plt.plot(x,L4,'m--', lw=2, label='n=4')
plt.xlabel('x'); plt.ylabel('L_n (x)')
plt.axis([0, 7, -10, 15])
plt.show()
fig.savefig('MakeLaguerrePoly101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.1.11의 결과와 같다. ■

예제 2.1.13 MATLAB함수 laguerreL.m을 사용해서, Laguerre함수를 구할 수 있다. 이 MATLAB함수의 사용법은 다음과 같다.

```
>> cp = laguerreL(m,x)
```

MATLAB함수 laguerreL.m를 사용해서 Laguerre함수를 구하기 위해서, 다음 MATLAB 프로그램 USElaguerrePoly101.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename USElaguerrePoly101.m
% laguerreL Polynomials by MATLAB function laguerreL.m
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
x = 0:0.03:10;
Lp00 = laguerreL(0,x); Lp0 = Lp00(1,:);
Lp11 = laguerreL(1,x); Lp1 = Lp11(1,:);
Lp22 = laguerreL(2,x); Lp2 = Lp22(1,:);
```

```
Lp33 = laguerreL(3,x); Lp3 = Lp33(1,:);
Lp44 = laguerreL(4,x); Lp4 = Lp44(1,:);
plot(x,Lp0,'k-.',x,Lp1,'r-',x,Lp2,'g:',x,Lp3,'b-.', ...
     x,Lp4,'m--','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','Best')
xlabel('x','fontsize',12,'fontangle','italic')
ylabel('L_n','fontsize',12,'fontangle','italic','rotation',0)
axis([0 7 -10 15])
saveas(gcf,'USElaguerrePoly101','jpg')
save('USElaguerrePoly101.txt','Lp00','Lp11','Lp22','Lp33','Lp44')
% End of Lprogram
% ------------------------------------------------------------------
```

이 MATLAB프로그램 USElaguerrePoly.m을 실행하면, Laguerre함수들 $\{L_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.8을 출력한다. 그림 2.1.8은 그림 2.1.7과 동일하다.

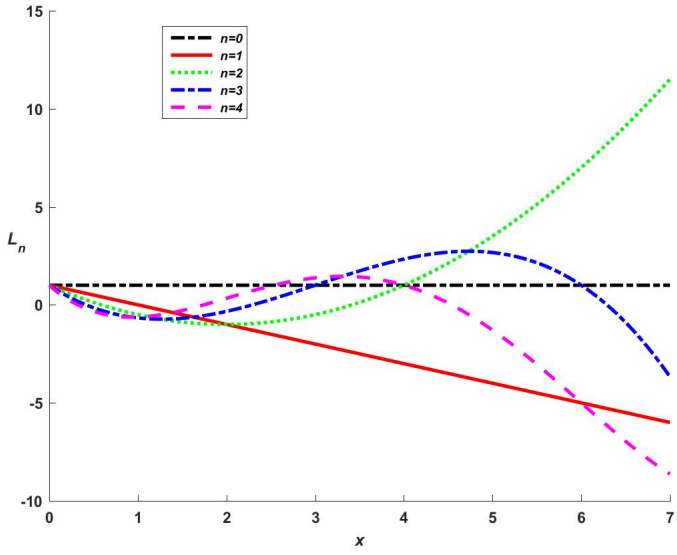


그림 2.1.8. Laguerre함수 II

예제 2.1.14 MATLAB함수 laguerreL.m은 수치적으로 뿐아니라 심볼릭으로도 사용할 수 있다. MATLAB함수 laguerreL.m를 사용해서 심볼릭으로 Laguerre함수를 구하기 위해서, 다음 MATLAB프로그램 USElaguerreL101.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename USElaguerreL101.m
% Laguerre Polynomials by MATLAB function laguerreL.m
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
syms x
```

```
hold all
for n=0:4
    ezplot(laguerreL(n,x))
end
axis([0 7 -10 15])
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','best')
xlabel('x','fontsize',12,'fontangle','italic')
ylabel('L_n','fontsize',12,'fontangle','italic','rotation',0)
title('\bf Chevyshev Polynomials')
hold off
saveas(gcf,'USElaguerreL101','jpg')
% End of lprogram
% --------------------------------------------------------------------
```

이 MATLAB프로그램 USElaguerreL.m을 실행하면, Legendre함수들 $\{L_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.9를 출력한다. 그림 2.1.9는 그림 2.1.7과 동일하다. ■

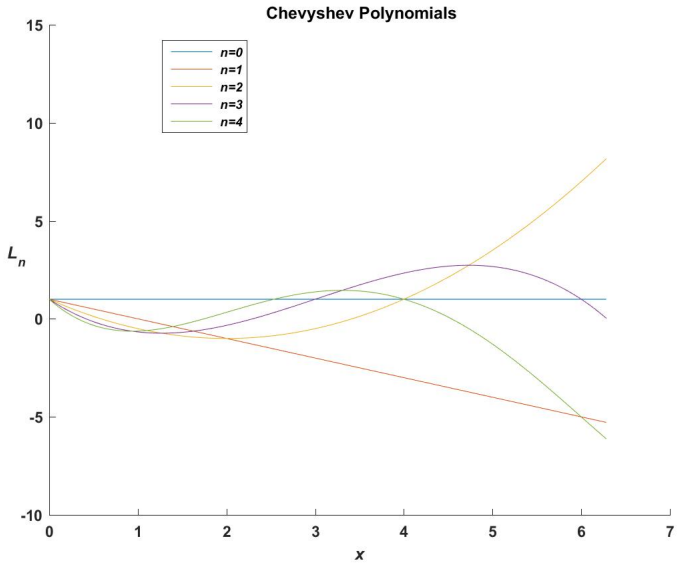


그림 2.1.9. Laguerre함수 III

예제 2.1.15 Python을 사용해서 예제 2.1.13을 다시 다루기 위해서, 다음 Python프로그램 USElaguerrePoly101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0,10, num=201)
```

```
Lp0 = np.polynomial.laguerre.lagval(x,[1])
Lp1 = np.polynomial.laguerre.lagval(x,[0,1])
Lp2 = np.polynomial.laguerre.lagval(x,[0,0,1])
Lp3 = np.polynomial.laguerre.lagval(x,[0,0,0,1])
Lp4 = np.polynomial.laguerre.lagval(x,[0,0,0,0,1])

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(x,Lp0,'k-.', lw=2, label='n=0')
plt.plot(x,Lp1,'r-',  lw=2, label='n=1')
plt.plot(x,Lp2,'g:',  lw=2, label='n=2')
plt.plot(x,Lp3,'b-.', lw=2, label='n=3')
plt.plot(x,Lp4,'m--', lw=2, label='n=4')
plt.xlabel('x'); plt.ylabel('L_n (x)')
plt.legend(loc='upper left', numpoints = 1 )
plt.axis([0, 7, -10, 15])
plt.show()
fig.savefig('USElaguerrePoly101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.1.13의 결과와 같다. ■

2.1.5 Hermite함수

제n차 Hermite함수 $H_n(x)$를 다음과 같이 정의한다.

$$H_n(x) \doteq [-1]^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2} \tag{2.1.21}$$

명제 2.1.12

Hermite함수열 $\{H_n\}$은 다음 식들을 만족한다.

(a) $H_n(x) = \sum_{k=0}^{\lfloor n/2 \rfloor} [-1]^k \frac{n!}{k!(n-2k)!} [2x]^{n-2k}$

(b) $H_{n+1}(x) = 2xH_n(x) - H_n'(x)$

(c) $H_n'(x) = 2nH_{n-1}(x)$

(d) $H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x)$

여기서 $\lfloor a \rfloor$는 a를 넘지 않는 최대정수이다.

증명. **(a)** 다음 식이 성립한다.

$$H_n(x) = [-1]^n e^{x^2} \frac{d^{n-1}}{dx^{n-1}} \left(\frac{de^{-x^2}}{dx} \right) = [-1]^n e^{x^2} \frac{d^{n-1}}{dx^{n-1}} \left(-2xe^{-x^2} \right) \tag{1}$$

같은 방법으로 미분을 계속하면, 성질 (a)가 성립함을 알 수 있다.

(b) 식 (2.1.21)에서 알 수 있듯이, 다음 식이 성립한다.

$$H_n'(x) = 2x[-1]^n e^{x^2} \frac{d^n}{dx^n} \left(e^{-x^2} \right) + [-1]^n e^{x^2} \frac{d^{n+1}}{dx^{n+1}} \left(e^{-x^2} \right) = 2xH_n(x) - H_{n+1}(x). \tag{2}$$

(c) 다음 식들이 성립한다.

$$H_n'(x) = 2n \sum_{k=0}^{\lfloor [n-1]/2 \rfloor} [-1]^k \frac{(n-1)!}{k!(n-1-2k)!} [2x]^{n-1-2k} = 2nH_n(x) \tag{3}$$

여기서 첫 번째 등호와 두 번째 등호는 성질 (a)에 의해서 성립한다.

(d) 성질 (b)와 성질 (c)로부터 성질 (d)가 성립함을 알 수 있다. ■

명제 2.1.12에서 알 수 있듯이, 다음 식들이 성립한다.

$$H_0(x) = 1, \quad H_1(x) = 2x, \quad H_2(x) = 4x^2 - 2 \tag{2.1.22}$$

$$H_3(x) = 8x^3 - 12x, \quad H_4(x) = 16x^4 - 48x^2 + 12 \tag{2.1.23}$$

$$H_5(x) = 32x^5 - 160x^3 + 120x \tag{2.1.24}$$

$$H_6(x) = 64x^6 - 480x^4 + 720x^2 - 120 \tag{2.1.25}$$

명제 2.1.13

Hermite함수열 $\{H_n\}$은 다음 식을 만족한다.

(a) $\sum_{n=0}^{\infty} \frac{1}{n!} H_n(x) t^n = \exp(-t^2 + 2tx)$

(b) $\int_{-\infty}^{\infty} e^{-x^2} H_m(x) H_n(x) dx = \sqrt{\pi} 2^n n! \delta_{m,n}$

증명. **(a)** 다음 함수를 정의하자.

$$g(x,t) \doteq \sum_{n=0}^{\infty} \frac{1}{n!} H_n(x) t^n \tag{1}$$

다음 식들이 성립한다.

$$\begin{aligned} \frac{\partial g(x,t)}{\partial x} &= \sum_{n=0}^{\infty} \frac{1}{n!} H_n'(x) t^n = \sum_{n=0}^{\infty} \frac{1}{n!} 2n H_{n-1}(x) t^n \\ &= 2t \sum_{n=1}^{\infty} \frac{1}{(n-1)!} H_{n-1}(x) t^{n-1} = 2t \sum_{n=0}^{\infty} \frac{1}{n!} H_n(x) t^n = 2t g(x,t) \end{aligned} \tag{2}$$

명제 2.1.12의 성질 (a)에서 알 수 있듯이, 다음 식들이 성립한다.

$$H_{2n}(0) = [-1]^n \frac{(2n)!}{n!}, \quad H_{2n+1}(0) = 0 \tag{3}$$

따라서 다음 식들이 성립한다.

$$g(0,t) = \sum_{n=0}^{\infty} \frac{1}{n} H_n(0) t^n = \sum_{m=0}^{\infty} \frac{1}{m!} [-1]^m t^{2m} = \exp(-t^2) \tag{4}$$

미분방정식 (2)를 풀면, 경계조건 (4)를 만족하는 해는 다음과 같다.

$$g(x,t) = \exp(-t^2 + 2tx) \tag{5}$$

이 $g(x,t)$를 Hermite함수열의 모함수(generating function)라 한다.

(b) L'Hospital정리를 사용해서, 임의의 자연수들 l과 m에 대해서 다음 식이 성립함을 증명할 수 있다.

$$\lim_{|x| \to \infty} \left| H_m(x) \frac{d^l}{dx^l} e^{-x^2} \right| = 0 \tag{6}$$

각 $m \leq n$에 대해서, 다음 식들이 성립한다.

$$\begin{aligned} [-1]^n \int_{-\infty}^{\infty} e^{-x^2} H_m(x) H_n(x) dx &= \int_{-\infty}^{\infty} H_m(x) \frac{d^n}{dx^n} e^{-x^2} dx \\ &= H_m(x) \frac{d^{n-1}}{dx^{n-1}} e^{-x^2} \Big|_{-\infty}^{\infty} - \int_{-\infty}^{\infty} H_m'(x) \frac{d^{n-1}}{dx^{n-1}} e^{-x^2} dx \\ &= -2m \int_{-\infty}^{\infty} H_{m-1}(x) \frac{d^{n-1}}{dx^{n-1}} e^{-x^2} dx \end{aligned} \tag{7}$$

여기서 첫 번째 등호는 식 (2.1.21)에 의해서, 두 번째 등호는 부분적분에 의해서, 그리고 세 번째 등호는 식 (6)에 의해서 성립한다. 같은 방법을 반복적용하면, 각 $m(\leq n)$에 다음 식이 성립함을 알 수 있다.

$$[-1]^n \int_{-\infty}^{\infty} e^{-x^2} H_m(x) H_n(x) dx = [-1]^m 2^m m! \int_{-\infty}^{\infty} H_0(x) \frac{d^{n-m}}{dx^{n-m}} e^{-x^2} dx \tag{8}$$

만약 $m = n$이면, 식 (8)로부터 다음 식들이 성립함을 알 수 있다.

$$\int_{-\infty}^{\infty} e^{-x^2} H_n^2(x) dx = 2^n n! \int_{-\infty}^{\infty} e^{-x^2} dx = 2^n n! \sqrt{\pi} \tag{9}$$

만약 $m < n$이면, 식 (8)을 다음과 같이 쓸 수 있다.

$$[-1]^n \int_{-\infty}^{\infty} e^{-x^2} H_m(x) H_n(x) dx = [-1]^m 2^m m! \frac{d^{n-m-1}}{dx^{n-m-1}} e^{-x^2} \Big|_{-\infty}^{\infty} = 0 \tag{10}$$

여기서 세 번째 등호는 식 (6)에 의해서 성립한다.

성질 (a)를 사용해서 성질 (b)를 증명할 수도 있다. 성질 (a)에서 알 수 있듯이, 다음 식이 성립한다.

$$\sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \frac{1}{m!n!} \exp(-x^2) H_m(x) H_n(x) t^m s^n = \exp(-x^2 - t^2 + 2tx - s^2 + 2sx) \tag{11}$$

따라서, 다음 식들이 성립한다.

$$\begin{aligned}
&\sum_{m=0}^{\infty} \sum_{n=0}^{\infty} \frac{1}{m!n!} \int_{-\infty}^{\infty} \exp(-x^2) H_m(x) H_n(x) dx t^m s^n \\
&= \int_{-\infty}^{\infty} \exp(-x^2 - t^2 + 2tx - s^2 + 2sx) dx \\
&= \int_{-\infty}^{\infty} \exp(-[x - s - t]^2) e^{2st} dx \\
&= \sqrt{\pi} e^{2st} = \sqrt{\pi} \sum_{n=0}^{\infty} \frac{1}{n!} 2^n [st]^n
\end{aligned} \tag{12}$$

식 (12)의 양변에서 항의 계수를 비교하면, 식 (9)와 식 (10)이 성립함을 알 수 있다. ■

명제 2.1.13의 성질 (b)에서 알 수 있듯이, Hermite함수열 $\{H_n\}$은 가중함수가 다음과 같은 직교함수열이다.

$$w(x) = e^{-x^2}, \quad (x \in (-\infty, \infty)) \tag{2.1.26}$$

즉, n 차 Hermite함수 $H_n(x)$ 와 임의의 $[n-1]$ 차 이하 멱함수는 가중함수 $w(x) = e^{-x^2}$ 에 대해서 직교이다. 이 직교성을 이용해서, 명제 2.1.12의 성질 (d) 를 다음과 같이 증명할 수 있다.

명제 2.1.14

Hermite함수열 $H_n(x)$ 는 다음 식들을 만족한다.

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x), \quad (n = 1, 2, \cdots)$$

증명. 멱함수 $xH_n(x)$ 는 $[n+1]$ 차 멱함수이다. 따라서 $xH_n(x)$ 를 다음과 같이 표기할 수 있다.

$$xH_n(x) = \sum_{k=0}^{n+1} c_k H_k(x) \tag{1}$$

각 $j(= 0, 1, \cdots, n-2)$ 에 대해서 다음 식들이 성립한다.

$$\begin{aligned} c_j &= c_j \frac{1}{2^j j!\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-x^2} H_j^2(x)dx = \frac{1}{2^j j!\sqrt{\pi}} \sum_{k=0}^{n+1} c_k \int_{-\infty}^{\infty} e^{-x^2} H_k(x)H_j(x)dx \\ &= \frac{1}{2^j j!\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-x^2} xH_n(x)H_j(x)dx = \frac{1}{2^j j!\sqrt{\pi}} \int_{-\infty}^{\infty} e^{-x^2} H_n(x)[xH_j(x)]dx = 0 \end{aligned} \tag{2}$$

여기서 첫 번째와 두 번째 등호들은 명제 2.1.13의 성질 (b) 에 의해서, 세 번째 등호는 식 (1) 에 의해서, 그리고 다섯 번째 등호는 $xH_j(x)$ 가 최대 $[n-1]$ 차원이어서 성립한다. 따라서, 다음 식이 성립한다.

$$xH_n(x) = c_{n+1}H_{n+1}(x) + c_nH_n(x) + c_{n-1}H_{n-1}(x) \tag{3}$$

명제 2.1.12의 성질 (a) 를 식 (3) 에 적용한 다음 양변의 계수들을 비교하면, 증명이 끝난다. ■

명제 2.1.15

Hermite함수 $H_n(x)$ 는 n 개 실근들을 갖는다.

증명. 명제 2.1.13의 증명의 식 (6)에서 알 수 있듯이, 임의의 자연수들 l과 m에 대해서 다음 식이 성립한다.

$$\lim_{|x|\to\infty}\left|H_m(x)\frac{d^l}{dx^l}e^{-x^2}\right| = 0 \tag{1}$$

함수 $f(x) = e^{-x^2}$은 $x = -\infty$과 $x = \infty$에서 근을 갖는다. 식 (1)에서 알 수 있듯이 함수 $f'(x)$도 $x = -\infty$과 $x = \infty$에서 근을 가지므로, Rolle정리에 의해 구간 $(-\infty, \infty)$에서 실근 1개를 갖음을 알 수 있다. 식 (1)에서 알 수 있듯이 함수 $f''(x)$도 $x = -\infty$과 $x = \infty$을 근으로 가지므로, Rolle정리에 의해 구간 $(-\infty, \infty)$에서 실근 2개를 갖음을 알 수 있다. 같은 방법을 반복적용하면, 식 (1)에서 알 수 있듯이 함수 $f^{(n)}(x)$도 $x = -\infty$과 $x = \infty$을 근으로 갖으며 따라서 Rolle정리에 의해 구간 $(-\infty, \infty)$에서 실근 n개를 갖음을 알 수 있다. 즉, Hermite 함수 $H^{(n)}(x)$는 구간 $(-\infty, \infty)$에서 n개 실근들을 갖는다. ■

예제 2.1.16 Hermite함수를 구하기 위해서, 다음 MATLAB프로그램 MakeHermitePoly101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename MakeHermitePoly101.m
% Make Hermite Polynomials
% Programmed by CBS
% ------------------------------------------------------------------------
clear, clf
x = -2:0.05:2;
H0 = x-x+1;
H1 = 2*x;
H2 = 2*x.*H1 - 2*1*H0;
H3 = 2*x.*H2 - 2*2*H1;
H4 = 2*x.*H3 - 2*3*H2;
plot(x,H0,'k.',x,H1,'r-',x,H2,'g:',x,H3,'b-.', ...
     x,H4,'m--','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','Best')
xlabel('x','fontsize',12,'fontangle','italic')
ylabel('H_n','fontsize',12,'fontangle','italic','rotation',0)
axis([-2 2 -40 60 ])
saveas(gcf,'MakeHermitePoly101','jpg')
save('MakeHermitePoly101.txt','H0','H1','H2','H3','H4')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 MakeHermitePoly101.m을 실행하면, Hermite함수들 $\{H_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.10을 출력한다.

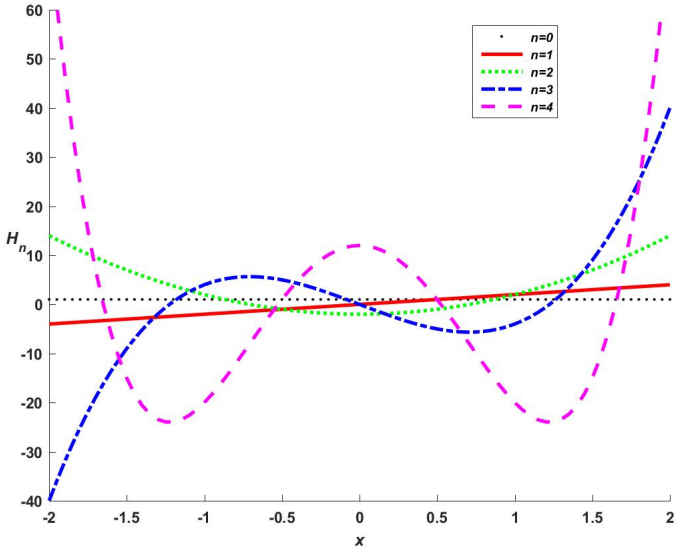


그림 2.1.10. Hermite함수 I

예제 2.1.17 Python을 사용해서 예제 2.1.16을 다시 다루기 위해서, 다음 Python프로그램 MakeHermitePoly101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 2, num=201)
H0 = x-x+1.0;
H1 = 2*x;
H2 = 2*x*H1 - 2*1*H0;
H3 = 2*x*H2 - 2*2*H1;
H4 = 2*x*H3 - 2*3*H2;

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(x, H0,'k-.', lw=2, label='n=0')
plt.plot(x, H1,'r-',  lw=2, label='n=1')
plt.plot(x, H2,'g:',  lw=2, label='n=2')
plt.plot(x, H3,'b-.', lw=2, label='n=3')
plt.plot(x, H4,'m--', lw=2, label='n=4')
plt.xlabel('x'); plt.ylabel('H_n (x)')
plt.axis([-2, 2, -40, 60 ])
plt.show()
fig.savefig('MakeHermitePoly101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.1.16의 결과와 같다. ■

예제 2.1.18 MATLAB함수 hermiteH.m을 사용해서, Hermite함수를 구할 수 있다. 이 MATLAB함수의 사용법은 다음과 같다.

```
>> cp = hermiteH(m,x)
```

MATLAB함수 hermiteH.m를 사용해서 Hermite함수를 구하기 위해서, 다음 MATLAB 프로그램 USEhermitePoly101.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename USEhermitePoly101.m
% hermiteH Polynomials by MATLAB function hermiteH.m
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
x = -2:0.01:2;
hp00 = hermiteH(0,x); hp0 = hp00(1,:);
hp11 = hermiteH(1,x); hp1 = hp11(1,:);
hp22 = hermiteH(2,x); hp2 = hp22(1,:);
hp33 = hermiteH(3,x); hp3 = hp33(1,:);
hp44 = hermiteH(4,x); hp4 = hp44(1,:);
plot(x,hp0,'k-.',x,hp1,'r-',x,hp2,'g:',x,hp3,'b-.', ...
     x,hp4,'m--','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','Best')
xlabel('x','fontsize',12,'fontangle','italic')
ylabel('H_n','fontsize',12,'fontangle','italic','rotation',0)
axis([-2 2 -40 60 ])
saveas(gcf,'USEhermitePoly101','jpg')
save('USEhermitePoly101.txt','hp00','hp11','hp22','hp33','hp44')
% End of hprogram
% ------------------------------------------------------------------
```

이 MATLAB프로그램 USEhermitePoly.m을 실행하면, Hermite함수들 $\{H_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.11을 출력한다. 그림 2.1.11는 그림 2.1.10과 동일하다.

예제 2.1.19 MATLAB함수 hermiteH.m은 수치적으로 뿐아니라 심볼릭으로도 사용할 수 있다. MATLAB함수 hermiteH.m를 사용해서 심볼릭으로 Hermite함수를 구하기 위해서, 다음 MATLAB프로그램 USEhermiteH101.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename USEhermiteH101.m
% Hermite Polynomials by MATLAB function hermiteH.m
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
syms x
hold all
for n=0:4
    ezplot(hermiteH(n,x))
```

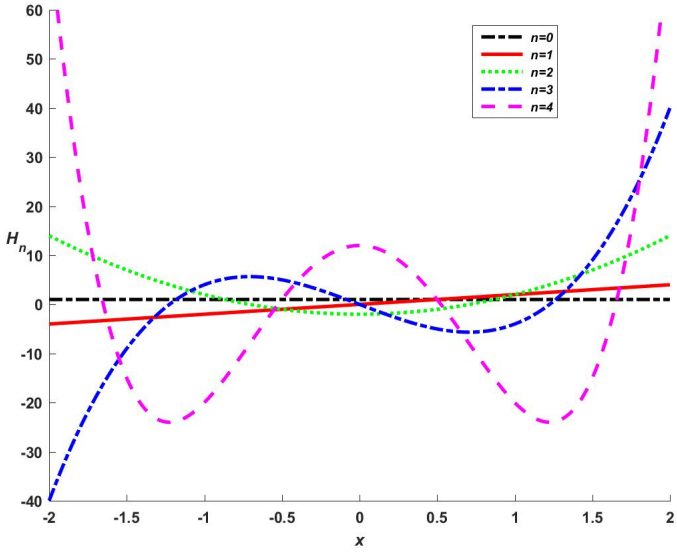


그림 2.1.11. Hermite함수 II

```
end
axis([-2 2 -40 60 ])
set(gca,'fontsize',11,'fontweigh','bold')
box off
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','best')
xlabel('x','fontsize',12,'fontangle','italic')
ylabel('H_n','fontsize',12,'fontangle','italic','rotation',0)
title('\bf Hermite Polynomials')
hold off
saveas(gcf,'USEhermiteH101','jpg')
% End of lprogram
% --------------------------------------------------------------
```

이 MATLAB프로그램 USEhermiteH.m을 실행하면, Hermite함수들 $\{H_n(x)|n = 0, 1, 2, 3, 4\}$를 계산하고, 또한 그림 2.1.12를 출력한다. 그림 2.1.12는 그림 2.1.10과 동일하다. ■

예제 2.1.20 Python을 사용해서 예제 2.1.18을 다시 다루기 위해서, 다음 Python프로그램 USEhermitePoly101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2,2, num=201)
Lp0 = np.polynomial.hermite.hermval(x,[1])
Lp1 = np.polynomial.hermite.hermval(x,[0,1])
```

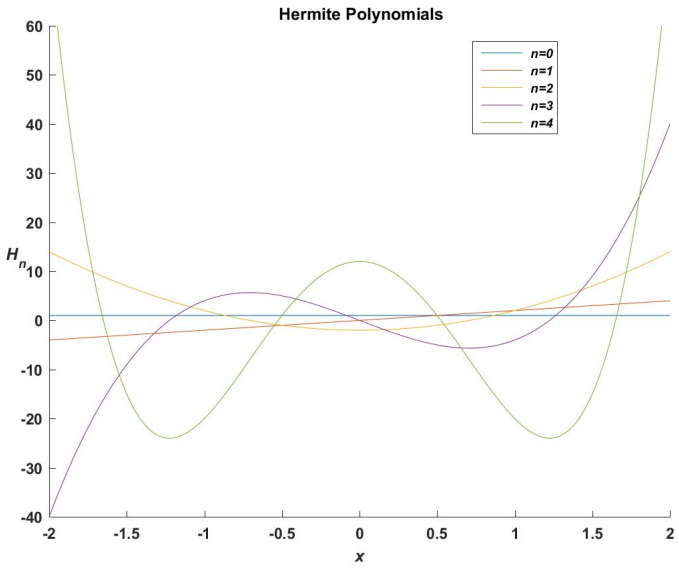


그림 2.1.12. Hermite함수 III

```
Lp2 = np.polynomial.hermite.hermval(x,[0,0,1])
Lp3 = np.polynomial.hermite.hermval(x,[0,0,0,1])
Lp4 = np.polynomial.hermite.hermval(x,[0,0,0,0,1])

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(x,Lp0,'k-.', lw=2, label='n=0')
plt.plot(x,Lp1,'r-',  lw=2, label='n=1')
plt.plot(x,Lp2,'g:',  lw=2, label='n=2')
plt.plot(x,Lp3,'b-.', lw=2, label='n=3')
plt.plot(x,Lp4,'m--', lw=2, label='n=4')
plt.xlabel('x'); plt.ylabel('L_n (x)')
plt.legend(loc='upper left', numpoints = 1 )
plt.axis([-2,2,-40,60])
plt.show()
fig.savefig('USEhermitePoly101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.1.18의 결과와 같다. ■

제2.2절 직교함수와 최소제곱추정

구간 $[a,b]$에서 함수 $f(x)$를 근사시키는 멱함수 $p_n(x)$를 구하기로 하자. 여기서 멱함수 $p_n(x)$의 차수는 n 이하라고 하자. 이러한 근사식을 구하기 위해서, 다음과 같이 최적화문제를

살펴보자.

$$\min_{p_n} \int_a^b w(x)[f(x) - p_n(x)]^2 dx \tag{2.2.1}$$

가중함수 w에 대한 직교함수열을 $\{\phi_n \mid n = 0, 1, 2, \cdots\}$이라 하자. 우리의 목적은 다음 오차제곱합 S를 최소화하는 것이다.

$$S \doteq \int_a^b w(x) \left[f(x) - \sum_{k=0}^{n} c_k \phi_k(x) \right]^2 dx \tag{2.2.2}$$

이 가중최소제곱법의 정규방정식(normal equation)들은 다음과 같다.

$$\frac{\partial S}{\partial c_k} = -2 \int_a^b w(x) \left[f(x) - \sum_{i=0}^{n} c_i \phi_i(x) \right] \phi_k(x) dx = 0, \quad (k = 0, 1, \cdots, n) \tag{2.2.3}$$

따라서 다음 식이 성립한다.

$$\int_a^b w(x) f(x) \phi_k(x) dx = \sum_{i=0}^{n} c_i \int_a^b w(x) \phi_i(x) \phi_k(x) dx \tag{2.2.4}$$

식 (2.2.4)와 $\{\phi_n\}$의 직교성에 의해서, 다음 식들이 성립한다.

$$c_k = \frac{< f, \phi_k >_w}{< \phi_k, \phi_k >_w}, \quad (k = 0, 1, \cdots, n) \tag{2.2.5}$$

이에 해당하는 함수 $f(x)$의 근사식 $p_n(x)$는 다음과 같다.

$$p_n(x) = \sum_{k=0}^{n} \frac{< f, \phi_k >_w}{< \phi_k, \phi_k >_w} \phi_k(x) \tag{2.2.6}$$

식 (2.2.6)의 특수한 경우로서, 함수 $f(x)$가 n차 이하인 멱함수이면 다음 등식이 성립한다.

$$f(x) = \sum_{k=0}^{n} \frac{< f, \phi_k >_w}{< \phi_k, \phi_k >_w} \phi_k(x) \tag{2.2.7}$$

다음 부등식들이 성립한다.

$$\begin{aligned} \|f\|_w^2 \geq \|p_n\|_w &= \left\langle \sum_{k=0}^{n} \frac{< f, \phi_k >_w}{< \phi_k, \phi_k >_w} \phi_k(x), \sum_{j=0}^{n} \frac{< f, \phi_j >_w}{< \phi_j, \phi_j >_w} \phi_j(x) \right\rangle_w \\ &= \sum_{k=0}^{n} \sum_{j=0}^{n} \frac{< f, \phi_k >_w}{< \phi_k, \phi_k >_w} \frac{< f, \phi_j >_w}{< \phi_j, \phi_j >_w} \langle \phi_k(x), \phi_j(x) \rangle_w \end{aligned} \tag{2.2.8}$$

따라서 다음 부등식들이 성립한다.

$$\|f\|_w^2 \geq \sum_{k=0}^{n} \frac{< f, \phi_k >_w}{< \phi_k, \phi_k >_w} \frac{< f, \phi_k >_w}{< \phi_k, \phi_k >_w} \langle \phi_k(x), \phi_k(x) \rangle_w = \sum_{k=0}^{n} \frac{< f, \phi_k >_w^2}{\|\phi_k\|_w^2} \tag{2.2.9}$$

만약 $\|f\|_w$가 유한이면, 다음 부등식이 성립한다.

$$\sum_{n=0}^{\infty} \frac{< f, \phi_n >_w^2}{\|\phi_n\|_w^2} \leq \|f\|_w^2 \tag{2.2.10}$$

정의 2.2.1

어떤 함수족 $\mathcal{F}$에 속하는 임의 함수 $f(x)$에 대해 다음 등식이 성립하면, 직교계 $\{\phi_n\}$은 함수족 $\mathcal{F}$에 대해 완비(complete)라고 한다.

$$\sum_{n=0}^{\infty} \frac{< f, \phi_n >_w^2}{\|\phi_n\|_w^2} = \|f\|_w^2$$

이 식을 Parseval등식이라 부른다. □

만약 직교계가 완비라면, 식 (2.2.6)에서 정의한 $p_n(x)$에 대해 다음 식이 성립함이 알려져 있다.

$$\lim_{n \to \infty} \|f(x) - p_n(x)\|_w = 0 \tag{2.2.11}$$

예제 2.2.1 Legendre함수를 사용해서 다음 함수의 근사식을 구해보자.

$$f(x) = x^4 - \pi x^3 + ex^2 - 1.234x + 0.9800, \quad (-1 \leq x \leq 1) \tag{1}$$

이 근사식을 구하기 위해서, 다음 MATLAB프로그램 LegendreRegress101.m을 실행하자.

```
% ------------------------------------------------------------------
% Filename LegendreRegress101.m
% Regression Analysis 1 with Legendre Polynomials
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
N = 4;
x = -1:0.2:1;
f = x.^4 - pi*x.^3 + exp(1)*x.^2 - 1.234*x + 0.9800;
p1(1,:) = x-x+1;  % Legendre w/ order 0
```

```
p1(2,:) = x;       % Legendre w/ order 1
for i1 = 1:1:N-1
    p1(i1+2,:)=(2*i1+1)/(i1+1)*x.*p1(i1+1,:)-i1/(i1+1)*p1(i1,:);
end
% regression coefficient
for i1 = 1:1:N+1
    c1(i1)=(f*p1(i1,:)')/(p1(i1,:)*p1(i1,:)');
end
c1
pp = c1*p1;          % fitted value
err = f - pp;        % error = observation-fitted value
% Plotting
plot(x,f,'b-.',x,f,'rd',x,pp,'k-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
legend('Function','Data points','Legendre',1)
xlabel('\bf x','fontsize',12)
ylabel('\bf f(x)','fontsize',12,'rotation',0)
saveas(gcf,'LegendreRegress101','jpg')
save('LegendreRegress101.txt','p1','c1')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 LegendreRegress101.m에서는 11개 관찰점들을 사용해서 회귀식을 구한다. 이 MATLAB프로그램을 실행하면, 다음 회귀계수들을 출력한다.

$$c_0 = 2.3521, \quad c_1 = -3.4708, \quad c_2 = 3.2025 \tag{2}$$

$$c_3 = -2.6212, \quad c_4 = 2.4978 \tag{3}$$

이 MATLAB프로그램 LegendreRegress101.m을 실행하면, 함수 $f(x)$와 Legendre근사식 $p_n(x)$를 그린 그림 2.2.1을 출력한다. 그림 2.2.1에서 알 수 있듯이, 11개 관찰점들을 사용한 Legendre근사식 $p(x)$는 함수 $f(x)$에 상당히 가깝다고 할 수 있다. ■

예제 2.2.2 Python을 사용해서 예제 2.2.1을 다시 다루기 위해서, 다음 Python프로그램 LegendreRegress101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

N = 4;
Nobs = 11;
x = np.linspace(-1,1, num=Nobs)
f = x**4 - np.pi*x**3 + np.exp(1)*x**2 - 1.234*x + 0.9800;
p1 = np.zeros((N+1,Nobs),float)
p1[0,:] = x-x+1.0;  # Legendre w/ order 0
```

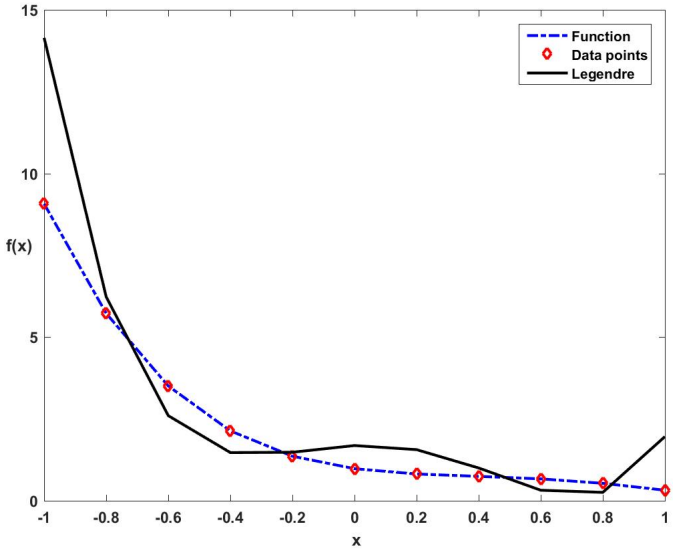


그림 2.2.1. Legendre회귀식

```
p1[1,:] = x;          # Legendre w/ order 1
for i1 in range(1,N):
    p1[i1+1,:] = (2*i1+1)/(i1+1)*x*p1[i1,:] - i1/(i1+1)*p1[i1-1,:]

c1 = np.zeros((1,N+1))
# regression coefficient
for i1 in range(1,N+2):
    c1[0,i1-1] =np.inner(f,p1[i1-1,:])/np.inner(p1[i1-1,:],p1[i1-1,:])
c1
pp = np.dot(c1, p1);          # fitted value; dot is better than inner.
err = f - pp;                 # error = observation-fitted value

# Plotting
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(x,f,'b-.', lw=2, label='Function')
plt.plot(x,f,'rd',  lw=2, label='Data Points')
plt.plot(x,pp[0,:],'k-', lw=2, label='Legendre')
plt.xlabel('x'); plt.ylabel('f(x)')
plt.legend(loc='upper right', numpoints=1)
# plt.axis([-2,2,-40,60])
plt.show()
fig.savefig('LegendreRegress101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.2.1의 결과와 같다. ■

예제 2.2.3 Legendre함수를 사용해서 다음 함수의 근사식을 다시 구해보자.

$$f(x) = x^4 - \pi x^3 + ex^2 - 1.234x + 0.9800, \quad (-1 \le x \le 1) \tag{1}$$

이번에는 101 개 관찰점들을 이용하는 근사식을 구하기 위해서, 다음 MATLAB 프로그램 LegendreRegress102.m을 실행하기로 하자.

```
% ------------------------------------------------------------------------
% Filename LegendreRegress102.m
% Regression Analysis 2 with Legendre Polynomials
% Programmed by CBS
% ------------------------------------------------------------------------
clear, clf
x = linspace(-1,1,101)';
f = x.^4 - pi*x.^3 + exp(1)*x.^2 - 1.234*x + 0.9800;
p0 = x-x+1;
p1 = x;
p2 = 1/2*( 3*x.*p1 - 1*p0 );
p3 = 1/3*( 5*x.*p2 - 2*p1 );
p4 = 1/4*( 7*x.*p3 - 3*p2 );
% Regression coefficient
c0 = (f'*p0)/(p0'*p0);
c1 = (f'*p1)/(p1'*p1);
c2 = (f'*p2)/(p2'*p2);
c3 = (f'*p3)/(p3'*p3);
c4 = (f'*p4)/(p4'*p4);
c = [ c0 c1 c2 c3 c4 ]'
pp = c0*p0 + c1*p1 + c2*p2 + c3*p3 + c4*p4;
error = f - pp;
% Plotting
subplot(2,1,1)
plot(x,f,'b-.',x,pp,'k-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
legend('Function','Legendre Regression',1)
xlabel('\bf x','fontsize',12)
ylabel('\bf f(x)','fontsize',12)
subplot(2,1,2)
plot(x,error,'r-','LineWidth',2.0)
hold on
set(gca,'fontsize',11,'fontweigh','bold')
plot(x,x-x,'k','Linewidth',1.2)
xlabel('\bf x','fontsize',12)
ylabel('\bf Error','fontsize',12)
axis( [ -1 1 -0.9 0.3 ] )
legend('Error','location','SE')
hold off
saveas(gcf,'LegendreRegress102','jpg')
save('LegendreRegress102.txt','c')
% end of program
% ------------------------------------------------------------------------
```

이 MATLAB 프로그램 LegendreRegress102.m에서는 101 개 관찰점들을 사용해서 회귀식을 구한다. 이 MATLAB 프로그램을 실행하면, 다음 회귀계수들을 출력한다.

$$c_0 = 2.1123, \quad c_1 = -3.1564, \quad c_2 = 2.4952 \tag{2}$$

$$c_3 = -1.4649, \quad c_4 = 0.6098 \tag{3}$$

이 MATLAB 프로그램 LegendreRegress102.m을 실행하면, 그림 2.2.2을 출력한다. 그림

2.2.2의 윗 그림은 함수 $f(x)$와 Legendre근사식 $p_n(x)$를 그린 것이고, 아랫 그림은 Legendre 근사식 $p_n(x)$와 함수 $f(x)$의 차이를 그린 것이다. 그림 2.2.2를 출력한다. ■

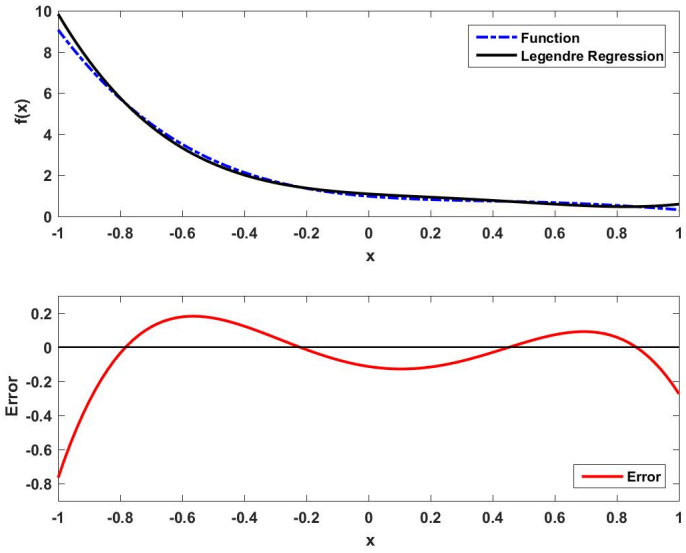


그림 2.2.2. Legendre회귀식에 의한 오차

예제 2.2.4 Python을 사용해서 예제 2.2.3을 다시 다루기 위해서, 다음 Python프로그램 LegendreRegress102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

N = 4;
Nobs = 101;
x = np.linspace(-1,1, num=Nobs)
f = x**4 - np.pi*x**3 + np.exp(1)*x**2 - 1.234*x + 0.9800;
p0 = x-x+1.0;
p1 = x;
p2 = 1/2*( 3*x*p1 - 1*p0 );
p3 = 1/3*( 5*x*p2 - 2*p1 );
p4 = 1/4*( 7*x*p3 - 3*p2 );

# Regression coefficient
c0 = np.inner(f,p0)/np.inner(p0,p0)
c1 = np.inner(f,p1)/np.inner(p1,p1)
c2 = np.inner(f,p2)/np.inner(p2,p2)
c3 = np.inner(f,p3)/np.inner(p3,p3)
c4 = np.inner(f,p4)/np.inner(p4,p4)
C = np.array([c0, c1, c2, c3, c4])    # Coefficients
pp = c0*p0 + c1*p1 + c2*p2 + c3*p3 + c4*p4;
```

```
error = f - pp;

# Plotting
fig = plt.figure()
ax1 = fig.add_subplot(211)
plt.plot(x,f,'g-', lw=2, label='Function')
plt.plot(x,pp,'k.', lw=2, label='Legendre Regression')
plt.xlabel('x'); plt.ylabel('f(x)')
plt.legend(loc='upper right', numpoints=1)
ax1 = fig.add_subplot(212)
plt.plot(x,error,'r-', lw=2, label='Error')
plt.plot(x,x-x,'k', lw=2, label='zero')
plt.xlabel('x'); plt.ylabel('Error')
plt.legend(loc='lower right', numpoints=1)
plt.show()
fig.savefig('LegendreRegress102Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.2.3의 결과와 같다. ■

제2.3절 직교함수근사

명제 2.1.5에서 알 수 있듯이, $\{T_n(\cos\theta)\,|\,n=0,1,\cdots\}$는 다음 식들을 만족한다.

$$\int_0^{2\pi} T_m(\cos\theta)T_n(\cos\theta)d\theta = 0, \quad (m \neq n) \tag{2.3.1}$$

즉, $\{T_n(\cos\theta)\}$는 구간 $[0,2\pi]$에서 가중함수를 $w(\theta)=1$로 하는 직교함수열이다. Chebyshev 함수열을 사용해서 비주기함수의 근사식을 구하기로 하자. 다음 명제는 Judd [31]의 정리 6.4.1을 인용한 것이다.

명제 2.3.1

구간 $[-1,1]$에서 C^k급 함수 f에 대해서 Chebyshev계수들을 다음과 같이 정의하자.

$$c_j \doteq \frac{2}{\pi}\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} T_j(x)f(x)dx, \quad (j=0,1,\cdots)$$

여기서 $T_j(x)$는 j차 Chebyshev함수이다. 또한, 다음 함수를 정의하자.

$$C_n(x) \doteq \frac{1}{2}c_0 + \sum_{j=1}^{n} c_j T_j(x)$$

이러한 조건 하에서, 다음 식을 만족하는 상수 $A(>0)$가 존재한다.

$$\|f - C_n\|_\infty \le A\frac{\ln n}{n^k}$$

즉, 함수열 $\{c_n\}$은 함수 f로 일양수렴한다.

명제 2.3.1에서 알 수 있듯이, 다음 식이 성립한다.

$$f(x) = \frac{1}{2}c_0 + \sum_{j=1}^{\infty} c_j T_j(x) \tag{2.3.2}$$

식 (2.3.2)를 함수 f의 Chebyshev표현(Chebyshev representation)이라 부른다.

예제 2.3.1 Chevyshev표현을 구하는 예로서, 다음 MATLAB프로그램 ChebyshevRepresenation101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename ChebyshevRepresentation101.m
% Chebyshev Coefficient of f(x) on [-1,1]
% Programmed by CBS
% ------------------------------------------------------------------------
function  c = ChebyshevRepresentation101
clear all, close all
syms x
for kk = 0:1:8
    kk1 = kk+1;
    c1(kk1) = quad(@(x)myChebyshevfun(x,kk1),-1,1); % c1(k+1) -> c_k
end
c1
% Plotting
xx = (-1:0.01:1)';                                  % xx -> x
ff = sin(pi*xx);                                    % ff -> f
Tp1(2,:) = xx; Tp1(1,:) = Tp1(2,:)-Tp1(2,:)+1;  % Tp1(i+1,:) -> T_i
Cheby(1,:) = 1/2*c1(1)*Tp1(1,:);                    % Cheby(i+1,:) -> C_i
Cheby(2,:) = Cheby(1,:) + c1(2)*Tp1(2,:);
for jj=2:1:7
    Tp1(jj+1,:) = 2*xx'.*Tp1(jj,:) - Tp1(jj-1,:);
    Cheby(jj+1,:) = Cheby(jj,:) + c1(jj+1)*Tp1(jj+1,:);
end
plot(xx,ff,'k-',xx,Cheby(1,:),'r:',xx,Cheby(2,:),'g-.', ...
     xx,Cheby(4,:),'b--',xx,Cheby(6,:),'m:','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf f(x)','\bf C_0 (x)','\bf C_1 (x)','\bf C_3 (x)', ...
       '\bf C_5 (x)','location','SE')
box off
xlabel('\bf x','fontsize',12), ylabel('\bf C_n (x)','fontsize',12)
axis([-1 1 -1.1 1.1 ])
saveas(gcf,'ChebyshevRepresentation101','jpg')
save('ChebyshevRepresentation101.txt','c1','Cheby')
end
% End of program
% ------------------------------------------------------------------------
```

```
function y = myChebyshevfun(x,kk1)
w = (1-x.^2).^(-1/2);
Ta = 1; Tb = x;
f = sin(pi*x);
y = 2/pi*w.*f;
for kdum=2:1:kk1
    Tc = 2.*x.*Tb - Ta;
    Ta = Tb; Tb = Tc;
    y = 2/pi*w.*Ta.*f;
end
end
% End of program
% --------------------------------------------------------------------
```

이 MATLAB프로그램 ChebyshevRepresentation101.m은 다음 함수의 Chebyshev표현을 구하기 위한 것이다.

$$f(x) = \sin \pi x, \quad (-1 \le x \le 1) \tag{1}$$

이 MATLAB프로그램에서는 MATLAB함수 quad.m을 사용한다. 즉, 적응Simpson공식을 적용해서 적분을 한다. 적응Simpson공식과 quad.m에 대해서는 제8장에서 자세히 다룰 것이다.

이 MATLAB프로그램을 실행하면, Chebyshev계수들이 다음과 같음을 알 수 있다.

$$c_0 = 0, \quad c_1 = 0.5692, \quad c_2 = 0, \quad c_3 = -0.6669 \tag{2}$$

$$c_4 = 0, \quad c_5 = 0.1043, \quad c_6 = 0, \quad c_7 = -0.0068 \tag{3}$$

이 Chebyshev계수들을 사용해서 구한 함수들 $\{C_n(x)|n = 0, 1, 3, 5\}$가 그림 2.3.1에 그려져 있다. 그림 2.3.1에서 알 수 있듯이, $C_5(x)$는 함수 $f(x)$와 아주 가깝다고 할 수 있다. ■

예제 2.3.2 Python을 사용해서 예제 2.3.1을 다시 다루기 위해서, 다음 Python프로그램 ChebyshevRepresentation.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

def AdaptiveSimpson(f, a, b, tol, fa=None, fb=None, fm=None, lev=0):
    if fa == None: fa = f(a)
    if fb == None: fb = f(b)
    if fm == None: fm = f((a + b)/2)
    if lev > 30:
```

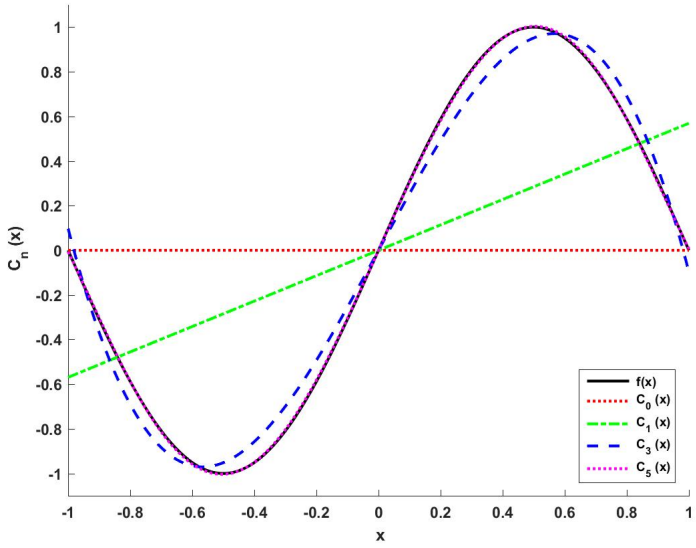


그림 2.3.1. Chebyshev표현

```
        print("Stop iteration and give up on this interval.")
        quad = (b - a)*(fa + 4*fm + fb)/6.
    else:
        h = b - a
        flm = f((3*a+b)/4)
        frm = f((a + 3*b)/4)
        Simpsonl = h*(fa + 4*flm + fm)/12
        Simpsonr = h*(fm + 4*frm + fb)/12
        SimpsonT = Simpsonl + Simpsonr
        Simpson = h*(fa + 4*fm + fb)/6.0
        err = (SimpsonT - Simpson)/15.0
        if abs(err) <= tol * h:
            quad = SimpsonT + err          # Richardson extrapolation
        else:
            m = (a + b)/2.
            quad = AdaptiveSimpson(f, a, m, tol, fa, fm, flm, lev+1) + \
                   AdaptiveSimpson(f, m, b, tol, fm, fb, frm, lev+1)
    return quad


def IntegrandD(xx):
    if abs(1-xx**2) > 10**(-12):
       dum1 = np.sin(np.pi*xx)/np.sqrt(1-xx**2)
    else:
        if xx > 0:
           dum1 = np.sqrt((1-xx)/(1+xx))*(np.pi-np.pi**3*(1-xx)**2)
        else:
           dum1 = np.sqrt((1+xx)/(1-xx))*(np.pi-np.pi**3*(1+xx)**2)
    integrandF = 2/np.pi*dum1
    return integrandF;

c = np.zeros((1,8))
def IntegrandF0(xx):
    return IntegrandD(xx)*1;
c[0,0]= AdaptiveSimpson(IntegrandF0,-1,1,10**(-10));
print("c0 = ", c[0,0])
def IntegrandF1(xx):
```

```
    return IntegrandD(xx)*xx;
c[0,1] = AdaptiveSimpson(IntegrandF1,-1,1,10**(-10));
print("c1 = ", c[0,1])
def IntegrandF2(xx):
    return IntegrandD(xx)*(2*xx**2-1);
c[0,2] = AdaptiveSimpson(IntegrandF2,-1,1,10**(-10));
print("c2 = ", c[0,2])
def IntegrandF3(xx):
    return IntegrandD(xx)*(4*xx**3-3*xx);
c[0,3] = AdaptiveSimpson(IntegrandF3,-1,1,10**(-10));
print("c3 = ", c[0,3])
def IntegrandF4(xx):
    return IntegrandD(xx)*(8*xx**4-8*xx**2+1);
c[0,4] = AdaptiveSimpson(IntegrandF4,-1,1,10**(-10));
print("c4 = ", c[0,4])
def IntegrandF5(xx):
    return IntegrandD(xx)*(16*xx**5-20*xx**3+5*xx);
c[0,5] = AdaptiveSimpson(IntegrandF5,-1,1,10**(-10));
print("c5 = ", c[0,5])
def IntegrandF6(xx):
    return IntegrandD(xx)*(32*xx**6-48*xx**4+18*xx**2-1);
c[0,6] = AdaptiveSimpson(IntegrandF6,-1,1,10**(-10));
print("c6 = ", c[0,6])
def IntegrandF7(xx):
    return IntegrandD(xx)*(64*xx**7 - 112*xx**5 + 56*xx**3 - 7*xx);
c[0,7] = AdaptiveSimpson(IntegrandF7,-1,1,10**(-10));
print("c7 = ", c[0,7])

# Plotting
xx = np.linspace(-1,1,21);                          # xx -> x
ff = np.sin(np.pi*xx);                              # ff -> f
Tp = np.zeros((8,21),float);
Cheby = np.zeros((8,21),float);
Tp[0,:] = np.ones((1,21),float);
Tp[1,:] = xx;
Cheby[0,:]= 1/2*c[0,0]*Tp[0,:];
Cheby[1,:] = Cheby[0,:] + c[0,1]*Tp[1,:];
for jj in range(2,8):
    Tp[jj,:] = 2*xx*Tp[jj-1,:] - Tp[jj-2,:];
    Cheby[jj,:] = Cheby[jj-1,:] + c[0,jj]*Tp[jj,:];

# Plotting
fig = plt.figure()
plt.plot(xx,ff,'k-', lw=2, label='f(x)')
plt.plot(xx,Cheby[0,:],'r:', lw=2, label='C_{0} (x)')
plt.plot(xx,Cheby[1,:],'g-.', lw=2, label='C_{1} (x)')
# plt.plot(xx,Cheby[2,:],'r:', lw=2, label='C_{2} (x)')
plt.plot(xx,Cheby[3,:],'b--', lw=2, label='C_{3} (x)')
# plt.plot(xx,Cheby[4,:],'r:', lw=2, label='C_{4} (x)')
plt.plot(xx,Cheby[5,:],'m-', lw=2, label='C_{5} (x)')
# plt.plot(xx,Cheby[6,:],'r:', lw=2, label='C_{6} (x)')
# plt.plot(xx,Cheby[7,:],'r:', lw=2, label='C_{7} (x)')
plt.xlabel('x'); plt.ylabel('C_{n}(x)')
plt.legend(loc='lower right', numpoints=1)
plt.axis([-1, 1, -1.1, 1.1 ])
plt.show()
fig.savefig('ChebyshevRepresentation101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.3.1의 결과와 같다. ■

다음 명제는 Judd [31]의 정리 6.4.2를 인용한 것이다.

명제 2.3.2

구간 $[-1,1]$에서 C^k급 함수 f를 다음과 같이 Chebyshev표현으로 나타낼 수 있다고 하자.

$$f(x) = \frac{1}{2}c_0 + \sum_{j=1}^{\infty} c_j T_j(x)$$

이러한 조건 하에서, 다음 식들을 만족하는 상수 $c(>0)$가 존재한다.

$$|c_j| \le c\frac{1}{j^k}, \quad (j = 1, 2, \cdots)$$

명제 2.3.2에서 알 수 있듯이, 평활한(smooth) 함수 f에 대해서 Chebyshev계수는 급속히 0에 수렴한다. 실제 문제에서 식 (2.3.2)의 Chebyshev표현, 즉 무한급수를 적용할 수는 없다. 따라서, 다음과 같은 함수 f의 M차 Chebyshev근사식 C_M을 사용한다.

$$C_M(x) = \frac{1}{2}c_0 + \sum_{j=1}^{M} c_j T_j(x) \tag{2.3.3}$$

명제 2.3.2은 M을 선택하는데 유용한 기준이다. 만약 $j(<M)$가 커짐에 따라 $|c_j|$가 급격히 작아지고, 또한 만약 $|\ c_M\ |$이 작다면, 절단오차(truncation error) $\sum_{j=M+1}^{\infty} c_j T_j(x)$는 무시할 수 있고, 오차를 $c_{M+1}T_{M+1}(x)$라 할 수 있다. 그러나, 만약 $|\ c_M\ |$이 $|\ c_{M-1}\ |$보다 충분히 작지 않으면, M차보다 높은 차수의 Chebyshev근사식을 사용해야만 한다. 비록, $|c_M|$이 작다하더라도, 이 점은 반드시 지켜져야 한다. 이러한 경우에는 비록 $|\ c_{M+1}T_{M+1}(x)\ |$이 작다 하더라도 $\sum_{j=M+1}^{\infty} c_j T_j(x)$을 무시할 수 없을 수도 있기 때문이다. 실제 문제에서 우리는 함수 f의 특성을 모르는 경우가 많다. 따라서, 만약 $j(<M)$가 커짐에 따라 $|\,c_j\,|$가 급격히 작아지지 않는다면, 이는 함수 f가 명제 2.3.2이 요구하는 만큼 평활하지 않을 수도 있다. 이 경우에는 Chebyshev근사법을 적용하는 대신에 다른 방법, 예를 들어 뒤에서 설명하게될 스플라인근사법 등을 적용하는 것이 좋다. 일반적으로 Chebyshev계수 c_j를 해석적으로 구하는 것은 쉬운 일이 아니다. 따라서, 수치적으로 c_j를 구하게 될 것이다.

예제 2.3.3 식 (2.1.9)∼식 (2.1.11)에서 알 수 있듯이, 구간 $[-1,1]$에서 다음 식들이 성립

한다.

$$1 = T_0(x), \tag{1}$$

$$x = T_1(x) \tag{2}$$

$$x^2 = \frac{1}{2}[T_0(x) + T_2(x)], \tag{3}$$

$$x^3 = \frac{1}{4}[3T_1(x) + T_3(x)] \tag{4}$$

$$x^4 = \frac{1}{8}[3T_0(x) + 4T_2(x) + T_4(x)] \tag{5}$$

$$x^5 = \frac{1}{16}[10T_1(x) + 5T_3(x) + T_5(x)] \tag{6}$$

$$x^6 = \frac{1}{32}[10T_0(x) + 15T_2(x) + 6T_4(x) + T_6(x)] \tag{7}$$

따라서, 함수 x^{M+1} 의 M 차 Chebyshev근사식과 그 오차 $e_M \doteq x^{M+1} - C_M(x)$는 각각 다음과 같다.

$$M = 1, \qquad x^2 \approx \frac{T_0(x)}{2}, \qquad e_1 = \frac{1}{2}T_2(x) \tag{8}$$

$$M = 2, \qquad x^3 \approx \frac{3T_1(x)}{4}, \qquad e_2 = \frac{1}{4}T_3(x) \tag{9}$$

$$M = 3, \qquad x^4 \approx \frac{3T_0(x) + 4T_2(x)}{8}, \qquad e_3 = \frac{1}{8}T_4(x) \tag{10}$$

$$M = 4, \qquad x^5 \approx \frac{10T_1(x) + 5T_3(x)}{16}, \qquad e_4 = \frac{1}{16}T_5(x) \tag{11}$$

즉, 함수 x^{M+1} 의 M 차 Chebyshev근사식은 $C_M(x)$ 이고 오차는 $T_{M+1}(x)/2^M$ 이다. 이 Chebyshev오차를 그리기 위해서, 다음 MATLAB파일 ChebyshevError101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename ChebyshevError101.m
% Chebyshev Polynomial Errors
% Programmed by CBS
% ------------------------------------------------------------------------
clear, clf
x = -1:0.01:1;
T0 = x-x+1;         err0 = T0-1;
T1 = x;
T2 = 2*x.*T1 - T0; err1 = 1/2*T2;
T3 = 2*x.*T2 - T1; err2 = 1/2^2*T3;
T4 = 2*x.*T3 - T2; err3 = 1/2^3*T4;
T5 = 2*x.*T4 - T3; err4 = 1/2^4*T5;
plot(x,err0,'k-',x,err1,'r:',x,err2,'g-.', ...
     x,err3,'b--',x,err4,'m-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
legend('\it n=0','\it n=1','\it n=2','\it n=3', ...
       '\it n=4','location','Best')
box off
```

```
xlabel('x','fontsize',12)
ylabel('e_n (x)','fontsize',12,'rotation',0)
axis([-1 1 -0.6 0.6 ])
saveas(gcf,'ChebyshevError101','jpg')
% End of program
% --------------------------------------------------------------------------
```

■

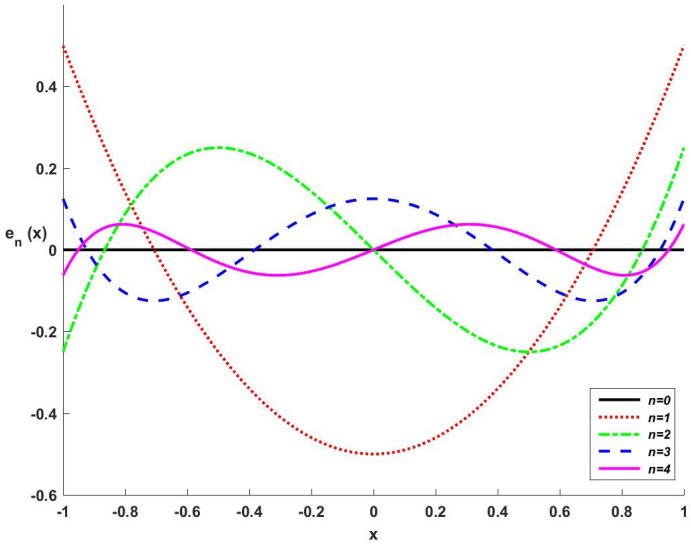


그림 2.3.2. Chebyshev오차

예제 2.3.4 Python을 사용해서 예제 2.3.3을 다시 다루기 위해서, 다음 Python프로그램 ChebyshevError101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1,1,201);
T0 = x-x+1;          err0 = T0-1;
T1 = x;
T2 = 2*x*T1 - T0; err1 = 1/2*T2;
T3 = 2*x*T2 - T1; err2 = 1/2**2*T3;
T4 = 2*x*T3 - T2; err3 = 1/2**3*T4;
T5 = 2*x*T4 - T3; err4 = 1/2**4*T5;

# Plotting
fig = plt.figure()
plt.plot(x,err0,'k-', lw=2, label='n=0')
plt.plot(x,err1,'r:', lw=2, label='n=1')
```

```
plt.plot(x,err2,'g-.', lw=2, label='n=2')
plt.plot(x,err3,'b--', lw=2, label='n=3')
plt.plot(x,err4,'m-', lw=2, label='n=4')
plt.xlabel('x'); plt.ylabel('e_{n}(x)')
plt.legend(loc='lower center', numpoints=1)
plt.axis([-1, 1, -0.6, 0.6 ])
plt.show()
fig.savefig('ChebyshevError101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.3.3의 결과와 같다. ■

예제 2.3.5 Chebyshev함수는 구간 $[-1,1]$에서 정의된다. 그러나, 변수변환을 적용함으로써, 정의역이 $[a,b]$인 함수 $f(x)$에 Chebyshev근사법을 적용할 수 있다. 다음 변수변환을 정의하자.

$$z \doteq z(x) \doteq 2\frac{x-a}{b-a} - 1 \tag{1}$$

따라서 다음 식이 성립한다.

$$z^{-1}(x) = a + \frac{b-a}{2}[x+1] \tag{2}$$

다음 명제가 성립함은 자명하다.

$$a \le x \le b \quad \Rightarrow \quad -1 \le z \le 1 \tag{3}$$

따라서, 함수 $f(x)$의 Chebyshev표현은 다음과 같다.

$$f(x) = \sum_{i=0}^{\infty} c_n T_n(z(x)) \tag{4}$$

여기서 c_n은 다음과 같다.

$$c_n = \frac{\int_a^b \frac{1}{\sqrt{1-z^2(x)}} f(x) T_n(z(x)) dx}{\int_a^b \frac{1}{\sqrt{1-z^2(x)}} T_n^2(z(x)) dx} \tag{5}$$

■

예제 2.3.6 구간 $[0,1]$에서 함수 $f(x) = x^2$의 Chebyshev근사식을 구해보기 위해서, 다음 M-파일 ChebyshevGeneral101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename ChebyshevGeneral101.m
% Chebyshev Coefficient of f(x) on [a,b]
% Programmed by CBS
% ------------------------------------------------------------------
function  c = ChebyshevGeneral101
syms x
a = 0; b = 1;
z = 2*(x-a)/(b-a) -1;
for kk = 0:1:8
    kk1 = kk+1;
    c1(kk1) = quad(@(z)myCGfun(z,kk1),-1,1);
end
c1
% Plotting
zz = (-1:0.01:1)';
xx = (zz+1)/2;
ff = xx.^2;
Tp1(2,:) = zz; Tp1(1,:) = Tp1(2,:)-Tp1(2,:)+1;
Cheby(1,:) = 1/2*c1(1)*Tp1(1,:);
Cheby(2,:) = Cheby(1,:) + c1(2)*Tp1(2,:);
for jj=2:1:7
    Tp1(jj+1,:) = 2*zz'.*Tp1(jj,:) - Tp1(jj-1,:);
    Cheby(jj+1,:) = Cheby(jj,:) + c1(jj+1)*Tp1(jj+1,:);
end
plot(xx,ff,'k-.',xx,Cheby(1,:),'r:',xx,Cheby(2,:),'g-.', ...
     xx,Cheby(3,:),'b--','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf f(x)','\bf C_0 (x)','\bf C_1 (x)','\bf C_2 (x)', ...
        'location','SE')
box off
xlabel('x','fontsize',12)
ylabel('C_n (x)','fontsize',12)
saveas(gcf,'ChebyshevGeneral101','jpg')
save('ChebyshevGeneral101.txt','Cheby')
end
% ------------------------------------------------------------------
function y = myCGfun(z,kk1)
w = (1-z.^2).^(-1/2);
Ta = 1; Tb = z;
f = ((z+1)/2).^2;
y = 2/pi*w.*f;
for kdum=2:1:kk1
    Tc = 2.*z.*Tb - Ta;
    Ta = Tb; Tb = Tc;
    y = 2/pi*w.*Ta.*f;
end
end
% End of program
% ------------------------------------------------------------------
```

다음 변수변환을 정의하자.

$$z \doteq z(x) \doteq 2\frac{x-0}{1-0} - 1 = 2x - 1 \tag{1}$$

함수 $f(x)$ 의 Chebyshev표현은 다음과 같다.

$$f(x) = \sum_{n=0}^{\infty} c_n T_n(z(x)) \tag{2}$$

이 MATLAB프로그램 ChebyshevGeneral101.m을 실행하면, Chebyshev계수들이 다음과 같음을 알 수 있다.

$$c_0 = \frac{3}{8}, \quad c_1 = \frac{1}{2}, \quad c_2 = \frac{1}{8}, \quad c_n = 0, \quad (n \geq 3) \tag{3}$$

식 (2)와 식 (3)에서 알 수 있듯이, 함수 $f(x)$ 의 Chebyshev근사식들은 다음과 같다.

$$C_0(x) = \frac{3}{8} 1_{[0,1]}(x) \tag{4}$$

$$C_1(x) = \left\{ \frac{3}{8} + \frac{1}{2}[2x-1] \right\} 1_{[0,1]}(x) \tag{5}$$

$$C_n(x) = \left\{ \frac{3}{8} + \frac{1}{2}[2x-1] + \frac{1}{8}\{2[2x-1]^2 - 1\} \right\} 1_{[0,1]}(x), \quad (n \geq 3) \tag{6}$$

식 (6)의 $C_n(x)$가 $f(x)$ 임을 쉽게 알 수 있다.

이 Chebyshev계수들을 사용해서 구한 함수들 $\{C_n(x) \,|\, n = 0, 1, 2\}$가 그림 2.3.3에 그려져 있다. 그림 2.3.3에서 $C_2(x)$와 함수 $f(x)$가 동일함을 확인할 수 있다. ■

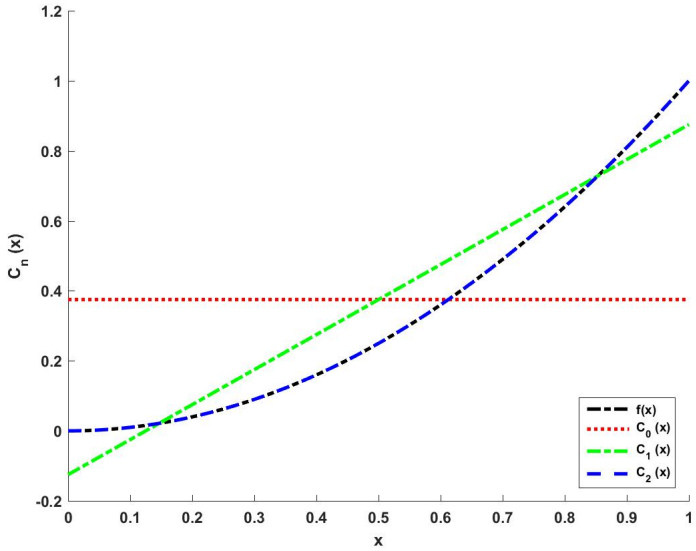


그림 2.3.3. 일반적 정의역에서 Chebyshev표현

예제 2.3.7 Python을 사용해서 예제 2.3.6을 다시 다루기 위해서, 다음 Python프로그램 ChebyshevGeneral101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import scipy.integrate as integrate
import matplotlib.pyplot as plt

# Obtaining Coefficients
def ChebyCoeff(f,a,b):
    z = lambda x: 2*(x-a)/(b-a)-1;
    T0 = lambda x: z(x)-z(x)+1;
    T1 = lambda x: z(x);
    T2 = lambda x: 2*z(x)*T1(x) - T0(x);
    T3 = lambda x: 2*z(x)*T2(x) - T1(x);
    T4 = lambda x: 2*z(x)*T3(x) - T2(x);
    T5 = lambda x: 2*z(x)*T4(x) - T3(x);
    T6 = lambda x: 2*z(x)*T5(x) - T4(x);
    T7 = lambda x: 2*z(x)*T6(x) - T5(x);
    wt = lambda x: 1/np.sqrt(1-z(x)*z(x));
    f0N = lambda x: wt(x)*f(x)*T0(x);  I0N = integrate.quad(f0N, a, b);
    f0D = lambda x: wt(x)*T0(x)*T0(x); I0D = integrate.quad(f0D, a, b);
    I0 = I0N[0]/I0D[0]
    f1N = lambda x: wt(x)*f(x)*T1(x);  I1N = integrate.quad(f1N, a, b);
    f1D = lambda x: wt(x)*T1(x)*T1(x); I1D = integrate.quad(f1D, a, b);
    I1 = I1N[0]/I1D[0]
    f2N = lambda x: wt(x)*f(x)*T2(x);  I2N = integrate.quad(f2N, a, b);
    f2D = lambda x: wt(x)*T2(x)*T2(x); I2D = integrate.quad(f2D, a, b);
    I2 = I2N[0]/I2D[0]
    f3N = lambda x: wt(x)*f(x)*T3(x);  I3N = integrate.quad(f3N, a, b);
    f3D = lambda x: wt(x)*T3(x)*T3(x); I3D = integrate.quad(f3D, a, b);
    I3 = I3N[0]/I3D[0]
    f4N = lambda x: wt(x)*f(x)*T4(x);  I4N = integrate.quad(f4N, a, b);
    f4D = lambda x: wt(x)*T4(x)*T4(x); I4D = integrate.quad(f4D, a, b);
    I4 = I4N[0]/I4D[0]
    f5N = lambda x: wt(x)*f(x)*T5(x);  I5N = integrate.quad(f5N, a, b);
    f5D = lambda x: wt(x)*T5(x)*T5(x); I5D = integrate.quad(f5D, a, b);
    I5 = I5N[0]/I5D[0]
    f6N = lambda x: wt(x)*f(x)*T6(x);  I6N = integrate.quad(f6N, a, b);
    f6D = lambda x: wt(x)*T6(x)*T6(x); I6D = integrate.quad(f6D, a, b);
    I6 = I6N[0]/I6D[0]
    f7N = lambda x: wt(x)*f(x)*T7(x);  I7N = integrate.quad(f7N, a, b);
    f7D = lambda x: wt(x)*T7(x)*T7(x); I7D = integrate.quad(f7D, a, b);
    I7 = I7N[0]/I7D[0]
    return  I0, I1, I2, I3, I4, I5, I6, I7

f = lambda x: x**2
a = 0;   b = 1;
c = ChebyCoeff(f,a,b)
print(c)

# Chebyshev Approximation
zz = np.linspace(-1,1,201);
xx = (zz+1)/2;
ff = xx**2;
Tp = np.zeros((8,201),float);
```

```
Cheby = np.zeros((8,201),float);
Tp[1,:] = zz;    Tp[0,:] = Tp[1,:]-Tp[1,:]+1;
Cheby[0,:] = c[0]*Tp[0,:];
Cheby[1,:] = Cheby[0,:] + c[1]*Tp[1,:];
for jj in range(2,8):
    Tp[jj,:] = 2*zz*Tp[jj-1,:] - Tp[jj-2,:];
    Cheby[jj,:] = Cheby[jj-1,:] + c[jj]*Tp[jj,:];

# Plotting
fig = plt.figure()
plt.plot(xx,ff,'k-.', lw=2, label='f(x)')
plt.plot(xx,Cheby[0,:],'r:', lw=2, label='C_{0} (x)')
plt.plot(xx,Cheby[1,:],'g-.', lw=2, label='C_{1} (x)')
plt.plot(xx,Cheby[2,:],'b--', lw=2, label='C_{2} (x)')
plt.xlabel('x'); plt.ylabel('C_{n}(x)')
plt.legend(loc='upper left', numpoints=1)
plt.axis([0, 1, -0.2, 1.2 ])
plt.show()
fig.savefig('ChebyshevGeneral101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.3.6의 결과와 같다. ■

식 (2.2.6)에서 알 수 있듯이, 직교함수들 $\{\phi_n(x)\}$를 사용한 평활한 함수 $f(x)$의 M차 근사식 $p_M(x)$는 다음과 같다.

$$p_M(x) = \sum_{k=0}^{M} \frac{< f, \phi_k >_w}{< \phi_k, \phi_k >_w} \phi_k(x) \tag{2.3.4}$$

만약 함수 $f(x)$가 적당한 기술적 조건을 만족하면, 이 M차 근사식 $p_M(x)$에 의한 오차 $e_M \doteq f(x) - p_M(x)$는 일반적으로 다음 식을 만족한다.

$$e_M = O(\phi_{M+1}(x)) \tag{2.3.5}$$

제2.4절 미니맥스근사와 Chebyshev절약

2.4.1 일양근사

앞에서 다룬 최소제곱추정법은 $L^2[a,b]$에서 함수 $f(x)$를 멱함수 $p_n(x)$로 근사시키는 것이다. 유감스럽게도 L^2-수렴, 즉 최소제곱추정법은 각 점 x에서 수렴에 대해 정보를 제공하지 않는다. 그러나, 우리는 구간 $[a,b]$에서 함수 f에 일양수렴하는 멱함수열 $\{p_n\}$을 구하고자 한다. 즉,

다음 식을 만족하는 멱함수열 $\{p_n\}$을 구하고자 한다.

$$\lim_{n\to\infty} \max_{x\in[a,b]} |f(x) - p_n(x)| = 0 \tag{2.4.1}$$

식 (2.4.1)을 만족하는 $p_n(x)$를 함수 f의 일양근사함수라 하자. 최소제곱근사법에 의해 구한 근사함수가 구간 $[a,b]$의 총오차(total error)를 최소화하는 반면에, 일양근사법에서는 각 점 $x(\in [a,b])$에서 $p_n(x)$가 $f(x)$의 좋은 근사가 되는 근사함수를 찾는 것이다. 따라서, 일양근사법은 최소제곱근사법보다 더 까다로운 조건을 필요로 하고, 이러한 조건이 만족되면 일양근사법은 최소제곱근사법보다 더 좋은 점별근사(pointwise approximation)를 제공한다. Weierstrass [66]가 1885년에 제시한 다음 정리는 연속함수에 대해 일양수렴하는 근사함수들이 존재함을 보여준다.

명제 2.4.1: Weierstrass근사정리

만약 함수 f가 구간 $[a,b]$에서 연속이면, 즉 f가 $C[a,b]$급이면, 임의의 $\epsilon(>0)$에 대해서, 다음 식을 만족하는 멱함수 p가 존재한다.

$$\max_{x\in[a,b]} |f(x) - p(x)| < \epsilon$$

만약 함수 f가 $C^k[a,b]$급이면, 각 $l(\le k)$에 대해서 다음 식을 만족하는 멱함수열 $\{p_n \,|\, n = 0, 1, \cdots\}$이 존재한다.

$$\lim_{n\to\infty} \max_{x\in[a,b]} \left| f^{(l)}(x) - p_n^{(l)}(x) \right| = 0$$

여기서 p_n은 n차 멱함수이다.

명제 2.4.1에 기술한 Weierstrass근사정리는 이론적으로 유용하지만, 실제 근사식을 구하는데 중요한 것은 아니다.

정의 2.4.1

다음과 같이 구간 $[a,b]$에서 제곱가적분인 함수들 f와 g를 살펴보자.

$$\|\{x_n\}_{n=1}^{\infty}\|_{\infty} \doteq \lim_{k\to\infty} \|\{x_n\}_{n=1}^{\infty}\|_k \doteq \lim_{k\to\infty} \left[\sum_{n=1}^{\infty} |x_n|^k\right]^{1/k} = \sup_n |x_n|$$

예제 2.4.1 명제 2.4.1을 살펴보기 위해서, 차수가 n인 Bernstein기저함수들을 다음과 같이 정의하자.

$$b_{m,n}(x) = \binom{n}{m} x^m (1-x)^{n-m}, \quad (m = 0, 1, \ldots, n) \tag{1}$$

낮은 차수의 Bernstein기저함수들은 다음과 같다.

$$b_{0,0}(x) = 1 \tag{2}$$

$$b_{0,1}(x) = 1 - x, \qquad b_{1,1}(x) = x \tag{3}$$

$$b_{0,2}(x) = (1-x)^2, \quad b_{1,2}(x) = 2x(1-x), \quad b_{2,2}(x) = x^2 \tag{4}$$

$$b_{0,3}(x) = (1-x)^3, \quad b_{1,3}(x) = 3x(1-x)^2, \quad b_{2,3}(x) = 3x^2(1-x), \quad b_{3,3}(x) = x^3 \tag{5}$$

$$b_{0,4}(x) = (1-x)^4, \quad b_{1,4}(x) = 4x(1-x)^3, \quad b_{2,4}(x) = 6x^2(1-x)^2 \tag{6}$$

$$b_{3,4}(x) = 4x^3(1-x), \quad b_{4,4}(x) = x^4 \tag{7}$$

Bernstein기저함수들의 선형결합을 Bernstein멱함수라 부른다. 즉, 차수가 n인 Bernstein 멱함수는 다음과 같다.

$$B_n(x) = \sum_{m=0}^{n} \beta_m b_{m,n}(x) \tag{8}$$

계수 β_m을 Bernstein계수 또는 Bézier계수라 부른다. 구간 $[0,1]$에서 정의되는 연속함수 f에 대해서 다음과 같은 Bernstein멱함수를 살펴보자.

$$B_n(f)(x) \doteq \sum_{m=0}^{n} f\left(\frac{m}{n}\right) b_{m,n}(x) \tag{9}$$

Phillip [43, 제7장]에서 알 수 있듯이, 다음 식들이 성립한다.

$$\lim_{n\to\infty} B_n(f)(x) = f(x) \;\; uniformly\ on\ [0,1] \tag{10}$$

$$\lim_{n\to\infty} \sup\{\,|f(x) - B_n(f)(x)| \,:\, 0 \le x \le 1\,\} = 0 \tag{11}$$

$$\left\| B_n(f)^{(k)} \right\|_\infty \le \frac{(n)_k}{n^k} \left\| f^{(k)} \right\|_\infty \tag{12}$$

$$\left\| f^{(k)} - B_n(f)^{(k)} \right\|_\infty \to 0 \tag{13}$$

여기서 $(n)_k \doteq n[n-1]\cdots[n-k+1]$이다. 식 (11)과 식 (13)에서 알 수 있듯이, Bernstein 멱함수를 사용해서 Weierstrass근사정리를 증명할 수 있다. ■

예제 2.4.2 MATLAB함수 bernstein.m를 사용해서 Bernstein근사함수를 구하기 위해서, 다음 MATLAB프로그램 USEbernstein101.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename USEbernstein101.m
% Bernstein Polynomials by MATLAB function bernstein.m
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
syms t
b00 = bernstein(@(t) sin(2*pi*t),0,t);
b10 = bernstein(@(t) sin(2*pi*t),10,t);
b20 = bernstein(@(t) sin(2*pi*t),20,t);
b30 = bernstein(@(t) sin(2*pi*t),30,t);
b100 = bernstein(@(t) sin(2*pi*t),100,t);
bp0 = ezplot(b00,[0,1]);
bp0.Color='red'; bp0.LineStyle='-'; bp0.LineWidth=2;
set(gca,'fontsize',11,'fontweigh','bold')
hold on
bp1 = ezplot(b10,[0,1]);
bp1.Color='green'; bp1.LineStyle='--'; bp1.LineWidth=2;
bp2 = ezplot(b20,[0,1]);
bp2.Color='blue'; bp2.LineStyle=':'; bp2.LineWidth=2;
bp3 = ezplot(b30,[0,1]);
bp3.Color='c'; bp3.LineStyle='-.'; bp3.LineWidth=2;
bp10 = ezplot(b100,[0,1]);
bp10.Color='k'; bp10.LineStyle='--'; bp10.LineWidth=2;
legend('\it n=0','\it n=10','\it n=20','\it n=30', ...
       '\it n=100','location','Best')
title('\bf Bernstein Polynomials')
plot([0 1],[1 1],'b:',[0 1],[-1 -1],'b:','LineWidth',1.2)
saveas(gcf,'USEbernstein101','jpg')
% End of bprogram
% ------------------------------------------------------------------
```

이 MATLAB프로그램 USEbernstein.m은 구간 $[0,1]$에서 $\sin(2\pi t)$를 Bernstein함수들 $\{B_n(t)|n = 0, 10, 20, 30, 100\}$로 근사시키고, 이 결과를 그린 그림 2.4.1을 출력한다. 그림

2.4.1에서 알 수 있듯이, Bernstein근사가 일양수렴을 하기는 하지만 좋은 근사식을 얻기 위해서 아주 높은 차수의 Bernstein멱함수를 사용해야하는 경우도 있다. ■

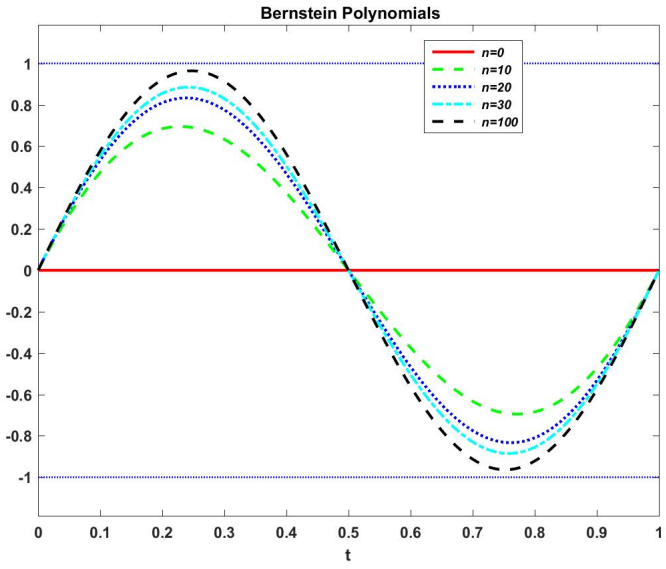


그림 2.4.1. Bernstein근사함수

예제 2.4.3 Python을 사용해서 예제 2.4.2을 다시 다루기 위해서, 다음 Python프로그램 USEbernstein101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: http://stackoverflow.com/questions/12643079/b%C3%A9zier-curve-
    fitting-with-scipy, Modified by CBS
"""
import numpy as np
import matplotlib.pyplot as plt
from scipy.misc import comb

def bernstein_poly(i, n, t):
    # The Bernstein polynomial of n, i as a function of t
    return comb(n, i) * ( t**(n-i) ) * (1 - t)**i

def bezier_curve(points, nPoints, nTimes=1000):
    # Given a set of control points,
    # return the bezier curve defined by the control points.
    # points should be a list of lists, or list of tuples
    #  such as [ [1,1],
    #            [2,3],
    #            [4,5], ..[Xn, Yn] ]
    # nTimes is the number of time steps, defaults to 1000
    # See http://processingjs.nihongoresources.com/bezierinfo/
    nPoints = len(points)
    xPoints = np.array([p[0] for p in points])
    yPoints = np.array([p[1] for p in points])
    t = np.linspace(0.0, 1.0, nTimes)
```

```
    polynomial_array = np.array([ bernstein_poly(i, nPoints-1, t) for i in
    range(0, nPoints)   ])
    xvals = np.dot(xPoints, polynomial_array)
    yvals = np.dot(yPoints, polynomial_array)
    return xvals, yvals

if __name__ == "__main__":
    nPoints = 1
    xpoints = np.linspace(0,1,2);
    ypoints = np.sin(2*np.pi*xpoints);
    points = np.column_stack((xpoints,ypoints));
    xvals00, yvals00 = bezier_curve(points,nPoints,2001)
    nPoints = 11
    xpoints = np.linspace(0,1,nPoints);
    ypoints = np.sin(2*np.pi*xpoints);
    points = np.column_stack((xpoints,ypoints));
    xvals10, yvals10 = bezier_curve(points,nPoints,2001)
    nPoints = 21
    xpoints = np.linspace(0,1,nPoints);
    ypoints = np.sin(2*np.pi*xpoints);
    points = np.column_stack((xpoints,ypoints));
    xvals20, yvals20 = bezier_curve(points,nPoints,2001)
    nPoints = 31
    xpoints = np.linspace(0,1,nPoints);
    ypoints = np.sin(2*np.pi*xpoints);
    points = np.column_stack((xpoints,ypoints));
    xvals30, yvals30 = bezier_curve(points,nPoints,2001)
    nPoints = 101
    xpoints = np.linspace(0,1,nPoints);
    ypoints = np.sin(2*np.pi*xpoints);
    points = np.column_stack((xpoints,ypoints));
    xvals100, yvals100 = bezier_curve(points,nPoints,2001)

# Plotting
fig = plt.figure()
plt.plot(xvals00,yvals00,'r-', lw=2, label='n=0')
plt.plot(xvals10,yvals10,'g--', lw=2, label='n=10')
plt.plot(xvals20,yvals20,'b:', lw=2, label='n=20')
plt.plot(xvals30,yvals30,'c-.', lw=2, label='n=30')
plt.plot(xvals100,yvals100,'k--', lw=2, label='n=100')
plt.plot(xvals00,yvals00+1,'b:', lw=2)
plt.plot(xvals00,yvals00-1,'b:', lw=2)
plt.xlabel('t');
plt.legend(loc='upper right', numpoints=1)
plt.axis([0, 1, -1.2, 1.2 ])
plt.show()
fig.savefig('USEbernstein101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.4.2의 결과와 같다. ■

2.4.2 미니맥스근사

함수 f 에 일양근사하는 멱함수를 찾기 위해서 미니맥스근사법 (minimax approximation) 을 널리 사용한다. 구간 $[a,b]$ 에서 함수 $f(x)$ 를 근사시키는 멱함수 $q(x)$ 를 구하기로 하자. 여기서 멱함수 $q(x)$ 의 차수는 n 이하라고 하자. 이러한 근사식을 구하기 위해서, L^∞ 공간에서 오차 $[f-q]$ 의 하한을 다음과 같이 정의하자.

$$\rho_n(f) \doteq \inf_{\text{degree}(q)\le n} \|f-q\|_\infty = \inf_{\text{degree}(q)\le n} \sup_x |f(x)-q(x)| \tag{2.4.2}$$

다음 명제는 식 (2.4.2) 의 하한을 얻을 수 있다는 것과 이 경우에 오차가 어느 한정된 범위에서 움직인다는 등진동성 (equi-oscilliation property) 에 관한 것이다. 이 명제의 증명은 Phillips & Taylor [44] 를 참조하라.

명제 2.4.2

만약 f 가 $C[a,b]$ 급이면, 다음 식을 만족하는 n 차 멱함수 q_n^* 이 일의적으로 존재한다.

$$\rho_n(f) = \|f-q_n^*\|_\infty$$

이 q_n^* 는 다음 식들을 만족하는 점들 $a \le x_0 < x_1 < \cdots < x_n < x_{n+1} \le b$ 이 적어도 $[n+2]$ 개 존재하는 멱함수이다.

$$f(x_j) - q_n^*(x_j) = m[-1]^j \rho_n(f), \quad (j = 0, 1, \cdots, n+1)$$

여기서 m 은 1 또는 -1 이다. 또한, 이 q_n^* 는 이 등진동성을 만족하는 일의적인 멱함수이다.

예제 2.4.4 구간 $[0,1]$ 에서 함수 $f(x) = e^x$ 의 미니맥스근사법에 의한 근사식들은 다음과 같다.

$$q_0^*(x) = 0.1859141 \tag{1}$$

$$q_1^*(x) = 1.718282x + 0.894067 \tag{2}$$

$$q_2^*(x) = 0.846024x^2 + 0.854752x + 1.008753 \tag{3}$$

$$q_3^*(x) = 0.279979x^3 + 0.421699x^2 + 1.016603x + 0.999455 \tag{4}$$

또한, 다음 식들이 성립한다.

$$\rho_0(f) = 0.859141, \qquad \rho_1(f) = 0.105933 \tag{5}$$

$$\rho_2(f) = 0.008753, \qquad \rho_3(f) = 0.000545 \tag{6}$$

식 (1) ∼ 식 (6)은 Young & Gregory [68, p. 315]에서 인용한 것이다.

미니맥스근사식 $q_3^*(x)$에 의한 오차 $e_3(x) = f(x) - q_3^*(x)$를 그리기 위해서, 다음 M-파일 MinimaxApprox101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename MinimaxApprox101.m
% Minimax approximation
% Programmed by CBS
% ------------------------------------------------------------------------
clear, clf
x = 0:0.001:1;
fx = exp(x);
q0 = x-x+1.859141;
q1 = 1.718282*x + 0.894067;
q2 = 0.846024*x.^2 + 0.854752*x + 1.008753;
q3 = 0.279979*x.^3 + 0.421699*x.^2 + 1.016603*x + 0.999455;
e3 = fx - q3;
xdum = [0 1];
ydum1 = [ 0.000545 0.000545]; ydum2 = - ydum1;
plot(x,e3,'k-',xdum,ydum1,'r--',xdum,ydum2,'b:','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('x','fontsize',13,'fontangle','italic')
ylabel('e_3 (x)','fontangle','italic','fontsize',13)
saveas(gcf,'MinimaxApprox101','jpg')
save('MinimaxApprox101.txt','fx','q3','e3')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 그림 2.4.2이 그려진다. 이 그림에서 명제 2.4.2가 성립함을 확인할 수 있다. ■

예제 2.4.5 Python을 사용해서 예제 2.4.4을 다시 다루기 위해서, 다음 Python프로그램 MinimaxApprox101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0,1,1001);
fx = np.exp(x);
q0 = x-x+1.859141;
```

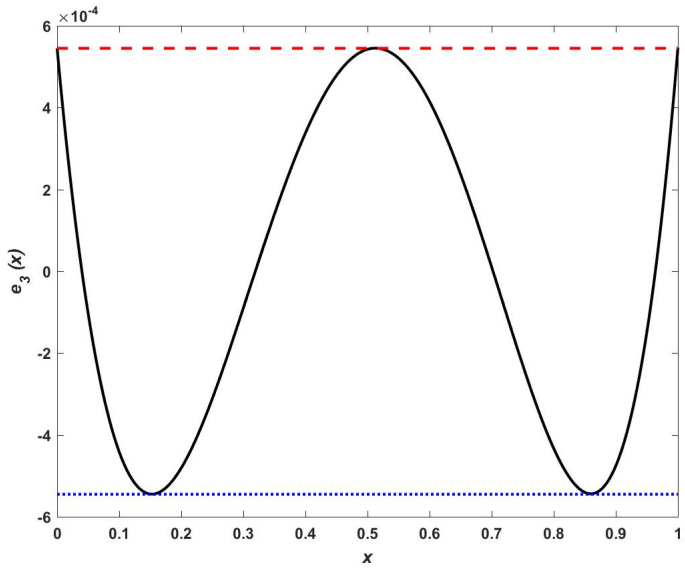


그림 2.4.2. 미니맥스근사에 의한 오차

```
q1 = 1.718282*x + 0.894067;
q2 = 0.846024*x**2 + 0.854752*x + 1.008753;
q3 = 0.279979*x**3 + 0.421699*x**2 + 1.016603*x + 0.999455;
e3 = fx - q3;

# Plotting
xdum = np.array([0, 1]);
ydum1 = np.array([ 0.000545, 0.000545]);
ydum2 = - ydum1;
fig = plt.figure()
plt.plot(x,e3,'k-',lw=2, label='n=0')
plt.plot(xdum,ydum1,'r--', lw=2, label='n=1')
plt.plot(xdum,ydum2,'b:', lw=2, label='n=2')
plt.xlabel('x');  plt.ylabel('e_{3} (x)');
plt.legend(loc='lower center', numpoints=1)
plt.axis([0, 1, -6*10**(-4), 6*10**(-4) ])
plt.show()
fig.savefig('MinimaxApprox101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.4.4의 결과와 같다. ■

명제 2.4.3: Jackson정리

만약 함수 f가 $C^k[a,b]$급이면, 각 $n(\geq k)$에 대해서 다음 식이 성립한다.

$$\rho_n(f) \leq \frac{(n-k)!}{n!}\left[\frac{\pi}{2}\right]^k \left[\frac{b-a}{2}\right]^k \|f^{(k)}\|_\infty$$

명제 2.4.3의 증명은 Powell [45, p. 197]을 참조하라. 명제 2.4.3은 다음과 같은 점에서 중요하다. 첫째, n이 증가함에 따라 근사함수 $q_n^*(x)$의 정밀도(precision)가 좋아진다. 둘째, 평활한 함수일수록 수렴속도가 크다. 그러나, 근사함수 $q_n^*(x)$를 해석적으로 구하는 것은 쉬운 일이 아니다. 따라서, 미니맥스법에 의한 근사함수 $q_n^*(x)$를 계산하는 대신에, 미니맥스법과 마찬가지로 일양수렴성을 제공하는 Chebyshev근사법을 사용하기로 하자.

2.4.3 Chebyshev근사

앞 소절에서 언급했듯이, 미니맥스법에 의한 근사함수를 구하는 것은 어려운 작업이다. 최소제곱근사법을 바탕으로 하는 Chebyshev근사식은 미니맥스법에 의한 근사식과 거의 비슷하다. 따라서, Chebyshev근사식을 미니맥스법에 의한 근사식의 대용으로 사용할 수 있다.

명제 2.4.4

만약 함수 f가 $C^1[-1,1]$급이면, n차 Chebyshev근사식 $C_n(x)$는 다음 식을 만족한다.

$$\rho_n(f) \leq \|f - C_n\|_\infty \leq \left[4 + \frac{4}{\pi^2}\ln n\right]\rho_n(f)$$

명제 2.4.4의 증명은 Rivlin [49, p. 14]을 참조하라. 명제 2.4.4에서 알 수 있듯이, L^2 공간에서 오차를 최소화하는 Chebyshev근사식이 L^∞ 공간에서 오차를 최소화하는 미니맥스 근사식보다 그리 나쁘지 않다. 더구나, Chebyshev근사식은 선형연산을 사용해서 쉽게 구할 수 있으므로, 미니맥스근사식 대신에 Chebyshev근사식을 사용하는 것이 실용적이다.

Chebyshev근사식은 가중함수를 $w(x) = [1-x^2]^{-1/2}$로 하는 직교근사식이다. 앞에서도 언급했듯이, Chebyshev근사식은 일양오차를 갖는다. 반면에, 다른 직교함수를 사용하는 근사식은 오차가 일양이 아니다. 예를 들어, Legendre근사식은 가중함수가 상수이다. 따라서, x가 끝점인 $+1$이나 -1에 가까이 가면, 오차가 커진다. 그 이유는 자명하다. 점 $x=0$에서 좋은 근사식이 되기 위해서는 점 $x=0$의 좌측과 우측의 가까운 점들에서도 좋은 근사식이 되어야 한다. 반면에, 점 $x=1$에서 좋은 근사식이 되기 위해서는 점 $x=1$의 좌측의 가까운

점들에서만 좋은 근사식이 되면 된다. 따라서, 오차제곱합을 최소화한다는 것은 양 끝점들에서 정확도를 희생해서 가운데 점들의 정확도를 높힘을 의미한다. 반면에, Chebyshev근사에서는 양 끝점에 가까울수록 오차에 더 큰 벌칙(penalty)을 주어, 양 끝점에서 오차가 커지는 것을 방지한다.

명제 2.4.4의 $\ln n$ 항이 신경이 쓰이는 독자들도 있을 것이다. 즉, n이 커지면 $\ln n$도 커지므로, 오차의 하한 $\rho_n(x)$이 커질수도 있다는 우려를 할 수도 있다. 그에 대한 해답은 명제 2.4.3에 있다. 즉, 명제 2.4.3을 명제 2.4.4에 대입하면, 다음 명제가 성립함을 알 수 있다.

명제 2.4.5

만약 함수 f가 $C^k[a,b]$급이면, 각 $n(\geq k)$에 대해서 다음 식이 성립한다.

$$\|f - C_n\|_\infty \leq \left[4 + \frac{4}{\pi^2}\ln n\right] \frac{(n-k)!}{n!} \left[\frac{\pi}{2}\right]^k \left[\frac{b-a}{2}\right]^k \|f^{(k)}\|_\infty$$

예제 2.4.6 명제 2.4.5를 예증하기 위해서, 구간 $[-1,1]$에서 정의되는 다음 함수들의 Chebyshev계수들을 구해보자.

$$f_1(x) = \exp(x+1), \qquad f_2(x) = [x+1]^{1/4} \tag{1}$$

$$f_3(x) = \max\{x^3, 0\}, \qquad f_4(x) = \min\{\max\{4[x-0,2], -1\}, 1\} \tag{2}$$

이 함수들의 Chebyshev계수들을 계산하기 위해서, 다음 MATLAB프로그램 ChebyshevErrorAnalysis101.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename ChebyshevErrorAnalysis101.m
% Chebyshev Error Analysis
% Programmed by CBS
% ------------------------------------------------------------------
function ChebyshevErrorAnalysis101
clear all, close all
x = [-1:0.0001:1]';
ii = 0:10;
% case 1
f1 = exp(x+1);
coe1 = ChebyshevCoeff(x,f1)
% case 2
f2 = (x+1).^0.25;
coe2 = ChebyshevCoeff(x,f2)
% case 3
f3 = max( 0, x.^3 );
coe3 = ChebyshevCoeff(x,f3)
```

```
% case 4
f4 = min( max( -1, 4*(x - 0.2) ), 1);
coe4 = ChebyshevCoeff(x,f4)
% plot
hold on
plot(ii,coe1,'r-',ii,coe2,'g:',ii,coe3,'b-.', ...
     ii,coe4,'m--','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
legend('e^{x+1}','(x+1)^{0.25}','max(x^{3},0)', ...
       'min(max(-1,4[x-0.2]),1)','location','Best')
plot(ii,coe1,'k.',ii,coe2,'k.',ii,coe3,'k.', ...
     ii,coe4,'k.','LineWidth',2.0)
xlabel('\bf Order i','fontsize',12)
ylabel('\bf Coefficient c_i','fontsize',12)
hold off
saveas(gcf,'ChebyshevErrorAnalysis101','jpg')
end
% end of program
% ------------------------------------------------------------------
function  c = ChebyshevCoeff(xx,fxx)
% x = column vector of x
% fx = column vector of y
% Chebyshev polynomials
deltax = xx(2) - xx(1);
x = xx(2:end-1); fx = fxx(2:end-1);
T1(:,1) = x-x+1;  % T_0
T1(:,2) = x;      % T_1
for ii = 2:10
    T1(:,ii+1) = 2*x.*T1(:,ii) - T1(:,ii-1); % T_i
end
% Chebyshev coefficients
weight = 1./sqrt( 1 - x.*x );
for ii=1:11
    c1(ii) = 2/pi*fx'*(weight.*T1(:,ii))*deltax; % c_i-1
end
c = c1';
end
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 ChebyshevErrorAnalysis101.m을 실행하면, 그림 2.4.3이 그려진다. 그림 2.4.3에서 알 수 있듯이, C^∞급인 함수 $f_1(x) = \exp(x+1)$의 Chebyshev계수 c_i는 차수 i가 커짐에 급격히 0에 가까이 간다. 반면에, C^2급인 함수 $f_3(x) = \max\{x^3, 0\}$의 Chebyshev계수 c_i는 차수 i가 커짐에 천천히 0에 가까이 간다. 또한, 구간 $(0,1]$에서 C^∞급인 함수 $f_2(x) = [x+1]^{1/4}$의 Chebyshev계수 c_i는 함수 $f_1(x)$과 함수 $f_3(x)$의 중간 형태를 보인다. 함수 $f_1(x)$와 함수 $f_2(x)$의 도함수들은 $x = -1$에서 ∞이다. 따라서, 이 함수들에 대해서는 명제 2.4.5를 바탕으로한 Chebyshev근사식의 오차분석은 의미가 없다. ■

예제 2.4.7 Python을 사용해서 예제 2.4.6을 다시 다루기 위해서, 다음 Python프로그램 ChebyshevErrorAnalysis101.Py를 실행해 보자.

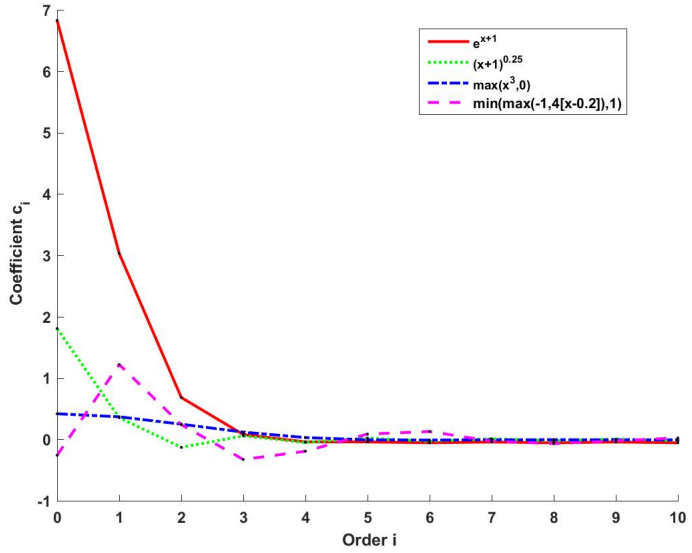


그림 2.4.3. Chebyshev 오차

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

def ChebyshevCoeff(xx,fxx):
    # x = column vector of x
    # fx = column vector of y
    # Chebyshev polynomials
    xxDim = len(xx);   print(xxDim);
    deltax = xx[1] - xx[0];
    x = xx[1:xxDim-1]; fx = fxx[1:xxDim-1];
    xDim = len(x); print(xDim)
    T = np.zeros((xDim,11),float);
    T[:,0] = x-x+1;  # T_0
    T[:,1] = x;      # T_1
    for ii in range(2,11):
        T[:,ii] = 2*x*T[:,ii-1] - T[:,ii-2];
    print(T)
    dum1 = 1/np.sqrt( 1 - x*x)*np.transpose(fx);
    c =  2/np.pi*deltax*np.matmul(dum1, T)
    return c

# Chebyshev Error
x = np.linspace(-1,1,10001)
xTrans = np.transpose(x)
ii = np.arange(0,11,1)
print(ii)
# case 1
f1 = np.exp(x+1);
coe1 = ChebyshevCoeff(x,f1);
print(coe1)
# case 2
```

```
f2 = (x+1)**0.25;
coe2 = ChebyshevCoeff(x,f2);
print(coe2)
# case 3
f3 = np.maximum( 0, x**3);
coe3 = ChebyshevCoeff(x,f3);
print(coe3)
# case 4
f4 = np.minimum( np.maximum( -1, 4*(x - 0.2) ), 1);
coe4 = ChebyshevCoeff(x,f4);
print(coe4)

# Plotting
fig = plt.figure()
plt.plot(ii,coe1,'r-',lw=2, label='e^{x+1}')
plt.plot(ii,coe2,'g:',lw=2, label='(x+1)^{0.25}')
plt.plot(ii,coe3,'b-.',lw=2, label='max(x^{3},0)')
plt.plot(ii,coe4,'m--',lw=2, label='min(max(-1,4[x-0.2]),1)')
plt.xlabel('Order i'); plt.ylabel('Coefficient c_i')
plt.legend(loc='upper right', numpoints=1)
plt.axis([ 0, 10, -1, 7 ])
plt.show()
fig.savefig('ChebyshevErrorAnalysis101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 2.4.6의 결과와 같다. ■

2.4.4 Chebyshev절약

만약 함수 $q(x)$가 m차 멱함수이면, 다음과 같은 함수 $q(x)$의 Chebyshev표현이 존재한다.

$$q(x) = \sum_{k=0}^{m} c_k T_k(x) \tag{2.4.3}$$

명제 2.3.1에서 알 수 있듯이, Chebyshev계수는 다음과 같다.

$$c_k = \frac{2}{\pi} \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} T_k(x) f(x) dx, \quad (k = 0, 1, \cdots) \tag{2.4.4}$$

함수 $q(x)$는 멱함수이므로 이 Chebyshev표현을 쉽게 구할 수 있다. 앞에서도 설명했듯이, 각 $n(< m)$에 대해서 함수 $q(x)$의 n차 최소제곱근사식은 다음과 같다.

$$C_n(x) = \sum_{k=0}^{n} c_k T_k(x) \tag{2.4.5}$$

이 $C_n(x)$를 함수 $q(x)$의 n차 Chebyshev절약(Chebyshev economization)이라 부른다. 이러한 과정을 거쳐서 높은 차수의 멱함수를 낮은 차수의 멱함수로 근사시킬 수 있다.

예제 2.4.8 구간 $[-1,1]$에서 함수 $f(x)=\sin x$를 3차 멱함수로 근사시키자. 함수 $f(x)$의 점 $x=0$에서 Taylor근사식을 구하면, 다음과 같다.

$$f(x) \approx f_3(x) \doteq x - \frac{1}{6}x^3 + \frac{1}{120}x^5 \tag{1}$$

함수 $f_3(x)$를 Chebyshev함수들로 표기하면, 다음과 같다.

$$\begin{aligned} f_3(x) &= T_1(x) - \frac{1}{24}[3T_1(x)+T_3(x)] + \frac{1}{1920}[10T_1(x)+5T_3(x)+T_5(x)] \\ &= \frac{169}{192}T_1(x) - \frac{5}{128}T_3(x) + \frac{1}{1920}T_5(x) \end{aligned} \tag{2}$$

따라서, 함수 $f(x)$의 3차 Chebyshev절약은 다음과 같다.

$$f_3(x) \approx C_3(x) = \frac{169}{192}T_1(x) - \frac{5}{128}T_3(x) \approx 0.9974x - 0.1562x^3 \tag{3}$$

Taylor근사식에 의한 오차와 Chebyshev근사식에 의한 오차를 계산하기 위해서, 다음 MATLAB프로그램 ChebyshevEconomization101.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename ChebyshevEconomization101.m
% Chebyshev Economization
% Programmed by CBS
% ------------------------------------------------------------------
clear, clf
x = [-1:0.0001:1]';
fx = sin(x);
% Taylor approximation
taylorx = x.*( 1 - 1/6*x.^2.*(1 - 1/120*x.^2 ) );
errorT = fx -taylorx;
% Chebyshev approximation
T1(:,1) = x-x+1;  % T_0
T1(:,2) = x;      % T_1
for ii = 2:10
    T1(:,ii+1) = 2*x.*T1(:,ii) - T1(:,ii-1); % T_ii
end
chebyshevx = 169/192*T1(:,2) - 5/128*T1(:,4);
errorC = fx - chebyshevx;
% plot
plot(x,errorT,'k--',x,errorC,'r-','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
legend('Taylor error','Chebyshev error','location','SE')
hold on
plot([-1 1],[0 0],'g:','linewidth',2)
hold off
```

```
xlabel('x','fontsize',12)
ylabel('error','fontsize',13)
saveas(gcf,'ChebyshevEconomization101','jpg')
save('ChebyshevEconomization101.txt','chebyshevx','errorC')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 그림 2.4.4가 그려진다. 그림 2.4.4에서 알 수 있듯이, Taylor근사식은 점 $x = 0$에서 오차가 거의 없고 이 점에서 멀어질수록 오차의 절대값이 커진다. 반면에, Chebyshev절약에 의한 오차는 등진동성을 보인다. ■

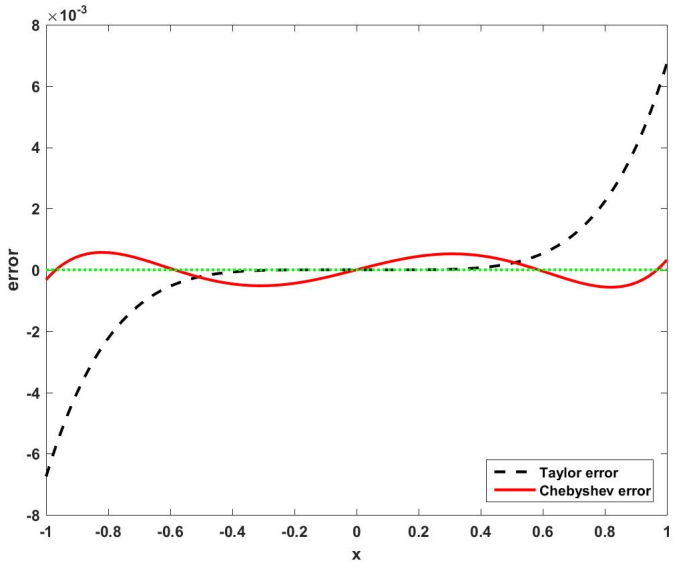


그림 2.4.4. Chebyshev절약에 의한 오차

예제 2.4.9 Python을 사용해서 예제 2.4.8을 다시 다루기 위해서, 다음 Python프로그램 ChebyshevEconomization101.Py를 실행해 보자.

```
#Python v3.6 작성작성자
# 조두현 경제학부 박사과정

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1,1,10001)
xDim = len(x)
fx = np.sin(x)테일러

# 전개한 식에 대입값과 실제 함수와의 오차를 구분
taylorxx = x*(1-1/6*x**2*(1-1/120*x**2))
errorT = fx - taylorxx

#chebyshev 절약
T = np.zeros((11,xDim),float)
```

```
T[0,:] = x -x+1
T[1,:] = x
for ii in range(2,11):
    T[ii,:]= 2*x*T[ii -1,:] -T[ii-2,:]
chebyshevx = 169/192*T[1,:]-5/128*T[3,:]
errorC = fx - chebyshevx

print(errorC)
#plotting show()
fig = plt.figure()
plt.plot(x,errorT,'k-',lw=2,label='TaylorError')
plt.plot(x,errorC,'r-',lw=2,label='ChebyshevError')
plt.xlabel('x')
plt.ylabel('Error')
plt.legend(loc='lower right', numpoints =1)
plt.plot(x,x-x,',g:',lw=2)

plt.axis([-1,1,-8*10**(-3),8*10**(-3)])
fig.savefig('ChebyshevEconomization.png')
```

이 Python프로그램을 실행한 결과는 예제 2.4.8의 결과와 같다. ■

제 3 장

보간

이 절에서는 기간구조모형이나 변동성곡면을 추정하는 데 자주 사용되는 보간법(interpolation method)에 대해 살펴보기로 하자. 보간법이란 주어진 유한개 조건들을 만족하는 '좋은(nice)' 함수를 구축하는 것이다. 이 절에서는 이 '좋은' 함수를 멱함수로 국한하기로 하자. 물론, 멱함수 이외에도 삼각함수 등 다양한 함수를 '좋은' 함수로 간주할 수 있다. 심지어는 전혀 평활하지 않은 Daubechies함수와 같은 웨이블릿함수(wavelet function)를 사용해서 보간을 할 수도 있다. 보간법을 사용하는 예로서 사분기(quaterly) 데이터를 월별(monthly) 데이터로 확장하는 것을 생각해볼 수 있다.

제3.1절 Lagrange보간

다음 식들을 만족하는 표본점들 $x_1, x_2, \cdots, x_n$을 살펴보자.

$$x_1 < x_2 < \cdots < x_n \tag{3.1.1}$$

함수 f에 대해서 다음 값들을 정의하자.

$$f_i \doteq f(x_i), \quad (i = 1, 2, \cdots, n) \tag{3.1.2}$$

함수 f를 점들 $D \doteq \{(x_i, f_i) \mid i = 1, 2, \cdots, n\}$을 지나는 적당한 함수 p_n으로 대용하는 것을 보간이라 한다. 이 함수 p_n은 다음 식들을 만족한다.

$$p_n(x_i) = f_i, \quad (i = 1, 2, \cdots, n) \tag{3.1.3}$$

또한, $x_1, x_2, \cdots, x_n$을 마디점들(interpolation knots)이라 한다.

만약 보간식 $p_n(x)$가 멱함수이면, $p_n(x)$는 최대 $[n-1]$차 멱함수이다. 다음과 같은 함수들을 정의하자.

$$l_i(x) \doteq \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}, \quad (i = 1, 2, \cdots, n) \tag{3.1.4}$$

이 함수들이 다음 식들을 만족함은 명백하다.

$$l_i(x_k) = \delta_{i,k}, \quad (i, k = 1, 2, \cdots, n) \tag{3.1.5}$$

다음 함수를 정의하자.

$$p_n(x) \doteq \sum_{i=1}^{n} f_i \, l_i(x) \tag{3.1.6}$$

식 (3.1.5)에서 알 수 있듯이, 함수 $p_n(x)$는 식 (3.1.3)을 만족한다. 이 $l_i(x)$를 Lagrange 보간함수 그리고 $p_n(x)$를 Lagrange보간식이라 부른다. 역으로, 식 (3.1.3)과 식 (3.1.6)을 만족하는 최대 $[n-1]$차 멱함수들 $l_1, l_2, \cdots, l_n$은 식 (3.1.4)를 만족한다.

예제 3.1.1 점들 $D = \{(0,6), (1,0), (2,2)\}$을 지나는 Lagrange보간식을 구해보자.

Lagrange보간함수들은 다음과 같다.

$$l_1(x) = \frac{[x-1][x-2]}{[0-1][0-2]} = \frac{x^2 - 3x + 2}{2} \tag{1}$$

$$l_2(x) = \frac{[x-0][x-2]}{[1-0][1-2]} = \frac{x^2 - 2x}{-1} \tag{2}$$

$$l_3(x) = \frac{[x-0][x-1]}{[2-0][2-1]} = \frac{x^2 - x}{2} \tag{3}$$

따라서, Lagrange보간식 $p_2(x)$는 다음과 같다.

$$p_2(x) = 6 \cdot l_1(x) + 0 \cdot l_2(x) + 2 \cdot l_3(x) = 4x^2 - 10x + 6 \tag{4}$$

■

Lagrange보간식은 이론적으로 간단해도 실용적이지 않다. 그 이유는 보간식을 구하는 목적은 마디점이 아닌 점 $x(\neq x_i)$에서 보간함수값 $p_n(x)$를 계산하는 것인데, 식 (3.1.6)을 이용해서 보간함수값 $p_n(x)$을 계산하는데 많은 연산을 필요로 하기 때문이다. 식 (3.1.6)을 사용해서, 각 x에 대한 보간함수값 $p_n(x)$를 계산하는 데는 곱셈 $3n$번, 나눗셈 n번, 그리고

덧셈 $[n-1]$ 번을 필요로 한다. 따라서, 보간식 (3.1.6)을 직접 사용하는 것은 계산효율적이 아니다.

Lagrange보간식을 효율적으로 사용하기 위해서, Lagrange보간식 (3.1.6)을 다음과 같이 표기하자.

$$p_n(x) = \sum_{j=0}^{n-1} a_j x^j \tag{3.1.7}$$

Horner형식을 사용해서 $[n-1]$ 차 멱함수의 함수값을 계산하는 데는 단지 곱셈 $[n-1]$ 번과 덧셈 $[n-2]$ 번을 필요로 한다. 따라서, 식 (3.1.7)의 계수들 $\{a_j\}$ 를 먼저 구하고, 다음 단계로 Horner형식을 적용하기로 하자.

다음 식들이 성립한다.

$$a_0 + a_1 x_i^1 + a_2 x_i^2 + \cdots + a_{n-1} x_i^{n-1} = f_i, \quad (i = 1, 2, \cdots, n) \tag{3.1.8}$$

다음과 같이 Vandermonde행렬 V 와 벡터들을 정의하자.

$$V \doteq \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix}, \quad \boldsymbol{a} \doteq \begin{bmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{bmatrix}, \quad \boldsymbol{y} \doteq \begin{bmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{bmatrix} \tag{3.1.9}$$

식 (3.1.8)을 다음과 같이 쓸 수 있다.

$$V\boldsymbol{a} = \boldsymbol{y} \tag{3.1.10}$$

다음 식이 성립함을 쉽게 증명할 수 있다.

$$\det(V) = \prod_{1 \le i < j \le n} [x_i - x_j] \tag{3.1.11}$$

만약 $x_1, x_2, \cdots, x_n$ 이 식 (3.1.1)을 만족하면, 식 (3.1.11)에서 알 수 있듯이 $\det(V)$ 는 0이 아니다. 따라서, 연립방정식 (3.1.10)은 일의적인 해를 갖는다. 즉, 다음 명제가 성립한다.

명제 3.1.1

만약 $x_1, x_2, \cdots, x_n$ 이 서로 다르면, Lagrange보간을 위한 방정식 (3.1.10) 의 해 $\{a_0, a_1, \cdots, a_{n-1}\}$ 이 일의적으로 존재한다.

증명. 만약 $[n-1]$ 차 이하인 멱함수들 $f^{(1)}$ 과 $f^{(2)}$ 가 식 (3.1.8) 을 만족하면, $[n-1]$ 차 이하인 멱함수 $p \doteq f^{(1)} - f^{(2)}$ 는 다음 식들을 만족한다.

$$p(x_j) = f^{(1)}(x_j) - f^{(2)}(x_j) = 0, \quad (j = 1, 2, \cdots, n) \tag{1}$$

즉, $[n-1]$ 차 이하의 멱함수 p 가 서로 다른 n 개의 근들 $x_1, x_2, \cdots, x_n$ 을 갖는다. 따라서, 항등식원리에 의해 다음 식이 성립한다.

$$p(x) \equiv 0, \quad (x \in R) \tag{2}$$

즉, $f^{(1)} = f^{(2)}$ 이다. 따라서, 일의성이 성립한다. ■

예제 3.1.2 식 (3.1.10) 를 사용해서, 다음 점들을 지나는 Lagrange보간식을 구해보자.

$$D = \{(-3, 169), (-1, 5), (1, -7), (2, -1), (3, 37)\} \tag{1}$$

이 Lagrange보간식을 구하기 위해서, 다음 MATLAB프로그램 LagrangeInter101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: LagrangeInter101.m
% Lagrange interpolation polynomial I
% Programmed by CBS
% ------------------------------------------------------------------------
function LagrangeInter101
% Calculate Lagrange coefficients
x = [ -3 -1 1 2 3 ];
y = [ 169 5 -7 -1 37 ];
% Evaluate Lagrange interpolation doefficient
[V,condi] = MakeVandermonde101(x)
a1 = inv(V)*y'
z = [-3.5:0.01:3.5];
ff =  a1(1) + z.*( a1(2) + z.*( a1(3) + z.*( a1(4) + a1(5)*z))); % Horner
% original funtion
fz = -5 + z.*( -4 + z.*(3 + z.*(-2 + z)));
err = max(abs(fz-ff))
% Plotting
plot(z,fz,'g-',z,ff,'k--',x,y,'rd','LineWidth',2.0)
legend('data point','function','Lagrange interpolation',1)
set(gca,'fontsize',11,'fontweigh','bold')
xlim([-3.5 3.5])
xlabel('\bf x','fontsize',12)
```

```
ylabel('y','fontsize',12,'rotation',0)
saveas(gcf,'LagrangeInter101.jpg')
save('LagrangeInter101.txt','z','fz','ff','-ascii')
end
% ------------------------------------------------------------------------
function [V,condi] = MakeVandermonde101(x);
% x = column vector
% V = Vandermonde matrix
n = length(x);
persym = zeros(n,n);
for dumi = 1:n
    persym(dumi,n+1-dumi) = 1;
end
Vdum = vander(x);
V = Vdum*persym;
condi = cond(V);
end
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 LagrangeInter101.m을 실행하면, 집합 D의 점들을 지나는 4차 Lagrange보간식이 다음과 같음을 알 수 있다.

$$p_4(x) = -5.000000 + x[-4.000000 + x\{3.000000 + x[-2.000000 + 1.000000x]\}] \qquad (2)$$

원래 집합 D는 다음 함수에서 추출한 관찰점들이다.

$$f(x) = x^4 - 2x^3 + 3x^2 - 4x - 5 \qquad (3)$$

구간 $[-3.5, 3.5]$에서 함수 $f(x)$와 Lagrange보간식 사이의 최대오차 $\max|f(x) - p_4(x)|$는 $3.4 \cdot 10^{-13}$이다. 또한, 이 MATLAB프로그램을 실행하면, 집합 D, 함수 $f(x)$, 그리고 Lagrange보간식 $p_4(x)$의 그림 3.1.1을 출력한다. 그림 3.1.1에서 확인할 수 있듯이, 함수 $f(x)$와 Lagrange보간식 $p_4(x)$는 구간 $[-3.4, 3.5]$에서 정확히 일치한다. 또한, 집합 D의 점들은 이 두 곡선들 위에 존재함을 확인할 수 있다. ■

예제 3.1.3 Python을 사용해서 예제 3.1.2을 다시 다루기 위해서, 다음 Python프로그램 LagrangeInter101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
```

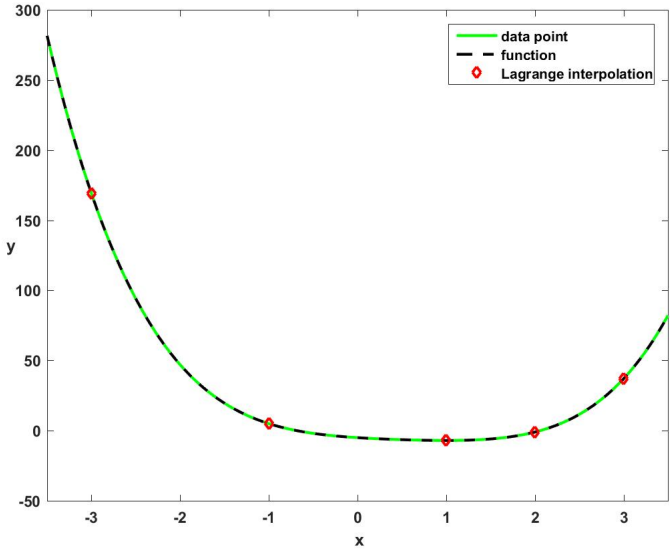


그림 3.1.1. Lagrange보간식 I

```
8
9  def MakeVandermonde101(x):
10     # x = column vector
11     # V = Vandermonde matrix
12     n = len(x)
13     persym = np.zeros((n,n),float)
14     for dumi in range(1,n+1):
15         persym[dumi-1,n-dumi] = 1;
16     Vdum = np.vander(x, n)
17     V = np.matmul(Vdum,persym);
18     condi = np.linalg.cond(V);
19     return V, condi
20
21 # Calculate Lagrange coefficients
22 x = np.array([ -3, -1, 1, 2, 3 ]);
23 y = np.array([ 169, 5, -7, -1, 37 ]);
24 # Evaluate Lagrange interpolation doefficient
25 [V,condi] = MakeVandermonde101(x);
26 a = np.matmul(np.linalg.inv(V),np.transpose(y));
27 z = np.arange(-3.5,3.51,0.01);
28 ff =  a[0] + z*( a[1] + z*( a[2] + z*( a[3] + a[4]*z))); # Honer
29 # original funtion
30 fz = -5 + z*( -4 + z*(3 + z*(-2 + z)));
31 err = max(np.abs(fz-ff))
32
33 # Plotting
34 fig = plt.figure()
35 plt.plot(z,fz,'g-',lw=2, label='data point')
36 plt.plot(z,ff,'k--',lw=2, label='function')
37 plt.plot(x,y,'rd',lw=2, label='Lagrange interpolation')
38 plt.xlabel('x'); plt.ylabel('y')
39 plt.legend(loc='upper right', numpoints=1)
40 plt.axis([ -3.5, 3.5, -50, 300 ])
41 plt.show()
42 fig.savefig('LagrangeInter101Py.png')
43
44 # End of Program
```

이 Python프로그램을 실행한 결과는 예제 3.1.2의 결과와 같다. ■

연립방정식 (3.1.10)의 해가 안정적이기 위해서는 다음과 같이 정의되는 조건수(condition number) $\mathrm{cond}(V)$가 작아야 한다.

$$\mathrm{cond}(V) \doteq ||V|| \cdot ||V^{-1}|| \tag{3.1.12}$$

여기서 $||A||$는 행렬 A의 노름(norm)이다. 행렬의 노름을 다양하게 정의할 수 있다. 그 중에서 스펙트럴노름(spectral norm)을 사용하면, 다음 식이 성립한다.

$$\mathrm{cond}(A) = \frac{|\lambda_1|}{|\lambda_n|} \tag{3.1.13}$$

여기서 $\lambda_1 > \lambda_2 > \cdots > \lambda_n$은 행렬 A의 고유값들이다. Vandermonde행렬 V의 조건수가 크므로, 연립방정식 (3.1.10)의 해는 안정적이 아니다.

예제 3.1.4 Vandermonde행렬의 조건수를 구하기 위해서, 다음 MATLAB프로그램 VandermondeCondiNo101.m을 실행하자.

```
% ------------------------------------------------------------------
% Filename: VandermondeCondiNo101.m
% Condition Number of Vandermonde matrix
% Programmed by CBS
% ------------------------------------------------------------------
function VandermondeCondiNo101
for dumi=1:9
    x = [1:dumi]';
    [V,condiNO] = MakeVandermonde(x);
    dumi,condiNO
end
save('VandermondeCondiNo101.txt','dumi','condiNO','-ascii')
end
% End of program
% ------------------------------------------------------------------
function [V,condi] = MakeVandermonde(x);
% x = column vector
% V = Vandermonde matrix
n = length(x);
persym = zeros(n,n);
for dumi = 1:n
    persym(dumi,n+1-dumi) = 1;
end
Vdum = vander(x);
V = Vdum*persym;
condi = cond(V);
end
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 VandermondeCondiNo101.m을 실행하면, 다음 Vandermonde행렬들의 조건수들을 출력한다.

$$V_n \doteq \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & 2 & 2^2 & \cdots & 2^{n-1} \\ 1 & 3 & 3^2 & \cdots & 3^{n-1} \\ 1 & 4 & 4^2 & \cdots & 4^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & n & n^2 & \cdots & n^{n-1} \end{bmatrix}, \quad (n = 1, 2, \cdots, 9) \tag{1}$$

출력된 조건수들이 다음 표에 수록되어 있다. 이 표에서 알 수 있듯이, 조건수는 지수적으로 증가한다.

n	조건수	n	조건수
1	1.00	6	7.31e+005
2	6.85	7	2.45e+007
3	70.92	8	9.52e+008
4	1.17e+003	9	4.23e+010
5	2.62e+004		

■

예제 3.1.5 Python을 사용해서 예제 3.1.4을 다시 다루기 위해서, 다음 Python프로그램 VandermondeCondiNo101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

def MakeVandermonde101(x):
    # x = column vector
    # V = Vandermonde matrix
    n = len(x)
    persym = np.zeros((n,n),float)
    for dumi in range(1,n+1):
        persym[dumi-1,n-dumi] = 1;
    Vdum = np.vander(x, n)
    V = np.matmul(Vdum,persym);
```

```
    condi = np.linalg.cond(V);
    return V, condi

# Condition Number of Vandermonde matrix
for dumi in range(1,10):
    x = np.arange(1,dumi+1,1);
    [V,condiNO] = MakeVandermonde101(x);
    print(dumi, condiNO)

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 3.1.4의 결과와 같다. ■

다음 함수를 정의하자.

$$F_n(x) \doteq \prod_{i=1}^{n}[x - x_i] \tag{3.1.14}$$

식 (3.1.14)에 로그미분법을 적용하면, 다음 식을 얻을 수 있다.

$$\frac{F_n'(x)}{F_n(x)} = \sum_{k=1}^{n}\frac{1}{x - x_k} \tag{3.1.15}$$

즉, 다음 식들이 성립한다.

$$F_n'(x) = \sum_{k=1}^{n}\prod_{j\neq k}[x - x_j] \tag{3.1.16}$$

따라서 다음 식들이 성립한다.

$$\frac{1}{F_n'(x_i)} = \prod_{j\neq i}\frac{1}{x_i - x_j}, \quad (i = 1, 2, \cdots, n) \tag{3.1.17}$$

식 (3.1.4)과 식 (3.1.17)에서 알 수 있듯이, 다음 식들이 성립한다.

$$l_i(x) = \frac{F_n(x)}{[x - x_i]F_n'(x_i)}, \quad (i = 1, 2, \cdots, n) \tag{3.1.18}$$

식 (3.1.18)을 식 (3.1.6)에 대입하면, 다음 식을 얻는다.

$$p_n(x) = \sum_{i=1}^{n} c_i \frac{F_n(x)}{[x - x_i]} \tag{3.1.19}$$

여기서 c_i는 다음과 같다.

$$c_i \doteq \frac{f_i}{F_n'(x_i)} \tag{3.1.20}$$

예제 3.1.6 식 (3.1.19)을 사용해서, 다음 점들을 지나는 Lagrange보간식을 구해보자.

$$D = \{(-3, 169), (-1, 5), (1, -7), (2, -1), (3, 37)\} \tag{1}$$

이 Lagrange보간식을 구하기 위해서, 다음 MATLAB프로그램 LagrangeInter102.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename: LagrangeInter102.m
% Lagrange interpolation polynomial II at x = z
% Programmed by CBS
% ------------------------------------------------------------------
function LagrangeInter102
clear all, clf, format long
% Calculate Lagrange coefficients
x = [ -3 -1 1 2 3 ];
y = [ 169 5 -7 -1 37 ];
c = LagrangeCoefficient(x,y)
% Evaluate Lagrange interpolation at x = z
z = [-3.5:0.01:3.5];
for k = 1:length(z)
    LI(k) = 0; % Lagrange interploation polynomial
    for dumj = 1:length(x)
        nume(dumj) = 1;
        for dumk = 1:length(x)
            if dumj ~= dumk
                nume(dumj) = nume(dumj)*(z(k) - x(dumk));
            end
        end
        LI(k) = LI(k) + nume(dumj)*c(dumj);
    end
end
% original funtion
fz = -5 + z.*( -4 + z.*(3 + z.*(-2 + z)));
err = max(abs(fz-LI))
% plot
plot(x,y,'rd',z,fz,'g-',z,LI,'k--','LineWidth',2.5)
legend('data point','function','Lagrange interpolation',1)
set(gca,'fontsize',11,'fontweigh','bold')
xlim([-3.5 3.5])
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12)
saveas(gcf,'LagrangeInter102.jpg')
save('LagrangeInter102.txt','fz','err','-ascii')
end
% End of program
% ------------------------------------------------------------------
function coeff = LagrangeCoefficient(x,ff)
n = length(x);
for i=1:n
    deno(i) = 1;
    for j=1:n
        if j ~= i
            deno(i) = deno(i)*(x(i)-x(j));
        end
        coeff(i) = ff(i)/deno(i);
    end
```

```
end
end
% End of program
% --------------------------------------------------------------
```

MATLAB프로그램 LagrangeInter102.m을 실행하면, 집합 D의 점들을 지나는 4차 Legendre보간계수들을 출력한다. 이 계수들은 다음과 같다.

$$c_1 = 0.704167, \quad c_2 = -0.104167, \quad c_3 = -0.437500 \tag{1}$$

$$c_4 = 0.066667, \quad c_5 = 0.770833 \tag{2}$$

즉, Lagrange보간식 $p_4(x)$는 다음과 같다.

$$\begin{aligned} p_4(x) = {} & 0.704167[x+1][x-1][x-2][x-3] - 0.104167[x+3][x-1][x-2][x-3] \\ & - 0.437500[x+3][x+1][x-2][x-3] + 0.066667[x+3][x+1][x-1][x-3] \\ & + 0.770833[x+3][x+1][x-1][x-2] \end{aligned} \tag{3}$$

원래 집합 D는 다음 함수에서 추출한 관찰점들이다.

$$f(x) = x^4 - 2x^3 + 3x^2 - 4x - 5 \tag{4}$$

구간 $[-3.5, 3.5]$에서 함수 $f(x)$와 Lagrange보간식 사이의 최대오차 $\max|f(x) - p_4(x)|$는 $1.7 \cdot 10^{-13}$이다. 또한, 이 MATLAB프로그램을 실행하면, 집합 D, 함수 $f(x)$, 그리고 Lagrange보간식 $p_4(x)$의 그림 3.1.2가 출력된다. 그림 3.1.2에서 확인할 수 있듯이, 함수 $f(x)$와 Lagrange보간식 $p_4(x)$는 구간 $[-3.5, 3.5]$에서 정확히 일치한다. 또한, 집합 D의 점들은 이 두 곡선들 위에 존재함을 알 수 있다. ■

예제 3.1.7 Python을 사용해서 예제 3.1.6을 다시 다루기 위해서, 다음 Python프로그램 LagrangeInter102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

```

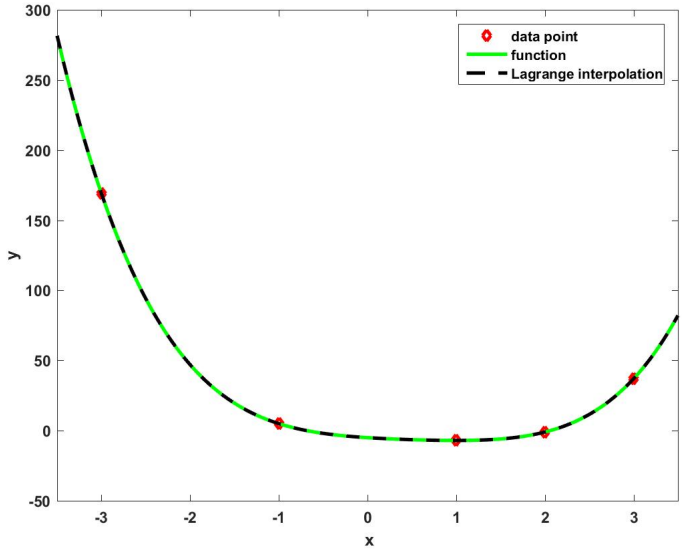


그림 3.1.2. Lagrange보간식 II

```
def LagrangeCoefficient(x,ff):
    n = len(x);
    deno = [None]*n;
    coeff = [None]*n;
    for ii in range(0,n):
        deno[ii] = 1;
        for jj in range(n):
            if jj != ii:
               deno[ii] = deno[ii]*(x[ii]-x[jj]);
        coeff[ii] = ff[ii]/deno[ii];
    return coeff

# Calculate Lagrange coefficients
x = np.array([ -3, -1, 1, 2, 3 ]);
xlength = len(x);
y = np.array([ 169, 5, -7, -1, 37 ]);
c = LagrangeCoefficient(x,y);
print(c)

# Evaluate Lagrange interpolation at x = z
z = np.arange(-3.5,3.51,0.01);
zlength = len(z);
LI = [0]*zlength;
nume = [0]*xlength;
for kk in range(0,zlength):
    LI[kk] = 0;      # Lagrange interploation polynomial
    for dumj in range(0,xlength):
        nume[dumj] = 1;
        for dumk in range(0,xlength):
            if dumj != dumk:
                nume[dumj] = nume[dumj]*(z[kk] - x[dumk]);
        LI[kk] = LI[kk] + nume[dumj]*c[dumj];

# original funtion
fz = -5 + z*( -4 + z*(3 + z*(-2 + z)));
err = max(np.abs(fz-LI))

```

```
# Plotting
fig = plt.figure()
plt.plot(x,y,'rd',lw=2, label='data point')
plt.plot(z,fz,'g-',lw=2, label='function')
plt.plot(z,LI,'k--',lw=2, label='Lagrange interpolation')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([ -3.5, 3.5, -50, 300 ])
plt.show()
fig.savefig('LagrangeInter102Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 3.1.6의 결과와 같다. ■

예제 3.1.8 예제 3.1.6를 다시 살펴보자. 특성다항식과 합성곱(convolution)을 사용하면, 이 문제를 좀 더 쉽게 풀 수 있다.

다음 함수들을 정의하자.

$$l_{i,j}(x) \doteq \prod_{k=1,k\neq i}^{j} \frac{x-x_k}{x_i-x_k}, \quad (j=1,2,\cdots,n) \tag{1}$$

다음 식들이 성립한다.

$$l_{i,0}(x) \doteq 1, \quad l_{i,j}(x) = \frac{1}{x_i-x_j} l_{i,j-1}(x)[x-x_j], \quad (j \neq i) \tag{2}$$

여기서 $l_{i,j-1}(x)[x-x_j]$는 멱함수 $l_{i,j-1}(x)$와 멱함수 $[x-x_j]$의 곱임을 유의하라. 따라서 이에 해당하는 계수는 합성곱이다.

식 (2)를 사용해서, Lagrange보간식을 구하기 위해서, 다음 MATLAB프로그램 LagrangeInter103.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: LagrangeInter103.m
% Lagrange interpolation polynomial III w/ convolution
% Programmed by CBS
% ------------------------------------------------------------------------
x = [ -3 -1 1 2 3 ];
y = [ 169 5 -7 -1 37 ];
n = length(x);
LL = zeros(n,n);
% Calculate Lagrange coefficient polynomials
for i = 1:n
    Ldum= 1;
    for j = 1:n
        if j ~= i
            Ldum = 1/( x(i)-x(j) )*conv( Ldum,poly( x(j)) );
        end
```

```
    end
    LL(i,:) = Ldum;
end
% Calculate Lagranage interpolation coefficient
aa1 = y*LL;
persym = zeros(n,n);
for dumi = 1:n
    persym(dumi,n+1-dumi) = 1;
end
a1 = aa1*persym
% Calculate Lagranage interpolation
z = [-3.5:0.01:3.5];
ff =  a1(1) + z.*( a1(2) + z.*( a1(3) + z.*( a1(4) + a1(5)*z)));
% original funtion
fz = -5 + z.*( -4 + z.*(3 + z.*(-2 + z)));
err = max(abs(fz-ff))
% Plotting
plot(x,y,'rd',z,fz,'g-',z,ff,'k--','LineWidth',2)
legend('data point','function','Lagrange interpolation',1)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-3.5 3.5])
xlabel('\bf x','fontsize',12), ylabel('y','fontsize',12)
saveas(gcf,'LagrangeInter103.jpg')
save('LagrangeInter103.txt','fz','err','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 LagrangeInter103.m에서는 MATLAB함수 poly를 사용했다. 이 poly는 어떤 정방행렬 A의 특성함수(characteristic function) $f_A(x) = \det(xI - A)$를 구하기 위한 것이다. 즉, MATLAB명령문 'poly(A)'는 이 특성함수의 계수들을 내림차순으로 출력한다. 예를 들어, 다음과 같은 MATLAB출력한 결과를 살펴보자.

```
>> A = [ 1 2 ; 3 4 ], pa = poly(A)
```

이 MATLAB명령문을 실행하면, 다음 식들이 성립함을 확인할 수 있다.

$$f_A(x) = \det(xI - A) = x^2 - 5x - 2 \tag{3}$$

특히 스칼라 a에 대해서 다음 식들이 성립한다.

$$f_a(x) = \det(x - a) = x - a \tag{4}$$

이 MATLAB프로그램에서는 식 (4)와 더불어 MATLAB함수 conv.m을 식 (2)에 적용해서, Lagrange보간함수들 $l_1, l_2, \cdots, l_n$을 구하였다.

MATLAB프로그램 LagrangeInter103.m을 실행하면, 집합 D의 점들을 지나는 4차 Leg-

endre보간식이 다음과 같음을 알 수 있다.

$$p_4(x) = -5.000000 + x[-4.000000 + x\{3.000000 + x[-2.000000 + 1.000000x]\}] \quad (5)$$

원래 집합 D는 다음 함수에서 추출한 관찰점들이다.

$$f(x) = x^4 - 2x^3 + 3x^2 - 4x - 5 \quad (6)$$

구간 $[-3.5, 3.5]$에서 함수 $f(x)$와 Lagrange보간식 사이의 최대오차 $\max|f(x) - p_4(x)|$는 1.1×10^{-13}이다. 또한, 이 MATLAB프로그램을 실행하면, 집합 D, 함수 $f(x)$, 그리고 Lagrange보간식 $p_4(x)$의 그림 3.1.3을 출력한다. 그림 3.1.3은 그림 3.1.2와 같다. 즉, 함수 $f(x)$와 Lagrange보간식 $p_4(x)$는 구간 $[-3.5, 3.5]$에서 정확히 일치한다. 또한, 집합 D의 점들은 이 두 곡선들 위에 존재한다. ■

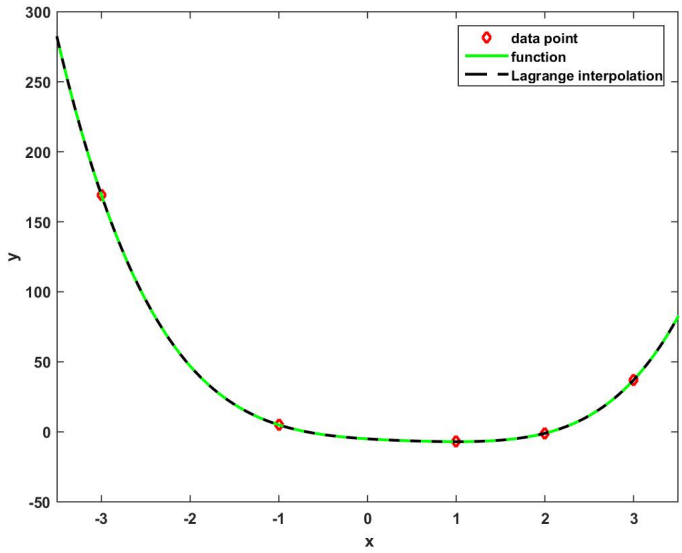


그림 3.1.3. Lagrange보간식 III

예제 3.1.9 Python을 사용해서 예제 3.1.8을 다시 다루기 위해서, 다음 Python프로그램 LagrangeInter103.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

```

```
import numpy as np
import matplotlib.pyplot as plt

# Calculate Lagrange coefficients
x = np.array([ -3, -1, 1, 2, 3 ]);
y = np.array([ 169, 5, -7, -1, 37 ]);
n = len(x);
LL = np.zeros((n,n),float)
# Calculate Lagrange coefficient polynomials
for ii in range(0,n):
    Ldum= 1;
    for jj in range(0,n):
        if jj != ii:

            Ldum = 1/( x[ii]-x[jj] )*np.convolve( Ldum, [1, -x[jj]] );
    LL[ii,:] = Ldum;

# Calculate Lagranage interpolation coefficient
aa = np.matmul(y,LL);
persym = np.zeros((n,n),float);
for dumi in range(0,n):
    persym[dumi,n-dumi-1] = 1;
a = np.matmul(aa,persym)

# Calculate Lagranage interpolation
z = np.arange(-3.5, 3.51, 0.01);
ff =  a[0]+ z*( a[1] + z*( a[2] + z*( a[3] + a[4]*z)));
# original funtion
fz = -5 + z*( -4 + z*(3 + z*(-2 + z)));
err = max(np.abs(fz-ff))

# Plotting
fig = plt.figure()
plt.plot(x,y,'rd',lw=2, label='data point')
plt.plot(z,fz,'g-',lw=2, label='function')
plt.plot(z,ff,'k--',lw=2, label='Lagrange interpolation')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([ -3.5, 3.5, -50, 300 ])
plt.show()
fig.savefig('LagrangeInter103Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 3.1.8의 결과와 같다. ■

예제 3.1.10 예제 3.1.8을 다시 살펴보자. MATLAB함수 polyfit.m를 사용하면, 멱함수를 설명함수로 하는 회귀식을 추정할 수 있다. 이 polyfit.m을 사용해서 Lagrange보간식을 구하기 위해, 다음 MATLAB프로그램 LagrangeInter104.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename: LagrangeInter104.m
% Lagrange interpolation polynomial IV w/ polyfit.m
% Programmed by CBS
% ------------------------------------------------------------------
x = [ -3 -1 1 2 3 ];
```

```
y = [ 169 5 -7 -1 37 ];
n = length(x);
% Calculate Lagranage interpolation using polyfit.m
[ aa1,S ] = polyfit(x,y,n-1)
% Calculate Lagrange interpolation
z = [-3.5:0.01:3.5];
ff = polyval(aa1,z);
% original funtion
a = [ 1 -2 3 -4 -5 ]
fz = polyval(a,z);
err = max(abs(fz-ff))
% Plotting
plot(x,y,'rd',z,fz,'g-',z,ff,'k--','LineWidth',2)
legend('data point','function','Lagrange interpolation',1)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-3.5 3.5])
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
saveas(gcf,'LagrangeInter104.jpg')
save('LagrangeInter104.txt','fz','err','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 LagrangeInter104.m에서는 MATLAB함수 polyfit.m을 사용하였다. 이 함수의 사용법은 다음과 같다.

```
>> [ regcoeff, S ] = polyfit(x,y,n)
```

이 MATLAB명령문이 실행되면, 설명변수가 x이고 종속변수가 y인 데이터세트를 차수가 n인 멱함수로 최소제곱추정한다. 이 명령문을 실행한 결과, 회귀계수들이 출력변수 regcoeff에 저장되고, 정규방정식의 계수행렬, 오차항의 자유도, 그리고 잔차벡터의 노름이 출력변수 S에 저장된다. 또한, 이 MATLAB프로그램에서는 멱함수의 계수들을 사용해서 멱함수를 계산하기 위해, MATLAB함수 polyval.m을 사용하였다. 이 함수의 사용법은 다음과 같다.

```
>> y = polyval(c,x)
```

이 MATLAB명령문은 주어진 계수벡터 $c \doteq [c_1\ c_2\ \cdots\ c_n\ c_{n+1}]$와 주어진 x값에 대한 다음 멱함수값 y를 계산한다.

$$y = \sum_{i=0}^{n} c_{n+1-i} x^i \tag{1}$$

이 MATLAB프로그램 LagrangeInte104.m을 실행하면, 집합 D의 점들을 지나는 4차 Legendre보간식이 다음과 같음을 알 수 있다.

$$p_4(x) = -5.000000 + x[-4.000000 + x\{3.000000 + x[-2.000000 + 1.000000x]\}] \tag{2}$$

원래 집합 D는 다음 함수에서 추출한 관찰점들이다.

$$f(x) = x^4 - 2x^3 + 3x^2 - 4x - 5 \tag{3}$$

구간 $[-3.5, 3.5]$에서 함수 $f(x)$와 Lagrange보간식 사이의 최대오차 $\max|f(x) - p_4(x)|$는 5.7×10^{-14}이다. 또한, 이 MATLAB프로그램을 실행하면, 집합 D, 함수 $f(x)$, 그리고 Lagrange보간식 $p_4(x)$의 그림 3.1.4를 출력한다. 그림 3.1.4는 그림 3.1.2와 같다. 즉, 함수 $f(x)$와 Lagrange보간식 $p_4(x)$는 구간 $[-3.5, 3.5]$에서 정확히 일치한다. 또한, 집합 D의 점들은 이 두 곡선들 위에 존재한다. ■

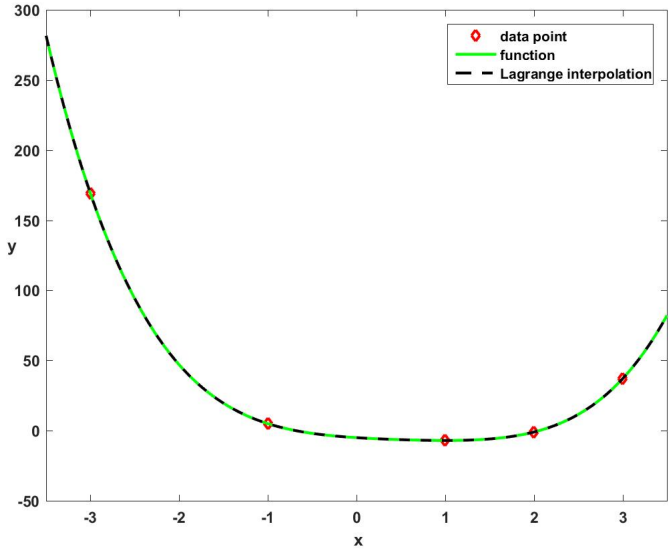


그림 3.1.4. Lagrange보간식 IV

예제 3.1.11 Python을 사용해서 예제 3.1.10을 다시 다루기 위해서, 다음 Python프로그램 LagrangeInter104.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

# Calculate Lagrange coefficients
x = np.array([ -3, -1, 1, 2, 3 ]);
y = np.array([ 169, 5, -7, -1, 37 ]);
n = len(x);
LL = np.zeros((n,n),float)
```

```
# Calculate Lagranage interpolation using polyfit
aa = np.polyfit(x, y, n-1)
# Calculate Lagrange interpolation
z = np.arange(-3.5, 3.51, 0.01);
ff = np.polyval(aa,z);
# original funtion
a = np.array([ 1, -2, 3, -4, -5 ])
fz = np.polyval(a,z);
err = max(np.abs(fz-ff))

# Plotting
fig = plt.figure()
plt.plot(x,y,'rd',lw=2, label='data point')
plt.plot(z,fz,'g-',lw=2, label='function')
plt.plot(z,ff,'k--',lw=2, label='Lagrange interpolation')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([ -3.5, 3.5, -50, 300 ])
plt.show()
fig.savefig('LagrangeInter104Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 3.1.10의 결과와 같다. ■

Lagrange보간식을 사용해서 좋은 근사식을 얻을 수 없는 대표적인 예가 다음과 같은 Runge함수이다.

$$f(x) = \frac{1}{1+x^2} \tag{3.1.21}$$

구간 $[-5,5]$를 등간격인 n개의 소구간들로 나누기로 하자. 즉, 다음과 같이 마디점들을 정하자.

$$x_i \doteq -5 + \frac{10i}{n}, \quad (i = 0, 1, \cdots, n) \tag{3.1.22}$$

각 마디점 x_i에서 함수값을 f_i라 하자. 또한, 주어진 점들 $\{(x_i, f_i)\}$를 지나는 $[n-1]$차 멱함수를 $p_n(x)$라 하자. 비록 n을 크게 잡는다고 해도, $p_n(x)$가 함수 $f(x)$에 가까운 것은 아니다. 좀 더 명확하게 말한다면, 각 $x(\in [-3.64, 3.64])$에 대해서 다음 식이 성립한다.

$$\overline{\lim_{n\to\infty}} |f(x) - p_n(x)| = \infty \tag{3.1.23}$$

여기서 함수 f가 C^∞급이라는 것을 고려한다면, 식 (3.1.23)은 상당히 심각한 문제이다. 식 (3.1.23)를 Runge현상 (Runge phenomenon)이라 부른다. 그러나, 명제 1.2.2의 관점에서 본다면, 식 (3.1.23)은 당연한 것이다. 왜냐하면, 이 Runge함수 f는 $x = \pm i$를 특이점 (또는 극점)으로 갖기 때문이다.

Runge현상에서 알 수 있듯이, 단순히 마디점들의 개수를 증가시킨다고 해서 항상 보간오차를 줄일 수 있는 것은 아니다. Runge현상은 보간구간의 끝점들 부근에서 일어난다. 즉, 이 문제는 보간구간에서 각 관찰점에 대한 가중함수를 1로 놓는데서 발생한다고 할 수 있다. 따라서, 양 끝점들에 가중값을 더 주는 보간법을 사용하면, 이 문제를 해결할 수 있다. 즉, 마디점들을 등간격으로 선택할 것이 아니라 양 끝에서 더 많은 마디점들을 선택하면, Runge현상을 피할 수 있을 것이다.

예제 3.1.12 다음과 같은 Runge함수 f를 살펴보자.

$$f(x) = \frac{1}{1 + 12x^2} \tag{1}$$

Runge함수는 Cauchy확률분포의 확률밀도함수에 해당한다. 이 Runge함수의 Lagrange보간식들을 그리기 위해서, 다음 MATLAB프로그램 RUNGEfunction101.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename RUNGEfunction101.m
% Lagrange interpolation of Runge function
% Programmed by CBS
% ------------------------------------------------------------------
clear all, clf
runge = inline('1./(1 + 12*x.*x )');
x = -1:0.01:1;
plot(x,runge(x),'k-','LineWidth',2.0);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-1,1])
hold all
for ndum = 1:3;
    npoint = ndum*5;
    node = -1:2/npoint:1;
    coeff = polyfit(node,runge(node),npoint);
    plot(x,polyval(coeff,x),'LineWidth',2.0);
end
hold off
xlabel('x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
legend('\bf f','\bf f_5','\bf f_{10}','\bf f_{15}',0)
saveas(gcf,'RUNGEfunction101','jpg')
save('RUNGEfunction101.txt','coeff')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 RUNGEfunction101.m을 실행하면, 그림 3.1.5가 그려진다. 그림 3.1.5에는 Runge함수 f가 흑색 점선, 5차 Lagrange보간식 p_5가 청색 직선, 10차 Lagrange보간식 p_{10}이 녹색 실선, 그리고 15차 Lagrange보간식 p_{15}가 적색 실선으로 그려져 있다. 이렇게 색깔을 바꾸어 곡선을 그리기 위해서, 'hold on' 대신 'hold all'을 사용하였다. 그림 3.1.5에서 알 수 있듯이, 차수가 높은 Lagrange보간식을 사용해도 Runge함수의 좋은 근사식을

얻을 수 없다. ■

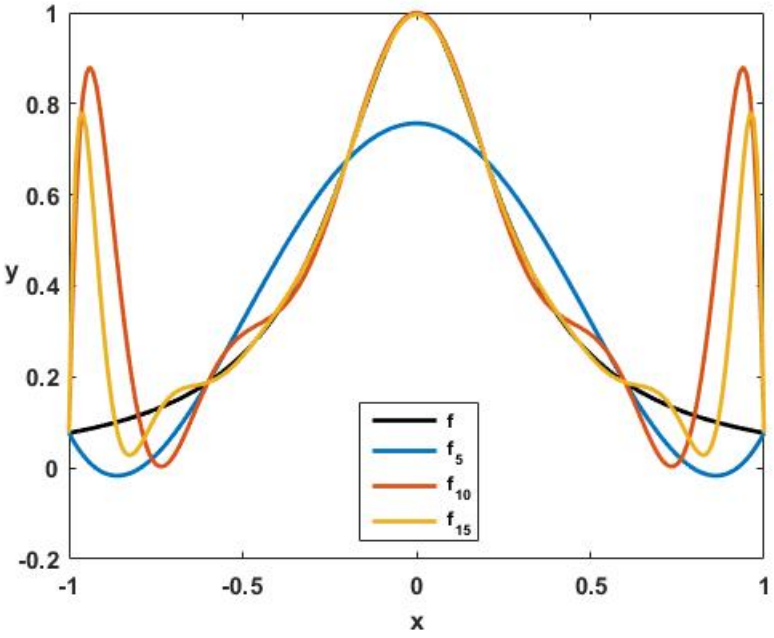


그림 3.1.5. Runge함수의 Lagrange보간

제3.2절 Hermite보간

보간함수 p_n이 다음 식들을 만족한다고 하자.

$$p_n(x_i) = f_i, \quad p_n'(x_i) = f_i', \quad (i = 1, 2, \cdots, n) \tag{3.2.1}$$

만약 보간식 p_n이 멱함수이면, 식 (3.2.1)를 만족하는 p_n은 최대 $[2n-1]$차의 멱함수이다. 각 점 x_i에서 함수값과 도함수값이 일치하는 멱함수를 찾기 위해서, 다음 함수들을 정의하자.

$$\tilde{h_i}(x) \doteq [x - x_i] l_i^2(x) \tag{3.2.2}$$

$$h_i(x) \doteq \{1 - 2l_i'(x)[x - x_i]\} l_i^2(x), \tag{3.2.3}$$

여기서 $l_i(x)$는 Lagrange보간함수이다. 다음 식들이 성립함을 쉽게 알 수 있다.

$$h_i(x_j) = \delta_{i,j}, \quad \tilde{h_i}'(x_j) = \delta_{i,j} \tag{3.2.4}$$

$$h_i'(x_j) = 0, \qquad \tilde{h_i}(x_j) = 0 \tag{3.2.5}$$

다음 함수를 정의하자.

$$p(x) \doteq \sum_{i=1}^{n} f_i h_i(x) + \sum_{i=1}^{n} f_i' \tilde{h_i}(x) \tag{3.2.6}$$

식 (3.2.4)와 (3.2.5)에서 알 수 있듯이, 함수 $p(x)$는 조건 (3.2.1)을 만족한다. 따라서, $[2n-1]$차 Hermite보간함수를 다음과 같이 정의하자.

$$p_{2n-1}(x) \doteq \sum_{i=1}^{n} f_i\{1 - 2l_i'(x_i)[x - x_i]\}l_i^2(x) + \sum_{i=1}^{n} f_i'[x - x_i]l_i^2(x) \tag{3.2.7}$$

Lagrange보간식 (3.1.6)과 마찬가지로 Hermite보간식 (3.2.7)은 계산효율적이 아니다. 실제로 Hermite보간식을 계산할 때는 Newton의 차분나누기법(divided difference method)을 사용한다. 이에 대한 자세한 내용은 Young & Gregory [68, pp. 288-293]를 참조하라.

예제 3.2.1 다음 점들을 지나는 Hermite보간식을 구해보자.

$$D = \{(-3, 169), (-1, 5), (1, -7), (2, -1), (3, 37)\} \tag{1}$$

이를 위해서, 다음 MATLAB프로그램 HermiteInter101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename HermiteInter101.m
% Hermite interpolation polynomial
% Using Newton's divided difference method
% Programmed by CBS
% ------------------------------------------------------------------------
function HermiteInter101
x = [ -3 -1 1 2 3 ]
y = -5 + x.*(-4+x.*(3+x.*(-2 + x)))
dy = -4 + x.*(6 + x.*(-6 + 4*x))
% Calculate the Hermite interpolation
coeff = HermiteCoefficient(x,y,dy);
z = [-3.5:0.01:3.5];
n = length(x);
for i=1:n
    xx(2*i-1) = x(i); xx(2*i) = x(i);
end
for ii = 1:length(z)
    dum1(1) = 1;
    dumf(1) = coeff(1);
    for jj = 2:2*n
```

```
        dum1(jj) = ( z(ii)-xx(jj-1) ).*dum1(jj-1);
        dumf(jj) =  coeff(jj).*dum1(jj);
    end
    ff(ii) = sum(dumf);
end
% Original funtion
a = [ 1 -2 3 -4 -5 ];
fz = polyval(a,z);
err = max(abs(fz-ff))
% Plotting
plot(z,fz,'g-',z,ff,'k--',x,y,'rd','LineWidth',2.0)
legend('function','Hermite interpolation','data point',1)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-3.5 3.5])
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
saveas(gcf,'HermiteInter101','jpg')
save('HermiteInter101.txt','fz','err')
end
% End of program
% ------------------------------------------------------------------------
function coeff = HermiteCoefficient(x,ff,dff)
n = length(x);
coeff(1) = ff(1);
for i=1:n
    xx(2*i-1) = x(i); yy(2*i-1) = ff(i);
    xx(2*i) = x(i); yy(2*i) = ff(i);
end
for k=1:n-1
    dd(2*k-1,1) = dff(k);
    dd(2*k,1) = ( yy(2*k+1)-yy(2*k) )/( xx(2*k+1)-xx(2*k) );
end
dd(2*n-1,1) = dff(n);
for j = 2:2*(n-1)
    for k = 1:2*n-j
    dd(k,j) = ( dd(k+1,j-1)-dd(k,j-1) )/( xx(k+j)-xx(k) );
    end
end
dd(1,2*n-1) = ( dd(2,2*(n-1))-dd(1,2*(n-1)) )/( xx(2*n)-xx(1) );
for j = 2:2*n
    coeff(j) = dd(1,j-1);
end
dd
coeff
end
% End of program
% ------------------------------------------------------------------------
```

마디점 x_i들은 다음과 같다.

$$x_0 = -3, \quad x_1 = -1, \quad x_2 = 1, \quad x_3 = 2, \quad x_4 = 3 \tag{1}$$

제4차 Hermite보간식을 다음과 같이 표기하자.

$$p_4(x) = c_0 + \sum_{i=1}^{4} c_i [x - x_0] \cdots [x - x_{i-1}] \tag{2}$$

여기서 $c_0, c_1, \cdots, c_4$는 차분나누기계수들이다.

MATLAB프로그램 HermiteInter101.m을 실행하면, 집합 D의 점들을 지나는 4차 Hermite 보간식의 차분나누기계수들이 다음과 같음을 알 수 있다.

$$c_0 = 169, \quad c_1 = -184, \quad c_2 = 51, \quad c_3 = -10, \quad c_4 = 1 \tag{2}$$

따라서, Hermite보간식 $p_4(x)$는 다음과 같다.

$$\begin{aligned} p_4(x) =& 169 - 184[x+3] + 51[x+3][x+1] \\ & -10[x+3][x+1][x-1] + [x+3][x+1][x-1][x-2] \end{aligned} \tag{5}$$

원래 집합 D는 다음 함수에서 추출한 관찰점들이다.

$$f(x) = x^4 - 2x^3 + 3x^2 - 4x - 5$$

구간 $[-3.5, 3.5]$에서 함수 $f(x)$와 Lagrange보간식 $p_4(x)$ 사이 최대오차 $\max|f(x) - p_4(x)|$는 $3.4 \cdot 10^{-13}$이다. 또한, 이 MATLAB프로그램을 실행하면, 집합 D, 함수 $f(x)$, 그리고 Hermite보간식 $p_4(x)$를 그린 그림 3.2.1을 출력한다. 그림 3.2.1에서 알 수 있듯이, 함수 $f(x)$와 Hermite보간식 $p_4(x)$는 구간 $[-3.5, 3.5]$에서 일치한다. 또한, 집합 D의 점들은 이 두 곡선들 위에 존재한다. ■

예제 3.2.2 Python을 사용해서 예제 3.2.1을 다시 다루기 위해서, 다음 Python프로그램 HermiteInter101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

def HermiteCoefficient(x,ff,dff):
    n = len(x);
    coeff = [0.0]*(2*n);
    coeff[0] = ff[0];
    xx = [0.0]*(2*n);
    yy = [0.0]*(2*n);
    for ii in range(0,n):
        xx[2*ii] = x[ii];   xx[2*ii+1] = x[ii];
        yy[2*ii] = ff[ii];  yy[2*ii+1] = ff[ii];
```

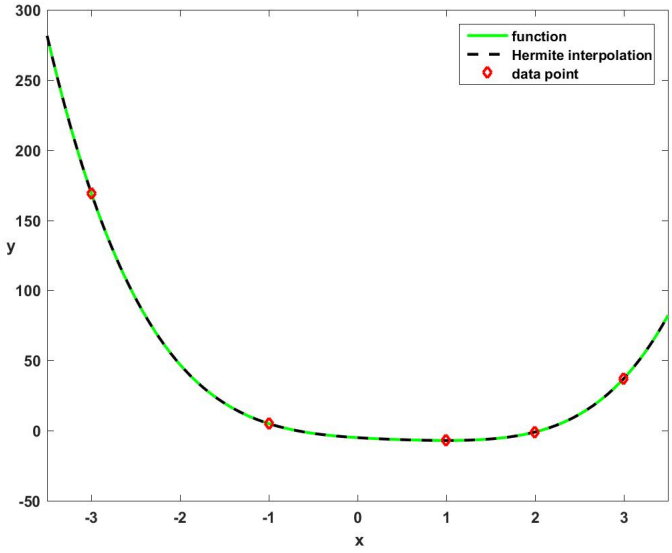


그림 3.2.1. Hermite보간식

```
    dd = np.zeros((2*n-1,2*n-1),float);
    # First column
    for kk in range(0,n-1):
        dd[2*kk,0] = dff[kk];
        dd[2*kk+1,0] = ( yy[2*kk+2]-yy[2*kk+1] )/( xx[2*kk+2]-xx[2*kk+1] );
    dd[2*n-2,0] = dff[n-1];
    # Second to Ninth columns
    for jj in range(1,2*n-2):
        for kk in range(0,(2*n-jj-1)):
            dd[kk,jj] = ( dd[kk+1,jj-1]-dd[kk,jj-1] )/( xx[kk+jj+1]-xx[kk] )
    ;
    dd[0,2*n-2] = ( dd[1,2*n-3]-dd[0,2*n-3])/( xx[2*n-1]-xx[0]);
    print(dd)
    for ll in range(1,2*n):
        coeff[ll] = dd[0,ll-1];
    print(coeff)
    return coeff

# Calculate Hermite interpolation
x = np.array([ -3, -1, 1, 2, 3 ]);
y = -5 + x*(-4+x*(3+x*(-2 + x)));
dy = -4 + x*(6 + x*(-6 + 4*x));
coeff = HermiteCoefficient(x,y,dy);

# Evaluate Hermite interpolation at x = z
z = np.arange(-3.5,3.51,0.01);
n = len(x);
xx = [0.0]*(2*n);
for ii in range(0,n):
    xx[2*ii] = x[ii];
    xx[2*ii+1] = x[ii];

dum1 = [0]*(2*n);
dumf = [0]*(2*n);
ff   = [0]*len(z);
for ii in range(0,len(z)):
    dum1[0] = 1;
```

```
    dumf[0] = coeff[0];
    for jj in range(1,2*n):
        dum1[jj] = ( z[ii]-xx[jj-1] )*dum1[jj-1];
        dumf[jj] =  coeff[jj]*dum1[jj];
    ff[ii] = np.sum(dumf);

# original funtion
a = np.array([ 1, -2, 3, -4, -5 ])
fz = np.polyval(a,z);
err = max(np.abs(fz-ff))

# Plotting
fig = plt.figure()
plt.plot(x,y,'rd',lw=2, label='data point')
plt.plot(z,fz,'g-',lw=2, label='function')
plt.plot(z,ff,'k--',lw=2, label='Hermite interpolation')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([ -3.5, 3.5, -50, 300 ])
plt.show()
fig.savefig('HermiteInter101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 3.2.1의 결과와 같다. ■

제3.3절 기수함수보간

마디점들 $x_1, x_2, \cdots, x_n$에 대해 다음 식들을 만족하는 함수들 $\phi_1, \phi_2, \cdots, \phi_n$이 존재한다고 하자.

$$\phi_i(x_j) = \delta_{i,j} \tag{3.3.1}$$

만약 함수들 $\phi_1, \phi_2, \cdots, \phi_n$이 서로 독립이면, 이들을 기수함수기저(cardinal function basis)라 한다. 이 기수함수기저를 사용하면, 다음과 같은 함수 f의 보간식을 얻는다.

$$p_n(x) = \sum_{i=1}^{n} f_i \phi_i(x) \tag{3.3.2}$$

Lagrange보간법과 Hermite보간법이 대표적인 기수함수보간법이다. 예를 들어, Lagrange 보간에서는 다음과 같은 기수함수기저 $\{l_j(x)\}$를 사용하였다.

$$l_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}, \quad (j = 1, 2, \cdots, n) \tag{3.3.3}$$

기수함수보간법의 장점은 선형결합의 계수가 주어진 값 f_i 자체라는 것이다. 반면에, 기수

함수보간법의 단점은 어떤 점 x에서 $p_n(x)$를 계산하기 위해서는 $\phi_1(x), \phi_2(x), \cdots, \phi_n(x)$를 모두 계산해야한다는 것이다. 따라서, 많은 계산량을 필요로 할 수도 있다.

제3.4절 Lagrange보간식의 보간오차

Lagrange보간식의 보간오차(interpolation error)에 대해서 살펴보자. 다음 명제가 성립한다. 이 명제의 증명은 Young & Gregory [68, pp. 266-267]를 참조하라.

명제 3.4.1

만약 함수 f가 C^n급이면, 구간 $[a,b]$의 분할 $\Pi \doteq \{a = x_1 < x_2 < \cdots < x_n = b\}$에 대해 다음 식을 만족하는 $\xi(\in [a,b])$가 존재한다.

$$f(x) - \sum_{i=1}^{n} f(x_i) l_i(x) = \frac{1}{n!} f^{(n)}(\xi) F_n(x)$$

여기서 $F_n(x) = \prod_{i=1}^{n} [x - x_i]$이다.

따름정리 3.4.1

명제 3.4.1의 조건 하에서, Π가 등간격 분할이라고 하자. 즉, 다음 식들이 성립한다고 하자

$$\frac{x_{i+1} - x_i}{n} = h, \quad (i = 1, 2, \cdots, n-1)$$

이러한 조건 하에서 다음 식이 성립한다.

$$\left| f(x) - \sum_{i-1}^{n} f(x_i) l_i(x) \right| \leq \frac{1}{4n} \max_{x \in [a,b]} |f^{(n)}(x)| h^n$$

증명. 구간 $[a,b]$에서 함수 $F_n(x)$는 소구간 $[x_1, x_2]$ 또는 소구간 $[x_{n-1}, x_n]$에서 최대값을 갖는다. 일반성을 잃지 않고, 함수 $F_n(x)$가 소구간 $[x_1, x_2]$에서 최대값을 갖는다고 가정하자. 다음 식들이 성립함은 명백하다.

$$x - x_i \leq [i-1]h, \quad (i = 3, 4, \cdots, n) \tag{1}$$

다음 식들이 성립한다.

$$\begin{aligned}
\max_{x\in[a,b]}|F_n(x)| &= \max_{x\in[a,b]}\left|\prod_{i=1}^{n}[x-x_i]\right| \\
&\le \max_{x\in[x_1,x_2]}\left|[x-x_1][x-x_2]\right| \max_{x\in[x_1,x_2]}\left|\prod_{i=3}^{n}[x-x_i]\right| \\
&\le \frac{1}{4}h^2(n-1)!h^{n-2} = \frac{1}{4}(n-1)!h^n \qquad (2)
\end{aligned}$$

여기서 두 번째 부등호는 식 (1)에 의해서 성립한다. 명제 3.4.1에 식 (2)를 대입하면, 증명이 끝난다. ■

명제 3.4.1에서 알 수 있듯이, 함수 f의 $[n-1]$차 Lagrange보간함수 $p_n(x)=\sum_{i=1}^{n} f_i l_i(x)$는 다음 식을 만족한다.

$$\sup_{x\in[a,b]}|f(x)-p_n(x)| \le \sup_{x\in[a,b]}F_n(x)\frac{1}{n!}||f^{(n)}||_\infty \qquad (3.4.1)$$

만약 식 $\lim_{n\to\infty}\frac{1}{n!}||f^{(n)}||_\infty = 0$이 성립하면, $p_n(x)$은 함수 f에 일양수렴한다. 그러나, C^∞급 함수 f에 대해서도 식 $\lim_{n\to\infty}\frac{1}{n!}||f^{(n)}||_\infty = \infty$가 성립하는 경우도 있다. 명제 3.4.1에서 알 수 있듯이, 함수 f의 $[2n-1]$차 Hermite보간함수 $P_{2n-1}(x)$에 대해서 다음 식을 만족하는 $\xi(\in[a,b])$가 존재한다.

$$|f(x)-P_{2n-1}(x)| \le F_{2n}^2(x)\frac{1}{(2n)!}|f^{(2n)}(\xi)| \qquad (3.4.2)$$

만약 n이 커짐에 따라 식 $\lim_{n\to\infty}\frac{1}{(2n)!}||f^{(2n)}(\xi)||_\infty = 0$이 성립하면, $P_{2n-1}(x)$은 함수값 $f(x)$에 수렴한다. 그러나, 식 (3.1.23)에서 알 수 있듯이, C^∞급 함수 f에 대해서도 식 $\lim_{n\to\infty}\frac{1}{(2n)!}||f^{(2n)}(\xi)||_\infty = \infty$가 성립하는 경우도 있다. 즉, Hermite보간함수를 사용한다고 해서 보간오차문제가 해결되는 것은 아니다.

예제 3.4.1 함수 $f(x)=\ln(x)$에 대해서 다음 식들이 성립한다.

$$f^{(n)}(x) = [-1]^{n-1}\frac{1}{(n-1)!}\frac{1}{x^n}, \quad (n=1,2,\cdots) \qquad (1)$$

아주 작은 양수 ξ에 대해서 $|f^{(n)}(\xi)|$는 매우 크다. 명제 3.4.1에서 알 수 있듯이, Lagrange 보간은 구간 $(0,1]$에서 함수 f의 좋은 근사식을 제공하지 못한다. ■

제3.5절 Chebyshev회귀식

앞에서도 언급했듯이, 보간오차를 줄이기 위해서 등간격이 아닌 마디점들을 선택하기로 하자. 이러한 목적을 달성하는데 가장 중요한 성질은 Chebyshev함수들의 미니맥스성이다.

명제 3.5.1

각 자연수 n에 대해서, 다음 최적화문제를 살펴보자.

$$s_n \doteq \inf_{p_{n-1}} \max_{x\in[-1,1]} |x^n + p_{n-1}(x)|$$

여기서 p_{n-1}은 최대 $[n-1]$차 멱함수이다. 이러한 조건 하에서, 이 최적화문제의 해는 다음과 같다.

$$p_{n-1}(x) = \frac{1}{2^{n-1}} T_n(x) - x^n$$

또한, 이 멱함수에 대해서 $s_n = 2^{1-n}$이다.

명제 3.5.1의 증명은 Powell [45, p. 78]을 참조하라. 명제 3.5.1에서 알 수 있듯이, $2^{1-n}T_n(x)$는 최고차가 x^n인 멱함수들 중에서 가장 작은 최대크기(maximum magnitude)를 갖는 멱함수이다.

명제 3.5.1에서 기술한 Chebyshev함수들의 미니맥스성을 보간오차에 적용해서, 좋은 마디점들을 선택해보자. 식 (3.4.1)에서 알 수 있듯이, 함수 f의 $[n-1]$차 Lagrange보간함수 $p_n(x) = \sum_{i=1}^{n} f_i l_i(x)$에 의한 최대오차 e_M은 다음 식을 만족한다.

$$e_M \leq \sup_{x\in[-1,1]} F_n(x) \frac{1}{n!} ||f^{(n)}||_\infty \tag{3.5.1}$$

여기서 유의할 점은 $n!$은 n에만 의존하고, $||f^{(n)}(x)||_\infty$은 함수 f에만 의존한다. 또한, 이들은 마디점들과는 독립이므로, 우리가 선택하는 마디점들 $x_1, x_2, \cdots, x_n$은 단지 $\sup_{x\in[-1,1]} F_n(x)$에만 의존한다. 따라서, 보간오차를 최소화하는 마디점들을 선택하는 문제는 $\sup F_n(x)$를 최소화하는 마디점을 선택하는 것이다. 즉, 다음 최적화문제를 푸는 것이다.

$$\min_{\Pi} \max_{x\in[-1,1]} F_n(x) = \min_{\Pi} \max_{x\in[-1,1]} \prod_{i=1}^{n} [x - x_i] \tag{3.5.2}$$

함수 $F_n(x)$의 최고항은 x^n이므로, 명제 3.5.1에서 알 수 있듯이, $F_n(x)$가 $2^{1-n}T_n(x)$이면

보간오차가 최소이다. 따라서, 방정식 $F_n(x) = 0$ 의 근들은 방정식 $T_n(x) = 0$ 의 근들과 같아야 한다. 즉, 다음 식들이 성립한다.

$$x_i = \cos\left(\frac{2i-1}{2n}\pi\right), \quad (i = 1, 2, \cdots, n) \tag{3.5.3}$$

Chebyshev함수의 근들을 마디점들로 하는 멱함수가 함수 f 의 보간식으로 적당한지를 알아보기 위해서, 다음 명제를 살펴보자.

명제 3.5.2

어떤 자연수 k 에 대해서 함수 f 가 $C^k[-1,1]$ 급 함수라고 하고, 이 함수 f 의 Chebyshev 함수 $T_n(x)$ 의 근들을 마디점들로 하는 n 차 보간함수를 C_n 이라 하자. 다음 식이 성립한다.

$$||f - C_n||_\infty \leq \left[\frac{2}{\pi}\ln(n+1) + 1\right] \frac{(n-k)!}{n!} \left[\frac{\pi}{2}\right]^k ||f^{(k)}||_\infty$$

명제 3.5.2의 증명은 Rivlin [49] 을 참조하라. 명제 3.5.2에서 알 수 있듯이, C^1 급 함수에 대해서 Chebyshev보간이 적당하다. 그러나, 그 결과는 Fourier해석에 의한 보간만큼 좋지는 않다.

구간 $[-1,1]$ 에서 보간을 할 때, 양 끝점들 -1과 1을 포함하는 것이 자연스러워 보인다. 그러나, 식 (3.5.3) 의 근들은 모두 구간 $(-1,1)$ 에 속한다. 따라서, 엄격히 말한다면 양 끝점들 -1 또는 1에 가까운 점 x 에서 Chebyshev보간은 내삽 (interpolation) 이 아니라 외삽 (extrapolation) 이다. 결과적으로, 이러한 점에서는 Chebyshev보간이 잘 작동하지 않는다. 즉, 보간오차가 크다. 만약 점 $x = -1$ 과 점 $x = 1$ 을 마디점에 포함시키려면, 다음과 같은 마디점들을 사용해야 한다.

$$x_i = \sec\left(\frac{\pi}{2n}\right)\cos\left(\frac{2i-1}{2}\frac{\pi}{n}\right), \quad (i = 1, 2, \cdots, n) \tag{3.5.4}$$

예제 3.5.1 등간격인 마디점들을 사용하는 Lagrange보간과 등간격이 아닌 마디점들을 사용하는 Chebyshev보간을 비교하기 위해서, 다음 MATLAB프로그램 Chebyshev2Lagrange101.m 을 실행해보자.

```
% ------------------------------------------------------------------------
% Filename Chebyshev2Lagrange101.m
```

```
% Chebyshev interpolation & Lagrange interpolation
% Programmed by CBS
% ------------------------------------------------------------------------
runge = inline('1./( 1+x.^2 )','x');
n = 11;
% Lagrange interpolation
EQUInode = -5:1:5;
lagrange = polyfit(EQUInode,runge(EQUInode),n-1);
CHEBnode = 5*cos( (2*(1:n) -1)/2/n*pi );
chebyshev = polyfit(CHEBnode,runge(CHEBnode),n-1);
% plot
xx = -5:0.01:5;
ytrue = runge(xx);
ylagrange = polyval(lagrange,xx);
ychebyshev = polyval(chebyshev,xx);
% Plotting
plot(xx,ytrue,'k-',xx,ylagrange,'r-.',xx,ychebyshev,'b--', ...
     'LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
legend('Original function','Lagrnage','Chebyshev','location','NW')
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
saveas(gcf,'Chebyshev2Lagrange101.jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 Chebyshev2Lagrange101.m을 실행하면, 다음과 같은 Runge함수의 Lagrange보간식과 Chebyshev보간식을 구한다.

$$f(x) = \frac{1}{1+x^2}, \quad (x \in [-5, 5]) \tag{1}$$

이 보간식들을 그린 것이 그림 3.5.1이다. 이 보간식들은 똑같이 11개의 마디점들을 사용했음에도 불구하고, Lagrange보간식보다는 Chebyshev보간식이 원래 함수에 더 가깝다. ■

예제 3.5.2 Python을 사용해서 예제 3.5.1을 다시 다루기 위해서, 다음 Python프로그램 Chebyshev2Lagrange101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

# Rundge function
runge = lambda x: 1/( 1+x**2 )
n = 11;
# Lagrange Interpolation
EQUInode = np.arange(-5, 6, 1);
lagrange = np.polyfit(EQUInode,runge(EQUInode),n-1);
```

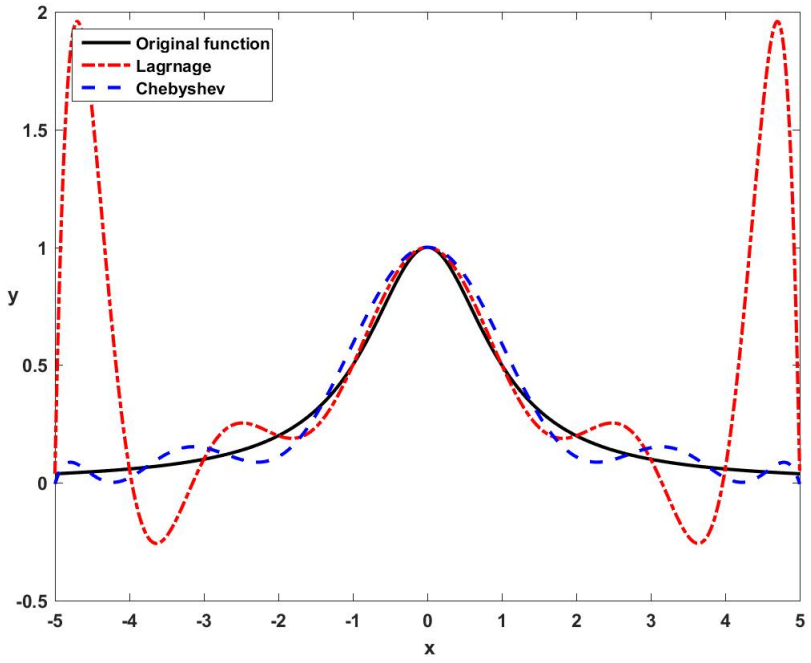


그림 3.5.1. Lagrange보간식과 Chebyshev보간식

```
CHEBnode = 5*np.cos( (2*np.linspace(1,n,num=n)-1 )/2/n*np.pi );
chebyshev = np.polyfit(CHEBnode,runge(CHEBnode),n-1);

# Approximation
xx = np.arange(-5,5.005,0.01);
ytrue = runge(xx);
ylagrange = np.polyval(lagrange,xx);
ychebyshev = np.polyval(chebyshev,xx);

# Plotting
fig = plt.figure()
plt.plot(xx,ytrue,'k-',lw=2, label='Original function')
plt.plot(xx,ylagrange,'r-.',lw=2, label='Lagrange')
plt.plot(xx,ychebyshev,'b--',lw=2, label='Chebyshev')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper center', numpoints=1)
plt.axis([ -5, 5, -0.5, 2 ])
plt.show()
fig.savefig('Chebyshev2Lagrange101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 3.5.1의 결과와 같다. ■

연속형 최소제곱추정법은 구간 $[a,b]$에서 함수 f를 $[n-1]$차원 멱함수로 근사시키는 것이다. 따라서, 구간 $[a,b]$의 모든 점들을 사용해서 근사식을 구한다. 반면에, 보간법은 마디점들 $x_1, x_2, \cdots, x_n (\subset [a,b])$에서 함수 f와 만나는 $[n-1]$차원 멱함수를 구하는 것이다.

만약 마디점들의 개수 n이 충분히 크다면, 차수가 $m(\leq n-1)$인 멱함수로 함수 f의 근사식을 나타낼 수 있다. 이러한 과정에 회귀분석을 사용하는 것은 당연하다. 따라서, 이러한 근사식을 회귀식이라 부르기로 하자. 회귀분석이론에 의하면, 주어진 n에 대해서 회귀식의 차수 m이 커지면 회귀식의 편의(bias)는 줄지만 분산은 커진다. 즉, m이 지나치게 커지면, 추정된 회귀식을 믿을 수 없다. 이를 다음과 같이 해석할 수 있다. 만약 차수를 m으로 하면, Chebyshev보간식에서 마지막 $[n-m-1]$개 항들을 버리는 것이다. 이 고차항들은 바람직하지 않는 진동(oscillation)을 발생시킬 수 있다. 따라서, 이 고차항들을 제거하는 것은 불필요한 진동을 제거함으로써, 보간식을 좀 더 평활하게 만드는 것이다. 이렇게 불필요한 진동을 제거하는 것은 Chebyshev절약에 해당한다. 구간 $[-1,1]$에서 n차 Chebyshev함수의 n개 근들 $\{z_1, z_2, \cdots, z_n\}$을 마디점들로 사용해서, 함수 f의 구간 $[a,b]$에서 m차 회귀다항식 $\hat{p}_m$을 구하는 알고리즘은 다음과 같다.

알고리즘 3.5.1

(1단계) 다음 식들을 사용해서, 구간 $[-1,1]$에서 n차 Chebyshev함수의 근들인 마디점들 $\{z_1, z_2, \cdots, z_n\}$을 구한다.

$$z_i = -\cos\left(\frac{2i-1}{2n}\pi\right), \quad (i = 1, 2, \cdots, n)$$

(2단계) 다음 식들을 사용해서, 이 마디점들을 다음과 같이 구간 $[a,b]$에서 마디점들로 변환한다.

$$x_i = a + \frac{b-a}{2}[z_i + 1], \quad (i = 1, 2, \cdots, n)$$

(3단계) 다음 식들을 사용해서, 이 마디점들에서 함수값들을 계산한다.

$$y_i = f(x_i), \quad (i = 1, 2, \cdots, n)$$

(4단계) 다음 식들을 사용해서, Chebyshev계수들 $\{c_k\}$를 계산한다.

$$c_k = \frac{\sum_{i=1}^{n} y_i T_k(z_i)}{\sum_{i=1}^{n} T_k^2(z_i)}, \quad (k = 0, 1, \cdots, m)$$

(5단계) 다음과 같은 함수 f의 구간 $[a,b]$에서 m차 회귀식 $\hat{p}_m$을 구한다.

$$\hat{p}_m(x) = \sum_{k=0}^{m} c_k T_k \left(2\frac{x-a}{b-a} - a \right)$$

예제 3.5.3 알고리즘 3.5.1을 적용해서 Runge함수의 Chebyshev회귀식을 구하기 위해, 다음 MATLAB프로그램 ChebyshevRegression101.m을 실행해보자.

```
% ------------------------------------------------------------------------
% Filename ChebyshevRegression101.m
% Chebyshev regression
% Programmed by CBS
% ------------------------------------------------------------------------
function ChebyshevRegression101
ftn = '1./( 1+x.^2 )';
runge = inline(ftn,'x');
a = -5; b = 5; n = 20; m = 9;
c1 = ChebyshevRegCoefficient(ftn,a,b,n,m)
% plots of original function and Chebyshev regression
xx = [a:(b-a)/1000:b]';
yy = runge(xx);    % original function
zz = 2/(b-a)*(xx-a) - 1;
TT1(:,1) = zz-zz+1;  % T_0
pp = c1(1)*TT1(:,1); % Chebyshev regression function
TT1(:,2) = zz;       % T_1
pp = pp + c1(2)*TT1(:,2);
for k = 2:m
    TT1(:,k+1) = 2*zz.*TT1(:,k) - TT1(:,k-1);    % T_k
    pp = pp + c1(k+1)*TT1(:,k+1);
end
% plot
plot(xx,yy,'k-',xx,pp,'r--','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
legend('Original function','Chebyshev regression',1)
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
axis([-5 5 0 1.3])
saveas(gcf,'ChebyshevRegression101.jpg')
save('ChebyshevRegression101.txt','c1','pp','-ascii')
end
% end of program
% ------------------------------------------------------------------------
function c1 = ChebyshevRegCoefficient(ftn,a,b,n,m)
% ftn, a, b : string function to be regressed on [a,b]
% n : number of Chebyshev nodes
% m : order of polynomial to be regressed
% c ; regression coefficient
dumi = [1:n]';
z = - cos( (2*dumi-1)/2/n*pi );
x = a + (b-a)/2*( z+1 );
y = eval(ftn);
% estimate regression coefficient
T1(:,1) = z-z+1;  % T_0
c1(1) = ( y'*T1(:,1) )/( T1(:,1)'*T1(:,1) );  % c_0
T1(:,2) = z;      % T_1
c1(2) = ( y'*T1(:,2) )/( T1(:,2)'*T1(:,2) );  % c_1
```

```
for k = 2:m
    T1(:,k+1) = 2*z.*T1(:,k) - T1(:,k-1);      % T_k
    c1(k+1) = ( y'*T1(:,k+1) )/( T1(:,k+1)'*T1(:,k+1) );  % c_k
end
end
% end of program
% ------------------------------------------------------------------------
```

MATLAB프로그램 ChebyshevRegression101.m을 실행하면, 다음과 같은 Runge함수의 Chebyshev회귀식을 추정한다.

$$f(x) = \frac{1}{1+x^2}, \quad (x \in [-5, 5]) \tag{1}$$

이 추정된 Chebyshev회귀식을 그린 것이 그림 3.5.2이다. 이 회귀식은 $n = 20$개의 관찰점들을 바탕으로 추정한 9차 Chebyshev회귀식이다. 이 추정된 Chebyshev회귀식은 그림 3.5.1의 Chebyshev보간식보다 평활함을 알 수 있다. ■

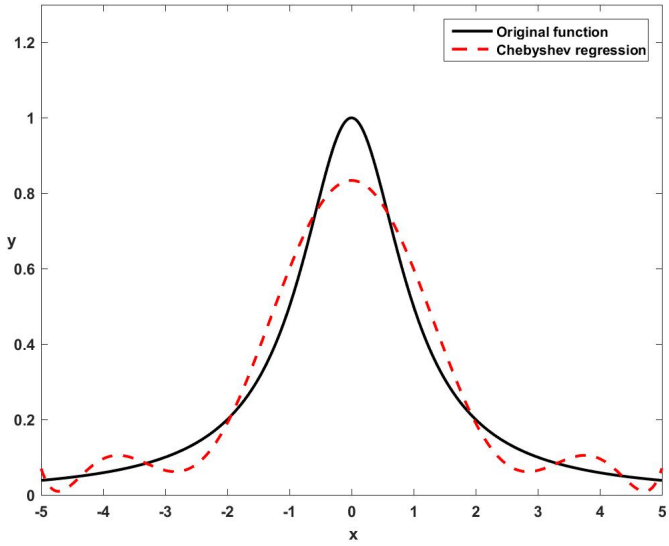


그림 3.5.2. 추정된 Chebyshev회귀식

예제 3.5.4 Python을 사용해서 예제 3.5.3을 다시 다루기 위해서, 다음 Python프로그램 ChebyshevRegression101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

```

```
import numpy as np
import matplotlib.pyplot as plt

def ChebyshevRegCoefficient(ftn,a,b,n,m):
# ftn, a, b : string function to be regressed on [a,b]
# n : number of Chebyshev nodes
# m : order of polynomial to be regressed
# c ; regression coefficient
    dumi = np.arange(1,n+1,1); dumi = np.transpose(dumi);
    z = - np.cos( (2*dumi-1)/2/n*np.pi );
    zlength = len(z)
    x = a + (b-a)/2*( z+1 );
    y = ftn(x);
    # estimate regression coefficient
    T = np.zeros([zlength,m+1],float)
    c = [0]*(m+1)
    T[:,0] = 1;
    c[0] = np.dot(y,T[:,0])/np.dot( T[:,0], T[:,0] );
    T[:,1] = z;
    c[1] = np.dot(y,T[:,1])/np.dot( T[:,1], T[:,1] );
    for k in range(2,m+1):
        T[:,k] = 2*z*T[:,k-1] - T[:,k-2];
        c[k] = np.dot(y,T[:,k])/np.dot( T[:,k], T[:,k] );
    return c

# Calculate Chebyshev Regression coefficients
runge = lambda x: 1/( 1+x**2 )
a = -5; b = 5; n = 20; m = 9;
c = ChebyshevRegCoefficient(runge,a,b,n,m)

# original function and Chebyshev regression
xx = np.arange(a,b+(b-a)/1000,(b-a)/1000);
xxlength = len(xx);
yy = runge(xx);       # original function
zz = 2/(b-a)*(xx-a) - 1;
TT = np.zeros([xxlength,m+1],float)
TT[:,0] = 1;
pp = c[0]*TT[:,0];        # Chebyshev regression function
TT[:,1] = zz;         # T_1
pp = pp + c[1]*TT[:,1];
for k in range(2,m):
    TT[:,k] = 2*zz*TT[:,k-1] - TT[:,k-2];
    pp = pp + c[k]*TT[:,k];

# Plotting
fig = plt.figure()
plt.plot(xx,yy,'k-',lw=2, label='Original function')
plt.plot(xx,pp,'r--',lw=2, label='Chebyshev regression')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([ -5, 5, 0, 1.3 ])
plt.show()
fig.savefig('ChebyshevRegression101Py.png')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 3.5.3의 결과와 같다. ■

제3.6절 다변수함수의 내삽

지금까지는 단변수함수의 내삽에 대해서 살펴보았다. 이 절에서는 다변수함수의 내삽에 대해서 살펴보자. 다변수함수의 내삽에 대한 자세한 내용을 기술하는 것은 본서의 범위를 넘는다. 따라서, 이 절에서는 MATLAB함수들 ndgrid.m과 griddedInterpolant.m을 사용해서 다변수함수를 내삽하는 방법만을 다루기로 하자.

다변수함수의 내삽법인 MATLAB함수 griddedInterpolant.m을 적용하는 첫 번째 단계는 내삽하고자 점들로 구성된 격자를 만들어야 한다. 이러한 과정을 수행하기 위해서는 MATLAB함수 ndgrid.m을 사용한다. 물론, 다른 MATLAB함수 meshgrid.m을 사용할 수도 있으나, 이 방법에서는 격자를 전치(transpose)시켜야하는 문제점이 있다. MATLAB함수 ndgrid.m의 사용법은 다음과 같다.

```
>> [X1,X2, ... ,Xn] = ndgrid(x1gv,x2gv, ... ,xngv)
>> [X1,X2, ... ,Xn] = ndgrid(xgv)
```

여기서 x1gv, x2gv,$\cdots$, xngv와 xgv는 각 좌표계에 해당하는 격자벡터(grid vector)이다. 예를 들어, X 좌표가 $\{1,2,3,4,5\}$이고 Y 좌표가 $\{11,22,33\}$인 격자를 만들기 위해서는 다음과 같은 MATLAB명령문들을 사용한다.

```
>> xgv = 1:5, ygv = [11 22 33], [X,Y] = ndgrid(xgv,ygv)
```

다른 예로서, X 좌표가 $\{1,2,3,4,5\}$이고 Y 좌표 역시 $\{1,2,3,4,5\}$인 격자를 만들기 위해서는 다음과 같은 MATLAB명령문들을 사용한다.

```
>> xygv = 1:5,  [X,Y] = ndgrid(xygv)
```

두 번째 단계로, 이 격자 상의 각 점에 대응하는 함수값을 계산한다. 예를 들어, 다음과 같은 MATLAB명령문을 사용할 수 있다.

```
>> V = X.* exp(-X.^2 - Y.^2);
```

또한, 다음과 같은 MATLAB명령문들을 사용할 수 있다.

```
>> myftn = @(u,v) u.^2*exp(-u.^2-v.^2), V = myftn(X,Y)
```

세 번째 단계로, MATLAB함수 griddedInterpolant.m를 사용해서 내삽을 한다. 이 MATLAB함수의 사용법을 알기 위해서는 다음 명령문들을 실행해보라.

```
>> showdemo griddedInterpolantDemo
>> doc griddedInterpolant
```

이에 대한 자세한 내용은 다음 웹주소를 참조하라.

```
http://www.mathworks.com/help/techdoc/ref/griddedinterpolantclass.html
```

격자 $[X, Y]$ 상에서 정의되는 함수값들 V 를 내삽하기 위해서는 다음 명령문을 실행하라.

```
>> F = griddedInterpolant(X,Y,V,'cubic')
```

여기서 사용된 'cubic'은 등간격으로 구성된 격자에만 적용할 수 있는 내삽법으로서 3차멱함수를 사용한다. 다른 내삽법으로는 가장 가까이 이웃한 점의 값을 사용하는 'nearest', 선형함수를 사용하는 'linear', 스플라인을 사용하는 'spline', 단변수함수의 내삽에만 적용할 수 있으며 구분화된 3차함수를 사용하는 'pchip'이 있다.

네 번째 단계로, 내삽된 값들을 사용하기 위해서는 F를 함수로 간주하면 된다. 예를 들어, 점 $[1.2345, 2.3456]$의 내삽값을 구하려면(query), 다음과 같은 명령문을 실행하라.

```
>> F(1.2345,2.3456)
```

만약 F좌표가 $\{1.11, 1.22\}$이고 Y 좌표가 $\{2.11, 2.22, 2.33\}$인 격자점들에서 내삽값들을 알려면, 다음과 같은 명령문을 실행하라.

```
>> F([1.11 1.22], 2.11:0.11:2.33)
```

만약 F에서 지정된 격자 [X,Y] 밖의 점에 대한 내삽값을 질문하면(query), 그 답은 'NaN'이다.

예제 3.6.1 MATLAB함수 griddedInterpolant.m를 사용해서 내삽하는 예로서, 다음 MATLAB프로그램 GridInterpolation101.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename: GridInterpolation101.m
% Grid-Based Interpolation 1
% Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, clf
% 1. Generating Values
% 1.1. MATLAB function peaks.m
GenerateValues = @(X,Y)(3*(1-X).^2 .*exp(-(X.^2) - (Y+1).^2) ...
             -10 * (X/5 - X.^3 - Y.^5) .* exp(-X.^2 - Y.^2) ...
             -1/3 * exp(-(X+1).^2 - Y.^2));
% 1.2. Creating the grid from a pair of grid vectors
xgv = -1.5:0.25:1.5;
```

```
ygv = -3:0.5:3;
[X,Y] = ndgrid(xgv,ygv);
% 1.3. Generating
V = GenerateValues(X,Y);
% 1.4. 3D plot
subplot(2,2,1)
surf(X,Y,V)
set(gca,'fontsize',11,'fontweigh','bold')
% ------------------------------------------------------------------
% 2. Interpolation
% 2.1. Cubic Interpolation
F = griddedInterpolant(X,Y,V,'cubic')
GVproperty = F.GridVectors
GV1 = F.GridVectors{1}
GV2 = F.GridVectors{2}
Imethod = F.Method
First2x3values = F.Values(1:4,1:3)
OddValues = F.Values(1:2:end,1:2:end)
% 2.2. Querying the Interpolant
% Example 1
Fvalue1 = F(0.9876,2.3456)
TrueValue1 = GenerateValues(0.9876,2.3456)
error1 = Fvalue1 - TrueValue1
% Example 2
XYvalue2 = -1 + 0.5*rand(5,2)
Fvalue2 = F(XYvalue2)
% Example 3
XYvalue3 = 0.5*rand(5,2)
Fvalue3 = F(XYvalue3(:,1),XYvalue3(:,2))
% Example 4
Fvalue4 = F(100,100)
% ------------------------------------------------------------------
% 3. Refine the surface
% 3.1. Refining Grid
xgvR = -1.5:0.1:1.5;
ygvR = -3:0.1:3;
[XR,YR] = ndgrid(xgvR,ygvR);
% 3.2. Creating a Refining Surface
VR = F(XR,YR);
% 3.3. 3D plot
subplot(2,2,2)
surf(XR,YR,VR)
set(gca,'fontsize',11,'fontweigh','bold')
% ------------------------------------------------------------------
% 4. Change the Interpolation Mehtod
% 4.1. Change the Method
F.Method = 'spline'
VS = F(XR,YR);
% 4.2. 3D plot
subplot(2,2,3)
surf(XR,YR,VS)
set(gca,'fontsize',11,'fontweigh','bold')
% ------------------------------------------------------------------
% 5. Diference by Methods
DifferMethods = VR-VS;
DiffMeth = max(max(abs(DifferMethods),[],1),[],2)
subplot(2,2,4)
surf(XR,YR,DifferMethods)
set(gca,'fontsize',11,'fontweigh','bold')
saveas(gcf,'GridInterpolation101.jpg')
% End of program
```

```
75 % --------------------------------------------------------------------
```

MATLAB프로그램 GridInterpolation101.m을 실행하면, MATLAB의 내장함수 peaks.m을 내삽한다. 이 peaks함수값을 나타내는 변수 GenerateValues는 다음과 같다.

$$f(x,y) = 3[1-x]^2 e^{-x^2-y^2} - 10\left[\frac{x}{5} - x^3 - y^5\right] e^{-x^2-y^2} - \frac{1}{3}e^{-[x+1]^2-y^2} \tag{1}$$

이 MATLAB프로그램의 첫 번째 단계에서는 첫 번째 격자벡터 xgv를 $[-1.5 : 0.25 : 1.5]$로 하고 두 번째 격자벡터 ygv를 $[-3 : 0.5 : 3]$로 하는 격자 [X,Y]를 만든다. 이때 MATLAB함수 ndgrid.m을 사용한다. 식 (1)의 GenerateValues를 사용해서, 이 격자에서 함수값들 V를 계산한다. 이렇게 계산된 표면 [X,Y,V]가 그림 3.6.1의 좌측상단 그래프이다.

두 번째 단계에서는 첫 번째 단계에서 구한 표면 [X,Y,V]을 내삽하기 위해서 다음과 같은 명령문을 수행한다.

```
>> F = griddedInterpolant(X,Y,V,'cubic')
```

이 MATLAB명령문이 실행한 결과물은 다음과 같으며, 내삽함수가 F에 저장된다.

```
F = griddedInterpolant


  Properties:
    GridVectors: {2x1 cell}
         Values: [13x13 double]
         Method: 'cubic'
  Methods
```

내삽함수 F에 포함된 격자벡터의 성질과 각 격자벡터를 출력하기 위한 MATLAB명령문들은 다음과 같다.

```
>> GVproperty = F.GridVectors
>> GV1 = F.GridVectors{1}
>> GV2 = F.GridVectors{2}
```

또한, 이 명령문들을 실행한 결과는 다음과 같다.

```
GVproperty =
    [1x13 double]
    [1x13 double]
```

```
GV1 =
  Columns 1 through 12
   -1.5000   -1.2500   -1.0000   -0.7500   -0.5000   -0.2500
         0    0.2500    0.5000    0.7500    1.0000    1.2500
  Column 13
    1.5000

GV2 =
  Columns 1 through 12
   -3.0000   -2.5000   -2.0000   -1.5000   -1.0000   -0.5000
         0    0.5000    1.0000    1.5000    2.0000    2.5000
  Column 13
    3.0000
```

내삽함수 F에 사용한 내삽방법을 출력하기 위해서는 F.Method를 사용하고, 행렬 V의 원소들을 출력하기 위해서는 F.Values를 사용한다. 예를 들어, 다음 MATLAB명령문들을 실행해보자.

```
>> Imethod = F.Method
>> First4x3values = F.Values(1:4,1:3)
>> OddValues = F.Values(1:2:end,1:2:end)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
Imethod = cubic

First4x3values =
    0.0042    0.0028    0.0452
   -0.0050   -0.0671   -0.1285
   -0.0299   -0.2346   -0.5921
   -0.0752   -0.5260   -1.4478

OddValues =
   0.0042  0.0452  0.3007 -2.7736 -0.8639  0.5539  0.0312
  -0.0299 -0.5921  1.8559 -1.6523  0.2289  2.0967  0.1099
  -0.1373 -2.6390  2.2247  1.4796  2.7942  4.5569  0.2335
  -0.2450 -4.7596 -0.7239  0.9810  3.6886  5.8591  0.2999
  -0.2228 -4.3468 -2.2222  0.3744  2.9344  4.5675  0.2336
  -0.1100 -2.1024 -0.2729  2.9369  2.4338  2.2099  0.1107
  -0.0298 -0.5293  0.8834  3.2695  1.5813  0.6771  0.0320
```

격자 [X,Y]에 포함되지 않은 점에서 내삽값을 구하기 위해서는 다음과 같이 내삽함수 F를 사용한다. 내삽함수 F를 사용하는 MATLAB명령문들은 다음과 같다.

```
>> Fvalue1 = F(0.9876,2.3456)
>> TrueValue1 = GenerateValues(0.9876,2.3456)
>> error1 = Fvalue1 - TrueValue1
>> XYvalue2 = -1 + 0.5*rand(5,2)
>> Fvalue2 = F(XYvalue2)
>> XYvalue3 = 0.5*rand(5,2)
>> Fvalue3 = F(XYvalue3(:,1),XYvalue3(:,2))
>> Fvalue4 = F(100,100)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
Fvalue1 =  1.1399
TrueValue1 =  1.1038
error1 =  0.0361

XYvalue2 =
   -0.9811   -0.8691
   -0.5574   -0.8323
   -0.5434   -0.6601
   -0.6019   -0.9317
   -0.9506   -0.6394

Fvalue2 =
    1.8882
    3.0642
    3.4437
    2.7250
    1.5629

XYvalue3 =
    0.0534    0.4519
    0.3269    0.4455
    0.2471    0.1671
    0.3895    0.3494
    0.3575    0.0989

Fvalue3 =
```

```
    0.2685
   -0.0274
    0.0041
   -0.0564
    0.0791

Fvalue4 =  NaN
```

세 번째 단계에서는 좀 더 세밀한 격자에서 내삽된 표면을 그리기 위해서, 첫 번째 격자벡터 xgvR을 $[-1.5 : 0.1 : 1.5]$로 하고 두 번째 격자벡터 ygvR을 $[-3 : 0.5 : 3]$로 하는 격자 [XR,YR]을 만든다. 이때 MATLAB함수 ndgrid.m을 사용한다. 내삽함수 F를 사용해서, 이 격자에서 함수값들 VR을 계산한다. 이들을 수행하기 위한 MATLAB명령문들은 다음과 같다.

```
>> xgvR = -1.5:0.1:1.5;
>> ygvR = -3:0.1:3;
>> [XR,YR] = ndgrid(xgvR,ygvR);
>> VR = F(XR,YR);
>> subplot(2,2,2)
>> surf(XR,YR,VR)
```

이렇게 계산된 표면 [XR,YR,VR]이 그림 3.6.1의 우측상단 그래프이다.

네 번째 단계에서는, 내삽법으로 'cubic'이 아닌 'spline'을 사용해서 내삽을 한 다음, 그 표면을 그린다. 이를 위해서 MATLAB명령문들을 실행하면, 그림 3.6.1의 좌측하단 그래프가 그려진다.

```
>> F.Method = 'spline'
>> VS = F(XR,YR);
>> subplot(2,2,3)
>> surf(XR,YR,VS)
```

다섯 번째 단계에서는, 내삽법으로 'cubic'을 사용한 결과와 'spline'을 사용한 결과의 차이를 조사한다. 이 두 내삽법들에 차이의 최대값은 0.2273이고, 그 차이를 그린 것이 그림 3.6.1의 우측하단 그래프이다. ■

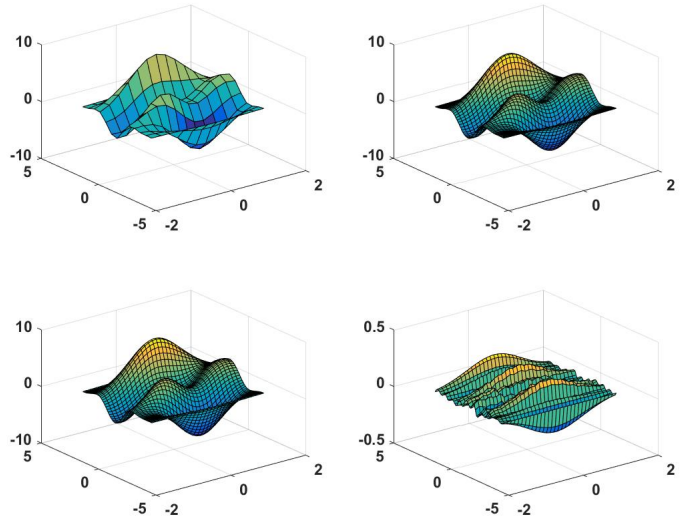


그림 3.6.1. MATLAB함수 griddedInterpolant를 사용한 내삽

제 4 장

스플라인

> Spline functions, and, more generally, piecewise polynomial functions are the most successful approximating functions in use today. They combine ease of handling in a computer with great flexibility, and are therefore particularly suited for the approximation of experimental data or design curve measurements.
>
> Carl de Boor and John R . Rice [18]

제4.1절 구분적 보간

지금까지는 구간 $[a,b]$ 전체에서 함수 f를 멱함수 하나로 보간하거나 추정하였다. 그러나, 현실적인 문제를 풀기 위해서는 함수 f를 소구간에 따라 다른 멱함수로 추정할 필요가 있다. 이 절에서는 구간 $[a,b]$의 분할 $\Pi \doteq \{a = x_1 < x_2 < \cdots < x_n = b\}$의 각 소구간에서 평활한 함수, 즉 구분적으로(piecewise) 평활한 함수를 사용해서 함수 f를 보간하는 문제를 생각해보자. 이에 대한 고전문헌으로는 Reinsch [47]가 있다.

가장 간단한 구분적 보간은 소구간 $\left[\frac{x_{i-1}+x_i}{2}, \frac{x_i+x_{i+1}}{2}\right)$에서 함수값을 $y_i \doteq f(x_i)$ 로 하는 것이다. 즉, 계단함수를 사용하는 것이다. 이 계단함수를 0차(zero-order) 보간함수(interpolant)라 하고, $\hat{f}_0$로 표기하자. 예제들을 바탕으로 이 방법을 살펴보자.

예제 4.1.1 다음과 같은 Runge함수 f를 살펴보자.

$$f(x) = \frac{1}{1+x^2} \tag{1}$$

구간 $[-5,5]$에서 이 Runge함수의 0차 보간함수를 그리기 위해서, 다음 MATLAB프로그램 NearestInterpolant101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: NearestInterpolant101.m
% Zero-order Interpolant with n = 10 using m-file interp1
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
runge = inline('1./( 1+x.^2 )','x');
x1 = [-5:1:5]' ;    % x(i-1)
y1 = runge(x1);     % y(i-1)
xx = -5:0.01:5;
yy = interp1(x1,y1,xx,'nearest');
ytrue = runge(xx);
plot(xx,ytrue,'k-', xx,yy,'r--',x1,y1,'ko','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xtick',-5:1:5)
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
legend('original function','interpolant','knots')
axis( [ -5 5 0 1.2 ])
saveas(gcf,'NearestInterpolant101.jpg')
save('NearestInterpolant101.txt','xx','yy','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 NearestInterpolant101.m을 실행하면, 그림 4.1.1을 출력한다. 이 MATLAB프로그램에서는 MATLAB함수 interp1.m에 모수로 'nearest'를 지정하였다. 이는 0차 보간함수를 출력하기 위한 것이다. ■

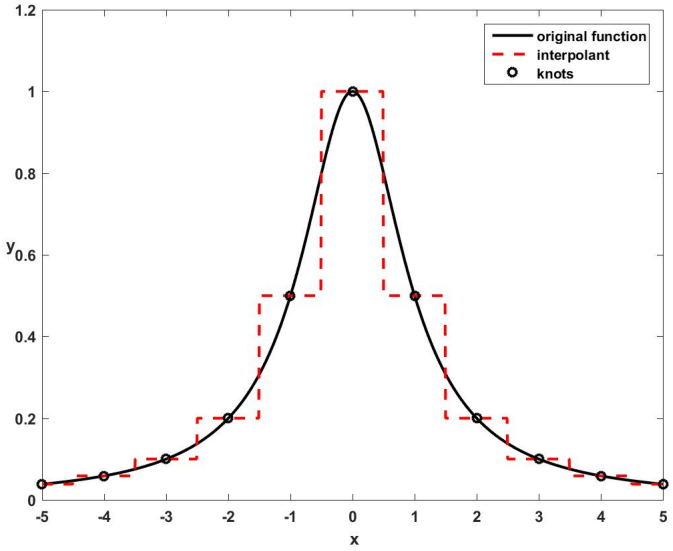


그림 4.1.1. 스텝보간함수

예제 4.1.2 Python을 사용해서 예제 4.1.1을 다시 다루기 위해서, 다음 Python프로그램 NearestInterpolant101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

runge = lambda x: 1/( 1+x**2 );
x1 = np.linspace(-5,5,11);
y1 = runge(x1);
fnearest = interp1d(x1, y1, kind='nearest')
xx = np.linspace(-5,5,1001);
yy = fnearest(xx);
ytrue = runge(xx);

# Plotting
fig = plt.figure()
plt.plot(xx,ytrue,'k-', lw=2, label='original function')
plt.plot(xx,yy,'r--', lw=2, label='interpolant')
plt.plot(x1,y1,'ko', lw=2, label='knots')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([-5, 5, 0, 1.2 ])
plt.show()
fig.savefig('NearestInterpolant101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 4.1.1의 결과와 같다. ■

각 소구간에서 함수 f를 직선으로 보간한 것을 1차 보간함수라고 한다. 즉, 1차 보간함수는 다음과 같은 구분적 선형함수 $\hat{f}_1$를 사용하는 것이다.

$$\hat{f}_1(x) = \sum_{i=1}^{n-1} \left\{ y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i} [x - x_i] \right\} 1_{[x_i, x_{i+1})}(x) \tag{4.1.1}$$

이를 선형보간함수라고도 부른다.

예제 4.1.3 다음과 같은 Runge함수를 살펴보자.

$$f(x) = \frac{1}{1 + x^2} \tag{1}$$

구간 $[-5, 5]$에서 이 Runge함수의 선형보간함수를 그리기 위해서, 다음 MATLAB프로그램 LinearInterpolant101.m을 실행하라.

```
% --------------------------------------------------------------------
```

```
% Filename: LinearInterpolant101.m
% Linear Interpolant with n = 10
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
runge = inline('1./( 1+x.^2 )','x');
x1 = [-5:1:5]' ;    % {x(i-1)}
y1 = runge(x1);     % {y(i-1)}
% Plotting
plot(x1,y1,'k')
xx = -5:0.01:5;
ytrue = runge(xx);
plot(xx,ytrue,'k-', x1,y1,'r--',x1,y1,'ko','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xtick',-5:1:5)
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
legend('original function','linear interpolant','knots')
axis( [ -5 5 0 1.2 ])
saveas(gcf,'LinearInterpolant101.jpg')
save('LinearInterpolant.txt','xx','ytrue','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 LinearInterpolant101.m을 실행하면, 그림 4.1.2를 출력한다. 그림 4.1.2에서 알 수 있듯이, 함수 $f(x)$와 선형보간함수는 보간노드들을 직선으로 연결한 구분적 선형함수이다. ■

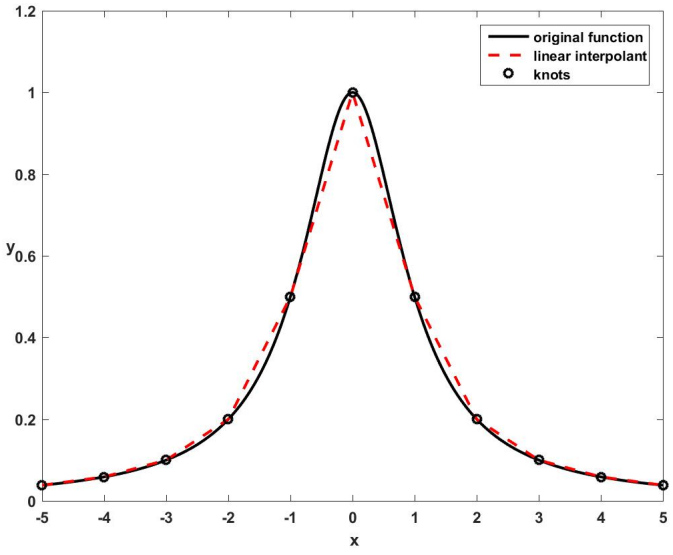


그림 4.1.2. 선형보간함수

예제 4.1.4 Python을 사용해서 예제 4.1.3을 다시 다루기 위해서, 다음 Python프로그램 LinearInterpolant101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

runge = lambda x: 1/( 1+x**2 );
x1 = np.linspace(-5,5,11);
y1 = runge(x1);
flinear = interp1d(x1, y1, kind='linear')
xx = np.linspace(-5,5,1001);
yy = flinear(xx);
ytrue = runge(xx);

# Plotting
fig = plt.figure()
plt.plot(xx,ytrue,'k-', lw=2, label='original function')
plt.plot(xx,yy,'r--', lw=2, label='interpolant')
plt.plot(x1,y1,'ko', lw=2, label='knots')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([-5, 5, 0, 1.2 ])
plt.show()
fig.savefig('LinearInterpolant101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 4.1.3의 결과와 같다. ■

만약 분할 Π의 각 점 에서 함수값 $f(x_i)$ 뿐 아니라 미분값 $f'(x_i)$를 알고 있다면, 다음과 같은 구분적 Hermite보간함수 $\hat{f}_2(x)$를 사용할 수 있다.

$$\hat{f}_2(x) = \sum_{i=1}^{n-1} \left[f(x_i)\varphi_{i,1}(x) + f(x_{i+1})\varphi_{i,3}(x) + f'(x_i)\varphi_{i,2}(x) + f'(x_{i+1})\varphi_{i,4}(x) \right] \tag{4.1.2}$$

여기서 각 함수는 다음과 같다.

$$\varphi_{i,1}(x) \doteq 1 - 3\left[\frac{x-x_i}{x_{i+1}-x_i}\right]^2 + 2\left[\frac{x-x_i}{x_{i+1}-x_i}\right]^3 \tag{4.1.3}$$

$$\varphi_{i,2}(x) \doteq [x-x_i]\left\{\left[\frac{x-x_i}{x_{i+1}-x_i}\right] - 1\right\}^2 \tag{4.1.4}$$

$$\varphi_{i,3}(x) \doteq \left[\frac{x-x_i}{x_{i+1}-x_i}\right]^2 \left\{3 - 2\left[\frac{x-x_i}{x_{i+1}-x_i}\right]\right\} \tag{4.1.5}$$

$$\varphi_{i,4}(x) \doteq [x_{i+1}-x_i]\left[\frac{x-x_i}{x_{i+1}-x_i}\right]^2 \left\{\left[\frac{x-x_i}{x_{i+1}-x_i}\right] - 1\right\} \tag{4.1.6}$$

이 함수 $\hat{f}_2$는 구간 $[a,b]$의 거의 모든 점에서 3차 멱함수이고 또한 $C^1[a,b]$급이다.

예제 4.1.5 다음과 같은 Runge함수 f를 살펴보자.

$$f(x) = \frac{1}{1+x^2}$$

구간 $[-5,5]$에서 이 Runge함수의 구분적 Hermite보간함수를 그리기 위해서, 다음 MATLAB 프로그램 CubicHermiteSpline101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename CubicHermiteSpline101.m
% Cubic Hermite Spline with n = 10 using m-file interp1
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
runge = inline('1./( 1+x.^2 )','x');
x1 = [-5:1:5]' ;    % x(i-1)
y1 = runge(x1);     % y(i-1)
xx = -5:0.02:5;
yy = interp1(x1,y1,xx,'pchip');
ytrue = runge(xx);
err = max(abs(yy-ytrue))
plot(xx,ytrue,'k-', xx,yy,'r--',x1,y1,'ko','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xtick',-5:1:5)
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
legend('original function','cubic Hermite','knots')
axis( [ -5 5 0 1.2 ])
saveas(gcf,'CubicHermiteSpline101.jpg')
save('CubicHermiteSpline101.txt','xx','yy','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 CubicHermiteSpline101.m에서는 MATLAB함수 interp1에 모수로 'pchip'를 지정하였다. 이는 구분적 Hermite보간함수를 출력하기 위한 것이다.

이 MATLAB프로그램을 실행하면, 구간 $[-5,5]$에서 함수 $f(x)$와 구분적 Hermite보간함수 사이의 최대오차 $\max|f(x)-\hat{f}_2(x)|$는 0.0178임을 알 수 있다. 또한, 이 MATLAB프로그램을 실행하면, 그림 4.1.3을 출력한다. 그림 4.1.3에서 Runge함수 $f(x)$는 흑색 실선으로, 그리고 구분적 Hermite보간함수는 적색 긴점선으로 그려져 있다. 이 구분적 Hermite보간함수는 보간노드들을 3차 멱함수로 연결한 함수이며, 이 구분적 Hermite보간함수의 1차 도함수는 각 점에서 연속임을 상기하라. ■

예제 4.1.6 Python을 사용해서 예제 4.1.5을 다시 다루기 위해서, 다음 Python프로그램 CubicHermiteSpline101.Py를 실행해 보자.

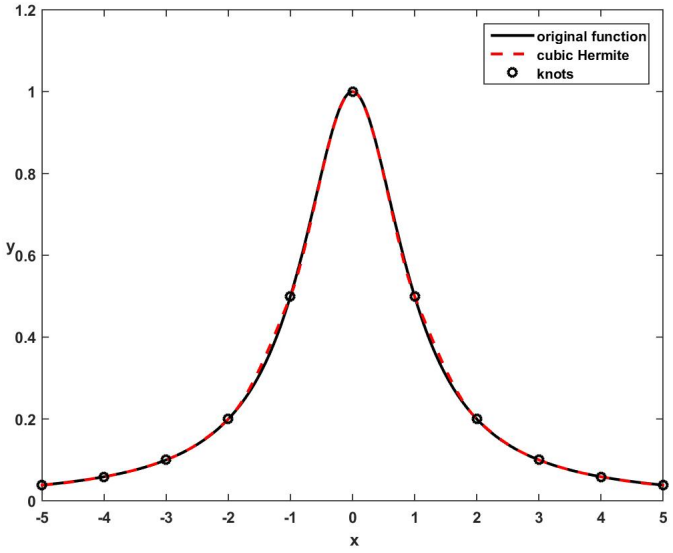


그림 4.1.3. 구분적 Hermite보간함수

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

runge = lambda x: 1/( 1+x**2 );
x1 = np.linspace(-5,5,11);
y1 = runge(x1);
fnearest = interp1d(x1, y1, kind='cubic')
xx = np.linspace(-5,5,1001);
yy = fnearest(xx);
ytrue = runge(xx);

# Plotting
fig = plt.figure()
plt.plot(xx,ytrue,'k-', lw=2, label='original function')
plt.plot(xx,yy,'r--', lw=2, label='interpolant')
plt.plot(x1,y1,'ko', lw=2, label='knots')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([-5, 5, 0, 1.2 ])
plt.show()
fig.savefig('CubicHermiteSpline101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 4.1.5의 결과와 같다. ■

만약 멱함수를 사용해서 보간 (interpolation) 을 한다면, 주어진 n 개 점들을 보간하기

위해서 $[n-1]$차 멱함수를 사용해야만 한다. 멱함수의 차수가 커지면 멱함수는 크게 변동한다. 더구나, 만약 연립선형방정식을 사용해서 보간함수의 계수를 구한다면, n이 큰 경우에 계수행렬은 악조건(ill-conditioned)을 이루는 것이 일반적이다. 따라서, 작은 반올림오차에 의해서도 결과가 크게 잘못될 수도 있다. 따라서, 전구간에서 멱함수 1개 만을 사용하는 것보다는 구분적 멱함수를 사용해서 보간하는 것이 더 좋은 경우가 많다. 특히, 국소적으로 급격하게 변동하는 함수를 나타내는데 멱함수보다 구분적 멱함수가 좋다. 다음 예제를 살펴보자.

예제 4.1.7 다음과 같은 계단함수를 살펴보자.

$$f(x) = \begin{cases} -1, & (x < 0), \\ 0, & (x = 0), \\ 1, & (x > 0). \end{cases}$$

구간 $[-1, 1]$에서 이 계단함수의 멱함수로 된 보간함수를 그리기 위해서, 다음 MATLAB 프로그램 PolynomialInterpolation101.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename: PolynomialInterpolation101.m
% Polynomial or Piecewise Polynomial
% Programmed by CBS
% ------------------------------------------------------------------
function PolynomialInterpolation101
clear all, close all
xtrue = -1:0.01:1;
ytrue = [ -ones(1,100),0, ones(1,100) ];
plot(xtrue,ytrue,'k.','LineWidth',1.5)
set(gca,'fontsize',11,'fontweigh','bold')
hold all
plot([0 0],[-1 1],'k','LineWidth',2)
xnew = -1:0.02:1;
for mm = 2:1:7
    xold = [1-mm:2:mm-1 ]/mm;
    if mod(mm,2) == 0
       yold = [ -ones(1,mm/2), ones(1,mm/2) ];
    else
       yold = [ -ones(1,(mm-1)/2), 0 , ones(1,(mm-1)/2) ];
    end
    ynew = PolynomialInterp(xold,yold,xnew);
    plot(xold,yold,'k.',xnew,ynew,'LineWidth',2.0)
end
hold off
saveas(gcf,'PolynomialInterpolation101.jpg')
save('PolynomialInterpolation101.txt','xnew','ynew','-ascii')
end
% ------------------------------------------------------------------
function ynew = PolynomialInterp(xold,yold,xnew)
%  ynew = PolynomialInterp(x,y,u) is ynew(i) = P(xneew(i))
%  where P is the polynomial of degree d = length(x)-1
%        satisfying  P(x(k)) = y(k).
```

```
nn = length(xold);
ynew = zeros(size(xnew));
for ndum = 1:nn
   ydum = ones(size(xnew));
   for ii = [1:ndum-1 ndum+1:nn]
       ydum = (xnew-xold(ii))./(xold(ndum)-xold(ii)).*ydum;
   end
   ynew = ynew + ydum*yold(ndum);
end
end
% End of program
% --------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 구간 $[-1, 1]$에서 함수 $f(x)$의 멱함수로 된 보간함수들을 계산해서 그림 4.1.4을 출력한다. 그림 4.1.4에서 알 수 있듯이, 멱함수의 차수가 커지면 보간함수의 변동이 커져서 원래 함수로부터 크게 이탈한다. ■

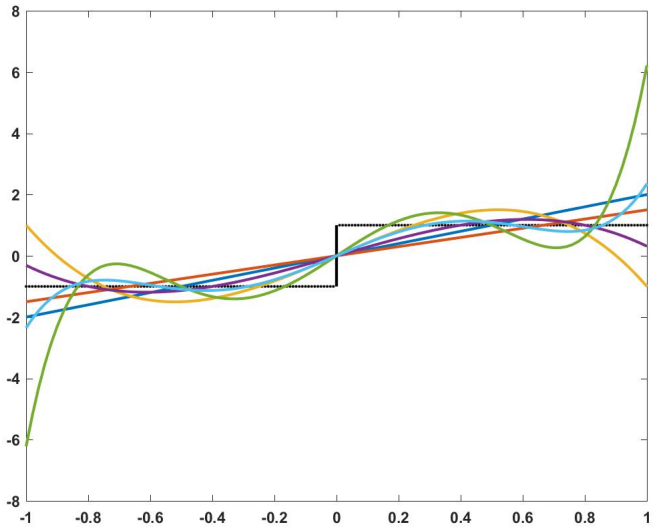


그림 4.1.4. 멱함수로 만들어진 보간함수들

예제 4.1.8 Python을 사용해서 예제 4.1.7을 다시 다루기 위해서, 다음 Python프로그램 PolynomialInterpolation101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

def PolynomialInterp(xold,yold,xnew):
```

```
# ynew = PolynomialInterp(x,y,u) is ynew(i) = P(xneew(i))
# where P is the polynomial of degree d = length(x)-1
#       satisfying  P(x(k)) = y(k).
    nn = len(xold);
    ynew = np.transpose([0.0]*len(xnew));
    for ndum in range(0,nn):
        ydum = np.transpose([1.0]*len(xnew));
        for ii1 in range(0,ndum):
            ydum = (xnew-xold[ii1])/(xold[ndum]-xold[ii1])*ydum;
        for ii2 in range(ndum+1,nn):
            ydum = (xnew-xold[ii2])/(xold[ndum]-xold[ii2])*ydum;
        ynew = ynew + ydum*yold[ndum];
    return ynew

# Polynomial Interpolation
xtrue = np.arange(-1,1.01,0.01);
ydum1 = np.transpose([1.0]*100)
ytrue = np.hstack((-ydum1,0,ydum1))

# Plotting
fig = plt.figure()
plt.plot(xtrue,ytrue,'k.',lw=2)
plt.hold(True); plt.plot([0,0],[-1,1],'k',lw=2)
xnew = np.arange(-1,1.01,0.02);
for mm in range(2,8):
    xold = np.arange(1-mm,mm,2)/mm;
    if np.mod(mm,2) == 0:
       ydum2 = np.transpose([1.0]*round(mm/2))
       yold = np.hstack((-ydum2,ydum2));
    else:
       ydum3 = np.transpose([1.0]*round((mm-1)/2))
       yold = np.hstack((-ydum3,0,ydum3));
    ynew = PolynomialInterp(xold,yold,xnew);
    plt.hold(True); plt.plot(xold,yold,'k.',lw=2)
    plt.hold(True); plt.plot(xnew,ynew,'k.',lw=2)
plt.axis([ -1, 1, -8, 8 ])
plt.show()
fig.savefig('PolynomialInterpolation101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 4.1.7의 결과와 같다. ■

제4.2절 스플라인

요즈음에는 운형자 (French curve) 라는 단어를 아는 사람이 별로 없다. 운형자는 설계를 하는 사람들이 곡선을 그리기 위해서 오랫동안 사용한 도구이다. 스플라인 (spline 또는 flexible cuvre) 이란 운형자가 존재하기 전부터 여러 점들을 부드럽게 연결하기 위해서 사용한 유연한 판이나 줄을 의미한다. 그림 4.2.1에서 스플라인에서 운형자로 넘어가는 과정을 짐작할 수 있다. 그림 4.2.1는 https://en.wikipedia.org/wiki/Flat_spline에서 인용한 것이다.

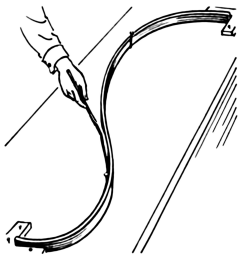


그림 4.2.1. 평평한 스플라인(flat spline)

4.2.1 스플라인함수의 정의

제4.1절에서 설명했듯이, 각 소구간마다 계수가 다른 멱함수를 이어서 만든 구분적 멱함수를 스플라인함수(spline function)라 한다. 스플라인함수를 설명하기 위해서, 다음과 같이 구간 $(-\infty, \infty)$의 분할 Π_∞를 정의하자.

$$\Pi_\infty \doteq \{\cdots < x_{-n} < \cdots < x_{-1} < x_0 < x_1 < \cdots < x_n < \cdots\} \tag{4.2.1}$$

여기서 x_i는 마디점이다.

정의 4.2.1

다음 조건들을 만족하는 함수 $S(x)$를 m차 스플라인함수라 한다.

(a) 각 소구간 (x_i, x_{i+1})에서 함수 $S(x)$는 m차 이하인 멱함수이다.

(b) 도함수들 $S, S^{(1)}, S^{(2)}, \cdots, S^{(m-1)}$은 각 마디점 x_i에서 연속이다.

스플라인함수에 관한 이론은 방대하다. 본서에서는 재무학과 경제학에서 필요한 수준 만을 다루고자 한다. 그러나 이 분야에서 앞으로 얼마나 깊게 스플라인함수를 다룰지는 모르겠다. 최근 Fengler & Hin [21]은 B-스플라인을 사용해서 내재변동성곡면(implied volatility surface)를 추정하였다. 스플라인에 대한 원고를 작성할 때 de Boor [16]를 가장 많이 참조하였다.

4.2.2 절단형 표현

다음과 같이 절단함수를 사용해서 스플라인함수를 나타낼 수 있다.

정리 4.2.1

제m차 스플라인함수 $S(x)$를 다음과 같이 멱함수와 절단멱함수들의 선형결합으로 나타낼 수 있다.

$$S(x) = p_m(x) + \sum_{i=1}^{n} c_i[x - x_i]_+^m$$

여기서 $A_+ \doteq \max\{A, 0\}$이고, $p_m(x)$는 m차 멱함수이며, $c_1, c_2, \cdots, c_n$은 상수들이다. 이 멱함수 $p_m(x)$와 상수들 $c_1, c_2, \cdots, c_n$는 일의적으로 정해진다.

증명. 이 표현의 존재성에 관해서는 de Boor [16, 제VIII장]를 참조하라. 여기서는 일의성만을 증명하기로 하자.

소구간 $[x_i, x_{i+1}]$에서 정해지는 m차 스플라인함수를 $S_{m,i}(x)$로 표기하자. 마디점 x_i에서 $[m-1]$차까지 도함수들이 연속이므로, 다음 식을 만족하는 c_i가 존재한다.

$$S_{m,i}(x) - S_{m,i-1}(x) = c_i[x - x_i]^m \tag{1}$$

각 $r(< n)$에 대해 소구간 $[x_r, x_{r+1}]$ 속하는 각 점 x에서 다음 식들이 성립한다.

$$\begin{aligned} S_{m,r}(x) &= S_{m,0}(x) + \sum_{i=1}^{r} [S_{m,i}(x) - S_{m,i-1}(x)] \\ &= S_{m,0}(x) + \sum_{i=1}^{r} c_i[x - x_i]^m = S_{m,0}(x) + \sum_{i=1}^{r} c_i[x - x_i]_+^m \end{aligned} \tag{2}$$

여기서 마지막 등호는 각 $x(\geq x_r)$에 대해서 식 $x - x_i = [x - x_i]_+$이 성립하기 때문이다. 각 $x\,(\leq x_{r+1})$에 대해서 다음 식이 성립한다.

$$\sum_{i=r+1}^{n} c_i[x - x_i]_+^m = 0 \tag{3}$$

식 (2)와 식 (3)에서 알 수 있듯이, 다음 식이 성립한다.

$$S_{m,r}(x) = S_{m,0}(x) + \sum_{i=1}^{n} c_i[x - x_i]_+^m \tag{4}$$

소구간 $[x_0, x_1]$에서 정해지는 m차 스플라인함수를 $S_{m,0}(x)$는 일의적으로 정해진다. 따라서,

다음 식이 성립한다.

$$p_m(x) = S_{m,0}(x) \tag{5}$$

식 (1)에서 알 수 있듯이, 다음 식이 성립한다.

$$S_{m,i}^{(m)}(x) - S_{m,i-1}^{(m)}(x) = c_i m! \tag{6}$$

따라서, 다음 식들이 성립한다.

$$\begin{aligned} c_i &= \frac{1}{m!}\left[S_{m,i}^{(m)}(x_i) - S_{m,i-1}^{(m)}(x_i)\right] \\ &= \frac{1}{m!}\left[S_{m,i}^{(m)}(x_i+0) - S_{m,i-1}^{(m)}(x_i-0)\right] \\ &= \frac{1}{m!}\left[S^{(m)}(x_i+0) - S^{(m)}(x_i-0)\right] \end{aligned} \tag{7}$$

식 (7)에서 알 수 있듯이, 각 i에 대해서 c_i는 일의적으로 정해진다. ■

정리 4.2.1의 스플라인 표현법을 이용하기 위해서는 멱함수 $p_m(x)$의 계수들 $[m+1]$개와 계수들 $c_1, c_2, \cdots, c_n$을 구하기 위해서는 미지수들이 $[m+n+1]$개인 연립1차방정식을 풀어야 한다. 만약 n이 크다면, 이 연립방정식은 악조건(ill-conditioned)이다. 따라서, 정리 4.2.1의 절단멱함수를 사용한 스플라인 표현법은 실제 문제를 푸는데 잘 사용되지 않는다. 그러나, 스플라인 이론을 전개하는데 이 표현법은 중요하다.

4.2.3 자연스플라인

제$[2k-1]$차 스플라인함수가 소구간 $(-\infty, x_{-n}]$과 소구간 $[x_n, \infty)$에서 $[2k-1]$차가 아닌 $[k-1]$차 멱함수으로 표현되면, 이를 자연스플라인함수(natural spline function)라 부른다. 정리 4.2.1에서 알 수 있듯이, 다음 식을 만족하는 $p_{2k-1}(x)$와 상수들 $c_1, c_2, \cdots, c_n$이 일의적으로 존재한다.

$$S(x) = p_{2k-1}(x) + \sum_{i=1}^{n} c_i [x - x_i]_+^{2k-1} \tag{4.2.2}$$

만약 $x < x_1$이면, 식 (4.2.2)을 다음과 같이 쓸 수 있다.

$$S(x) = p_{2k-1}(x), \quad (x < x_1) \tag{4.2.3}$$

자연스플라인함수의 정의와 식 (4.2.2)에서 알 수 있듯이, 멱함수 $p_{2k-1}(x)$는 $[k-1]$차이다. 따라서, 제$[2k-1]$차 자연스플라인함수 $S(x)$를 다음과 같이 멱함수와 절단멱함수들의 일의적 선형결합으로 나타낼 수 있다.

$$S(x) = P_{k-1}(x) + \sum_{i=1}^{n} c_i [x - x_i]_+^{2k-1} \tag{4.2.4}$$

여기서 $P_{k-1}(x)$는 $[k-1]$차 멱함수이다. 식 (4.2.4)에 대한 자세한 내용은 Greville [26]을 참조하라. 만약 $x > x_n$이면, 식 (4.2.4)을 다음과 같이 쓸 수 있다.

$$S(x) = P_{k-1}(x) + \sum_{i=1}^{n} c_i [x - x_i]^{2k-1}, \quad (x > x_n) \tag{4.2.5}$$

자연스플라인함수의 정의에 의해서 식 (4.2.5)의 $S(x)$는 $[k-1]$차 멱함수이다. 따라서, 각 $r(= 0, 1, \cdots, k-1)$에 대해서 x^{2k-1-r}항의 계수는 0이다. 즉, 다음 식들이 성립한다.

$$\sum_{i=1}^{n} c_i x_i^r = 0, \quad (r = 0, 1, \cdots, k-1) \tag{4.2.6}$$

데이터세트 $\{(x_i, y_i) \,|\, i = 1, 2, \cdots, n\}$과 자연수 $k(\leq n)$가 주어지면, 각 $i\,(= 1, 2, \cdots, n)$에 대해서 x_i를 마디점으로 하고 식 $S(x_i) = y_i$를 만족하는 $[2k-1]$차 자연스플라인함수 $S(x)$가 일의적으로 존재한다. 이에 대한 증명은 Greville [26]을 참조하라.

정리 4.2.2

주어진 데이터세트 $\{(x_i, y_i) \,|\, i = 1, 2, \cdots, n\}$과 자연수 $k(\leq n)$에 대해서, 함수 $S(x)$는 각 $i\,(= 1, 2, \cdots, n)$에 대해서 x_i를 마디점으로 하고 식 $S(x_i) = y_i$를 만족하는 $[2k-1]$차 자연스플라인함수라고 하자. 각 x_i에 대해 식 $f(x_i) = y_i$를 만족하는 C^k급 함수 $f(x)$와 임의의 $a(\leq x_1)$와 $b(\geq x_n)$에 대해 다음 식이 성립한다.

$$\int_a^b [f^{(k)}(x)]^2 dx \geq \int_a^b [S^{(k)}(x)]^2 dx$$

여기서 등호는 식 $f(x) \equiv S(x)$이 성립하는 경우에 한해서 성립한다.

증명. 다음 식이 성립한다.

$$\int_a^b [f^{(k)}(x)]^2 dx = \int_a^b [S^{(k)}(x)]^2 dx + \int_a^b [f^{(k)}(x) - S^{(k)}(x)]^2 dx + 2\int_a^b S^{(k)}(x)\left[f^{(k)}(x) - S^{(k)}(x)\right] dx \tag{1}$$

부분적분을 사용하면, 각 $i(=0,1,\cdots,k-2)$에 대해서 다음 식이 성립함을 알 수 있다.

$$\begin{aligned}&\int_a^b S^{(k+i)}(x)\left[f^{(k-i)}(x) - S^{(k-i)}(x)\right] dx \\ &= S^{(k+i+1)}(b)\left[f^{(k-i)}(b) - S^{(k-i)}(b)\right] - S^{(k+i+1)}(a)\left[f^{(k-i)}(a) - S^{(k-i)}(a)\right] \\ &\quad - \int_a^b S^{(k+i+1)}(x)\left[f^{(k-i-1)}(x) - S^{(k-i-1)}(x)\right] dx\end{aligned} \tag{2}$$

자연스플라인의 정의에 의해서 식 $S^{(k+i+1)}(b) = 0$와 식 $S^{(k+i+1)}(a) = 0$가 성립한다. 따라서, 식 (2)에서 알 수 있듯이 다음 식이 성립한다.

$$\begin{aligned}&\int_a^b S^{(k+i)}(x)\left[f^{(k-i)}(x) - S^{(k-i)}(x)\right] dx \\ &= -\int_a^b S^{(k+i+1)}(x)\left[f^{(k-i-1)}(x) - S^{(k-i-1)}(x)\right] dx\end{aligned} \tag{3}$$

식 (2)와 식 (3)에서 알 수 있듯이, 다음 식이 성립한다.

$$\int_a^b S^{(k)}(x)\left[f^{(k)}(x) - S^{(k)}(x)\right] dx = [-1]^{k-1}\int_a^b S^{(2k-1)}(x)\left[f'(x) - S'(x)\right] dx \tag{4}$$

소구간 (x_j, x_{j+1})에서 도함수 $S^{(2k-1)}(x)$는 상수이다. 이 상수를 α_j로 표기하면, 다음 식이 성립한다.

$$\begin{aligned}&\int_a^b S^{(2k-1)}(x)\left[f'(x) - S'(x)\right] dx \\ &= \sum_{j=1}^{n-1}\alpha_j \int_{x_j}^{x_{j+1}}\left[f'(x) - S'(x)\right] dx \\ &= \sum_{j=1}^{n-1}\alpha_j\left[f\left(x_{j+1}\right) - S\left(x_{j+1}\right)\right] - \sum_{j=1}^{n-1}\alpha_j\left[f\left(x_j\right) - S\left(x_j\right)\right] \\ &= 0\end{aligned} \tag{5}$$

식 (4) 와 식 (5) 에서 알 수 있듯이, 다음 식이 성립한다.

$$\int_a^b S^{(k)}(x)\left[f^{(k)}(x) - S^{(k)}(x)\right]dx = 0 \tag{6}$$

식 (1) 과 식 (6) 에서 알 수 있듯이, 다음 식이 성립한다.

$$\int_a^b [f^{(k)}(x)]^2 dx = \int_a^b [S^{(k)}(x)]^2 dx + \int_a^b [f^{(k)}(x) - S^{(k)}(x)]^2 dx \tag{7}$$

식 (7) 에서 알 수 있듯이, 다음 식이 성립한다.

$$\int_a^b [f^{(k)}(x)]^2 dx \geq \int_a^b [S^{(k)}(x)]^2 dx \tag{8}$$

식 (8) 에서 등호는 다음 식이 성립하는 경우에 한해서 성립한다.

$$f^{(k)} = S^{(k)} \quad a.e. \tag{9}$$

또한, 각 $j(=0,1,\cdots,k-1)$ 에 대해서 함수 $f^{(j)} - S^{(j)}$ 는 구간 $[x_1, x_n]$ 에서 연속이다. 따라서, 다음 식이 성립한다.

$$f^{(k)}(x) - S^{(k)}(x) \equiv 0 \tag{10}$$

즉, $f(x) - S(x)$ 는 차수가 $k-1$ 이하인 멱함수이다. 차수가 $k-1$ 이하인 멱함수 $f(x) - S(x)$ 가 서로 다른 근들 $x_1, x_2, \cdots, x_n$ 을 갖고 식 $k \leq n$ 이므로 항등식정리에 의해서 다음 식이 성립한다.

$$f(x) \equiv S(x) \tag{11}$$

즉, 식 (11) 이 성립하는 경우에 한해서 식 (8) 의 등호가 성립한다. ■

정리 4.2.2를 자연스플라인함수의 노름최소화성 (minimum norm property) 이라 한다. 정리 4.2.2에 의하면, $k=2$, 즉 3차 자연스플라인함수인 경우에는 다음 식이 성립한다.

$$\int_a^b [f''(x)]^2 dx \geq \int_a^b [S''(x)]^2 dx \tag{4.2.7}$$

함수 $f(x)$의 곡률(curvature) $\kappa(x)$는 다음 식들을 만족한다.

$$\kappa(x) = \frac{|f''(x)|}{\{1 + [f'(x)]^2\}^{3/2}} \approx |f''(x)| \tag{4.2.8}$$

따라서, $\int_a^b [f''(x)]^2 dx$는 구간 $[a, b]$에서 총곡률(total curvature)의 근사값이다. 식 (4.2.7)에서 알 수 있듯이, 평활한 내삽함수의 총곡률은 적어도 자연스플라인함수의 총곡률보다 작지 않다. 이러한 성질을 자연스플라인함수의 곡률최소화성(minimum curvature property)이라 한다.

4.2.4 카디날스플라인

비음인 정수 n에 대해서 다음 두 조건들을 만족하는 함수 $S(\cdot)$의 집합을 $\mathcal{S}_n$으로 표기하자. 첫째, $S(\cdot)$는 $C^{n-1}(\mathbb{R})$급 함수이다. 둘째, 각 정수 m에 대해 소구간 $(m, m+1)$에서 $S(\cdot)$는 n차 이하 멱함수이다. Schoenberg [54, p. 1]에 의하면, 집합 $\mathcal{S}_n$에 속하는 함수를 차수가 n인 카디날스플라인함수(cardinal spline function)이라 부른다. 다음 명제가 성립함은 자명하다.

$$S(\cdot) \in \mathcal{S}_n \quad \Leftrightarrow \quad S^k(\cdot) \in \mathcal{S}_{n-k},\ (1 \le k \le n) \tag{4.2.9}$$

이를 좀 더 일반화해서, 마디점들이 $x_1, x_2, \cdots, x_n$인 카디날스플라인을 정의할 수 있다. 다음 식들을 만족하는 스플라인함수들 $C_1(\cdot), C_2(\cdot), \cdots, C_n(\cdot)$을 카디날스플라인들이라고 한다.

$$C_i(x_j) = \delta_{i,j},\ \ (i, j = 1, 2, \cdots, n) \tag{4.2.10}$$

여기서 $\delta_{i,j}$는 Kronecker함수이다. 마디점이 n개인 m차 스플라인 $C_i(\cdot)$는 $m + n + 1$개 자유모수들(free parameters)를 갖는다. 식 (4.2.10)에 의해서 제약식이 n개 구성되므로, 적절한 $[m + 1]$개 제약식들이 추가되어야 카디날스플라인들을 완전히 정할 수 있다. 만약 m이 홀수이면, 즉 $m = 2k - 1$이면, 양끝 마디점들에 다음과 같은 조건을 부과하기도 한다.

$$C_i^{(r)}(x_1) = 0 = C_i^{(r)}(x_n),\ \ (r = k, k+1, \cdots, 2k-1) \tag{4.2.11}$$

식 (4.2.11)는 $2k = m + 1$개 제약조건을 부과한다.

제m차 스플라인함수 $S(\cdot)$를 다음과 같이 멱함수와 카디날스플라인함수들의 선형결합으로

나타낼 수 있음은 자명하다.

$$S(x) = P_m(x) + \sum_{i=1}^{n} a_i C_i(x) \tag{4.2.12}$$

여기서 $P_m(\cdot)$는 m차 멱함수이며, $a_1, a_2, \cdots, a_n$은 상수들이다. 식 (4.2.10)에서 알 수 있듯이, 관찰점들이 $\{(x_i, y_i) \,|\, i = 1, 2, \cdots, n\}$을 지나는 보간스플라인은 다음과 같다.

$$S(x) = P_m(x) + \sum_{i=1}^{n} y_i C_i(x) \tag{4.2.13}$$

식 (4.2.13)는 Lagrange보간식에 대응하는 식이다. 식 (4.2.11)이 성립하는 경우, 만약 $C_1(\cdot), C_2(\cdot), \cdots, C_n(\cdot)$가 자연스플라인함수들이면, $S(\cdot)$도 자연스플라인이다.

4.2.5 B-스플라인

다음과 같은 구간 $(-\infty, \infty)$의 분할 Π_∞를 살펴보자.

$$\Pi_\infty \doteq \{\cdots < x_{-n} < \cdots < x_{-1} < x_0 < x_1 < \cdots < x_n < \cdots\} \tag{4.2.14}$$

여기서 x_i는 마디점이다. 즉, 마디점들의 개수가 ∞이며, 식 $\lim_{i\to\infty} x_i = \infty$와 식 $\lim_{i\to-\infty} x_i = -\infty$가 성립한다고 가정하자. De Boor [16, p. 108]의 정의에 의하면, 마디점들을 연결하는 $[k-1]$차 B-스플라인(B-spline)들은 다음과 같다.

$$B_{i,1}(x) \doteq [x_{i+k} - x_i] \sum_{l=i}^{i+k} \left\{ \prod_{\substack{j=i \\ j\neq l}}^{i+k} \frac{1}{x_j - x_l} \, [x - x_l]_+^{k-1} \right\} \tag{4.2.15}$$

다음 식들이 성립함은 자명하다.

$$B_{i,1}(x) = 1_{[x_i, x_{i+1})}(x) = \begin{cases} 1, & (x_i \le x < x_{i+1}) \\ 0, & (\text{otherwise}) \end{cases} \tag{4.2.16}$$

$$B_{i,2}(x) = \frac{x - x_i}{x_{i+1} - x_i} 1_{[x_i, x_{i+1})}(x) + \frac{x_{i+2} - x}{x_{i+2} - x_{i+1}} 1_{[x_{i+1}, x_{i+2})}(x) \tag{4.2.17}$$

즉, 다음 식이 성립한다.

$$B_{i,2}(x) \doteq \frac{x - x_i}{x_{i+1} - x_i} B_{i,1}(x) + \frac{x_{i+2} - x}{x_{i+2} - x_{i+1}} B_{i+1,1}(x), \quad (i \in Z) \tag{4.2.18}$$

함수 $B_{i,1}(x)$를 1계(order 1) 또는 0차(degree 0) B-스플라인이라 부르고, 함수 $B_{i,2}(x)$를 2계(order 2) 또는 1차(degree 1) B-스플라인이라 부른다. 식 (4.2.18)에서 짐작할 수 있듯이, 각 $k(= 2, 3, \cdots)$에 대해서 다음 점화식이 성립한다.

$$B_{i,k+1}(x) \doteq \frac{x - x_i}{x_{i+k} - x_i} B_{i,k}(x) + \frac{x_{i+k+1} - x}{x_{i+k+1} - x_{i+1}} B_{i+1,k}(x), \quad (i \in \mathbb{Z}) \tag{4.2.19}$$

함수 $B_{i,k+1}(x)$를 $[k+1]$계(order k+1) 또는 k차(degree k) B-스플라인이라 부른다. 식 (4.2.19)를 사용해서 $[k-1]$차 스플라인들 $\{B_{j,k}(x) \,|\, j \in \mathbb{Z}\}$를 사용해서 k차 스플라인 $B_{i,k+1}(x)$를 구할 수 있다. 지지대(support set)가 $[x_0, x_n]$인 B-스플라인은 차수가 n 미만인 구분멱함수(piecewise polynomial)이다. 어떤 주어진 마디점들의 집합에 대해서 B-스플라인은 일의적이다. 따라서, B-스플라인의 B는 기저(basis)를 의미함을 짐작할 수 있다.

일반적인 스플라인과는 달리 B-스플라인은 도함수가 존재하지 않거나 존재하더라도 연속이 아닌 경우가 있다. 예를 들어, 마디점들이 $\{0, 1, 2, 3\}$인 차수가 2인 B-스플라인들은 다음과 같다.

$$B_{0,2} = \frac{1}{2}x^2, \quad (0 \le x < 1) \tag{4.2.20}$$

$$B_{1,2} = \frac{1}{2}\left[-2x^2 + 6x - 3\right], \quad (1 \le x < 2) \tag{4.2.21}$$

$$B_{2,2} = \frac{1}{2}[3 - x]^2, \quad (2 \le x \le 3) \tag{4.2.22}$$

다음 식들이 성립한다.

$$\text{At } x = 1, \qquad B_1 = B_2 = 0.5; \qquad \frac{dB_1}{dx} = \frac{dB_2}{dx} = 1 \tag{4.2.23}$$

$$\text{At } x = 2, \qquad B_2 = B_3 = 0.5; \qquad \frac{dB_2}{dx} = \frac{dB_3}{dx} = -1 \tag{4.2.24}$$

$$\text{For any } x, \qquad \frac{d^2B_1}{dx^2} = 1, \qquad \frac{d^2B_2}{dx^2} = -2, \quad \frac{d^2B_3}{dx^2} = 1 \tag{4.2.25}$$

차수들이 $0, 1, 2, 3$이고 마디점들이 $\{-7.5, -7, -6, -5, -3, -0.5, 0, 3, 3.5, 5.5, 6, 7\}$인 B-스플라인들이 그림 4.2.2에 그려져 있다.

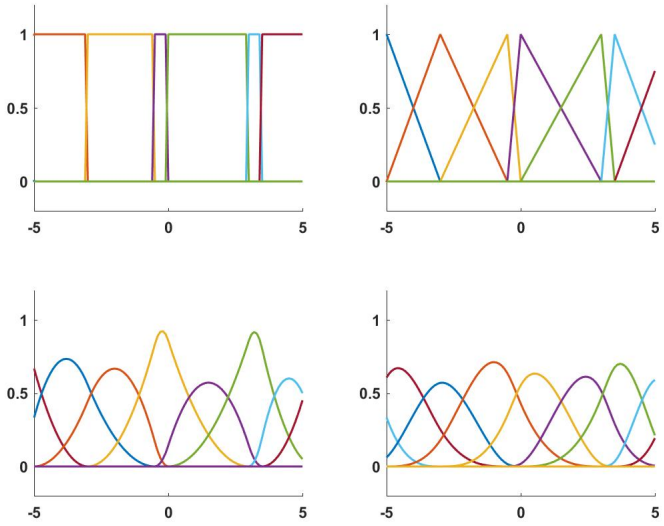


그림 4.2.2. B-스플라인함수

차수(degree)가 q인 B-스플라인의 일반적 성질들은 다음과 같다.

(a) 이 B-스플라인은 차수가 q인 멱함수들로 구성된다.

(b) 각 멱함수는 q개 내마디점들(inner knots)을 연결한다.

(c) 이 B-스플라인은 $[q+2]$개 마디점들에 의해 생성된 정의역(domain)에서 양수이고 다른 곳에서는 0이다.

(d) 양쪽 경계점들을 제외한 나머지 점들에서는 $2q$개 멱함수들이 겹쳐진다.

(e) 각 점 x에서 $[q+1]$개 B-스플라인들이 양수이다.

예제 4.2.1 B-스플라인들을 그리기 위해서 다음 MATLAB프로그램 `BSplineGraph101.m`을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: BSplineGraph101.m
% using Spline Toolbox function csapi
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
runge = @(x) 1./( 1+x.^2 );
x1 = [-10:0.5:-7 -6 -5 -3 -0.5 0 3 3.5 5.5 6 7 8 9 9.5 10]
x1Length = length(x1)
y1 = runge(x1);     % y(i-1)
xx = [-10:0.1:10];
xxLength = length(xx);
% Order 1 / Degree 0
```

```
B11 = xx - xx;  B11 = (xx >= x1(1))&(xx < x1(2) );
B12 = xx - xx;  B12  = (xx >= x1(2))&(xx < x1(3) );
B13 = xx - xx;  B13  = (xx >= x1(3))&(xx < x1(4) );
B14 = xx - xx;  B14  = (xx >= x1(4))&(xx < x1(5) );
B15 = xx - xx;  B15  = (xx >= x1(5))&(xx < x1(6) );
B16 = xx - xx;  B16  = (xx >= x1(6))&(xx < x1(7) );
B17 = xx - xx;  B17  = (xx >= x1(7))&(xx < x1(8) );
B18 = xx - xx;  B18  = (xx >= x1(8))&(xx < x1(9) );
B19 = xx - xx;  B19  = (xx >= x1(9))&(xx < x1(10) );
B110 = xx - xx;  B110  = (xx >= x1(10))&(xx < x1(11) );
B111 = xx - xx;  B111  = (xx >= x1(11))&(xx < x1(12) );
B112 = xx - xx;  B112  = (xx >= x1(12))&(xx < x1(13) );
B113 = xx - xx;  B113  = (xx >= x1(13))&(xx < x1(14) );
B114 = xx - xx;  B114  = (xx >= x1(14))&(xx < x1(15) );
B115 = xx - xx;  B115  = (xx >= x1(15))&(xx < x1(16) );
B116 = xx - xx;  B116  = (xx >= x1(16))&(xx < x1(17) );
B117 = xx - xx;  B117  = (xx >= x1(17))&(xx < x1(18) );
B118 = xx - xx;  B118  = (xx >= x1(18))&(xx < x1(19) );
B119 = xx - xx;  B119  = (xx >= x1(19))&(xx < x1(20) );
B120 = xx - xx;  B120  = (xx >= x1(20))&(xx < x1(21) );
subplot(2,2,1)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-5,5])
hold all
plot(xx,B11,xx,B12,xx,B13,xx,B14,xx,B15,xx,B16,xx,B17,xx,B18, ...
     xx,B19,xx,B110,xx,B111,xx,B112,xx,B113,xx,B114,xx,B115,xx,B116, ...
     xx,B117,xx,B118,xx,B119,'LineWidth',1.5)
axis([-5,5, -0.2, 1.2 ])
hold off
% Order 2
B21 = (xx-x1(1))/(x1(2)-x1(1)).*B11+(x1(3)-xx)/(x1(3)-x1(2)).*B12;
B22 = (xx-x1(2))/(x1(3)-x1(2)).*B12+(x1(4)-xx)/(x1(4)-x1(3)).*B13;
B23 = (xx-x1(3))/(x1(4)-x1(3)).*B13+(x1(5)-xx)/(x1(5)-x1(4)).*B14;
B24 = (xx-x1(4))/(x1(5)-x1(4)).*B14+(x1(6)-xx)/(x1(6)-x1(5)).*B15;
B25 = (xx-x1(5))/(x1(6)-x1(5)).*B15+(x1(7)-xx)/(x1(7)-x1(6)).*B16;
B26 = (xx-x1(6))/(x1(7)-x1(6)).*B16+(x1(8)-xx)/(x1(8)-x1(7)).*B17;
B27 = (xx-x1(7))/(x1(8)-x1(7)).*B17+(x1(9)-xx)/(x1(9)-x1(8)).*B18;
B28 = (xx-x1(8))/(x1(9)-x1(8)).*B18+(x1(10)-xx)/(x1(10)-x1(9)).*B19;
B29 =   (xx-x1(9))/(x1(10)-x1(9)).*B19  +(x1(11)-xx)/(x1(11)-x1(10)).*B110;
B210 = (xx-x1(10))/(x1(11)-x1(10)).*B110+(x1(12)-xx)/(x1(12)-x1(11)).*B111;
B211 = (xx-x1(11))/(x1(12)-x1(11)).*B111+(x1(13)-xx)/(x1(13)-x1(12)).*B112;
B212 = (xx-x1(12))/(x1(13)-x1(12)).*B112+(x1(14)-xx)/(x1(14)-x1(13)).*B113;
B213 = (xx-x1(13))/(x1(14)-x1(13)).*B113+(x1(15)-xx)/(x1(15)-x1(14)).*B114;
B214 = (xx-x1(14))/(x1(15)-x1(14)).*B114+(x1(16)-xx)/(x1(16)-x1(15)).*B115;
B215 = (xx-x1(15))/(x1(16)-x1(15)).*B115+(x1(17)-xx)/(x1(17)-x1(16)).*B116;
B216 = (xx-x1(16))/(x1(17)-x1(16)).*B116+(x1(18)-xx)/(x1(18)-x1(17)).*B117;
B217 = (xx-x1(17))/(x1(18)-x1(17)).*B117+(x1(19)-xx)/(x1(19)-x1(18)).*B118;
B218 = (xx-x1(18))/(x1(19)-x1(18)).*B118+(x1(20)-xx)/(x1(20)-x1(19)).*B119;
B219 = (xx-x1(19))/(x1(20)-x1(19)).*B119+(x1(21)-xx)/(x1(21)-x1(20)).*B120;
subplot(2,2,2)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-5,5])
hold all
plot(xx,B21,xx,B22,xx,B23,xx,B24,xx,B25,xx,B26,xx,B27,xx,B28, ...
     xx,B29,xx,B210,xx,B211,xx,B212,xx,B213,xx,B214,xx,B215,xx,B216, ...
     xx,B217,xx,B218,xx,B219,'LineWidth',1.5)
axis([-5,5, -0.2, 1.2 ])
hold off
% Order 3
B31 = (xx-x1(1))/(x1(3)-x1(1)).*B21    +(x1(4)-xx)/(x1(4)-x1(2)).*B22;
B32 = (xx-x1(2))/(x1(4)-x1(2)).*B22    +(x1(5)-xx)/(x1(5)-x1(3)).*B23;
B33 = (xx-x1(3))/(x1(5)-x1(3)).*B23    +(x1(6)-xx)/(x1(6)-x1(4)).*B24;
B34 = (xx-x1(4))/(x1(6)-x1(4)).*B24    +(x1(7)-xx)/(x1(7)-x1(5)).*B25;
```

```
B35 = (xx-x1(5))/(x1(7)-x1(5)).*B25    +(x1(8)-xx)/(x1(8)-x1(6)).*B26;
B36 = (xx-x1(6))/(x1(8)-x1(6)).*B26    +(x1(9)-xx)/(x1(9)-x1(7)).*B27;
B37 = (xx-x1(7))/(x1(9)-x1(7)).*B27    +(x1(10)-xx)/(x1(10)-x1(8)).*B28;
B38 = (xx-x1(8))/(x1(10)-x1(8)).*B28    +(x1(11)-xx)/(x1(11)-x1(9)).*B29;
B39 =   (xx-x1(9))/(x1(11)-x1(9)).*B29  +(x1(12)-xx)/(x1(12)-x1(10)).*B210;
B310 = (xx-x1(10))/(x1(12)-x1(10)).*B210+(x1(13)-xx)/(x1(13)-x1(11)).*B211;
B311 = (xx-x1(11))/(x1(13)-x1(11)).*B211+(x1(14)-xx)/(x1(14)-x1(12)).*B212;
B312 = (xx-x1(12))/(x1(14)-x1(12)).*B212+(x1(15)-xx)/(x1(15)-x1(13)).*B213;
B313 = (xx-x1(13))/(x1(15)-x1(13)).*B213+(x1(16)-xx)/(x1(16)-x1(14)).*B214;
B314 = (xx-x1(14))/(x1(16)-x1(14)).*B214+(x1(17)-xx)/(x1(17)-x1(15)).*B215;
B315 = (xx-x1(15))/(x1(17)-x1(15)).*B215+(x1(18)-xx)/(x1(18)-x1(16)).*B216;
B316 = (xx-x1(16))/(x1(18)-x1(16)).*B216+(x1(19)-xx)/(x1(19)-x1(17)).*B217;
B317 = (xx-x1(17))/(x1(19)-x1(17)).*B217+(x1(20)-xx)/(x1(20)-x1(18)).*B218;
B318 = (xx-x1(18))/(x1(20)-x1(18)).*B218+(x1(21)-xx)/(x1(21)-x1(19)).*B219;
subplot(2,2,3)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-5,5])
hold all
plot(xx,B31,xx,B32,xx,B33,xx,B34,xx,B35,xx,B36,xx,B37,xx,B38, ...
     xx,B39,xx,B310,xx,B311,xx,B312,xx,B313,xx,B314,xx,B315,xx,B316, ...
     xx,B317,xx,B318,'LineWidth',1.5)
axis([-5,5, -0.2, 1.2 ])
hold off
% Order 4
B41 = (xx-x1(1))/(x1(4)-x1(1)).*B31     +(x1(5)-xx)/(x1(5)-x1(2)).*B32;
B42 = (xx-x1(2))/(x1(5)-x1(2)).*B32     +(x1(6)-xx)/(x1(6)-x1(3)).*B33;
B43 = (xx-x1(3))/(x1(6)-x1(3)).*B33     +(x1(7)-xx)/(x1(7)-x1(4)).*B34;
B44 = (xx-x1(4))/(x1(7)-x1(4)).*B34     +(x1(8)-xx)/(x1(8)-x1(5)).*B35;
B45 = (xx-x1(5))/(x1(8)-x1(5)).*B35     +(x1(9)-xx)/(x1(9)-x1(6)).*B36;
B46 = (xx-x1(6))/(x1(9)-x1(6)).*B36     +(x1(10)-xx)/(x1(10)-x1(7)).*B37;
B47 = (xx-x1(7))/(x1(10)-x1(7)).*B37    +(x1(11)-xx)/(x1(11)-x1(8)).*B38;
B48 = (xx-x1(8))/(x1(11)-x1(8)).*B38    +(x1(12)-xx)/(x1(12)-x1(9)).*B39;
B49 =   (xx-x1(9))/(x1(12)-x1(9)).*B39  +(x1(13)-xx)/(x1(13)-x1(10)).*B310;
B410 = (xx-x1(10))/(x1(13)-x1(10)).*B310+(x1(14)-xx)/(x1(14)-x1(11)).*B311;
B411 = (xx-x1(11))/(x1(14)-x1(11)).*B311+(x1(15)-xx)/(x1(15)-x1(12)).*B312;
B412 = (xx-x1(12))/(x1(15)-x1(12)).*B312+(x1(16)-xx)/(x1(16)-x1(13)).*B313;
B413 = (xx-x1(13))/(x1(16)-x1(13)).*B313+(x1(17)-xx)/(x1(17)-x1(14)).*B314;
B414 = (xx-x1(14))/(x1(17)-x1(14)).*B314+(x1(18)-xx)/(x1(18)-x1(15)).*B315;
B415 = (xx-x1(15))/(x1(18)-x1(15)).*B315+(x1(19)-xx)/(x1(19)-x1(16)).*B316;
B416 = (xx-x1(16))/(x1(19)-x1(16)).*B316+(x1(20)-xx)/(x1(20)-x1(17)).*B317;
B417 = (xx-x1(17))/(x1(20)-x1(17)).*B317+(x1(21)-xx)/(x1(21)-x1(18)).*B318;
subplot(2,2,4)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-5,5])
hold all
plot(xx,B41,xx,B42,xx,B43,xx,B44,xx,B45,xx,B46,xx,B47,xx,B48, ...
     xx,B49,xx,B410,xx,B411,xx,B412,xx,B413,xx,B414,xx,B415,xx,B416, ...
     xx,B417,'LineWidth',1.5)
axis([-5,5, -0.2, 1.2 ])
hold off
saveas(gcf,'BSplineGraph101.jpg')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 `B_SplineGraph101.m`은 마디점들이 $\{0, 1, 3, 7, 13\}$ 인 B-스플라인들을 그리기 위한 것이다. 이 MATLAB프로그램을 실행하면, 그림 4.2.3을 출력한다. ■

예제 4.2.2 Python을 사용해서 예제 4.2.1을 다시 다루기 위해서, 다음 Python프로그램

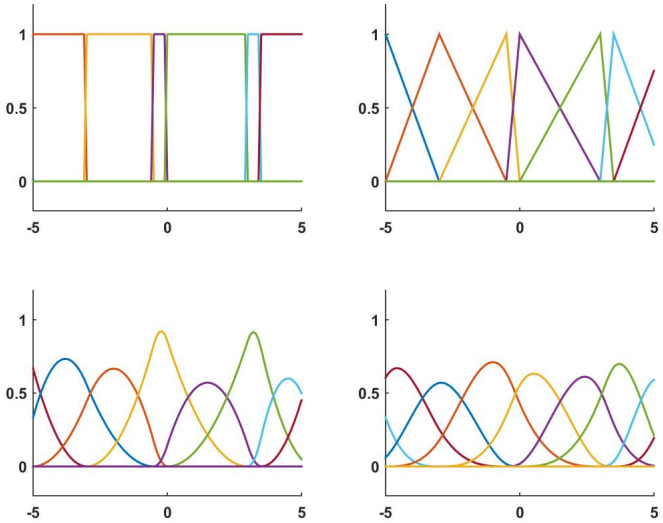


그림 4.2.3. B-스플라인함수

BSplineGraph101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""


import numpy as np
import matplotlib.pyplot as plt

runge = lambda x: 1/( 1+x**2 );
x1a = np.arange(-10,-7+0.1,0.5);
x1b =np.array([-6, -5, -3, -0.5, 0, 3, 3.5, 5.5, 6, 7, 8, 9, 9.5, 10]);
x1 = np.hstack((x1a,x1b))
x1Length = len(x1)
y1 = runge(x1);
xx = np.linspace(-10,10,201);
xxLength = len(xx);

# Order 1 / Degree 0
B11 = xx - xx;  B11  = (xx >= x1[0])&(xx < x1[1]);
B12 = xx - xx;  B12  = (xx >= x1[1])&(xx < x1[2]);
B13 = xx - xx;  B13  = (xx >= x1[2])&(xx < x1[3]);
B14 = xx - xx;  B14  = (xx >= x1[3])&(xx < x1[4]);
B15 = xx - xx;  B15  = (xx >= x1[4])&(xx < x1[5]);
B16 = xx - xx;  B16  = (xx >= x1[5])&(xx < x1[6]);
B17 = xx - xx;  B17  = (xx >= x1[6])&(xx < x1[7]);
B18 = xx - xx;  B18  = (xx >= x1[7])&(xx < x1[8]);
B19 = xx - xx;  B19  = (xx >= x1[8])&(xx < x1[9]);
B110 = xx - xx;  B110  = (xx >= x1[9]) &(xx < x1[10]);
B111 = xx - xx;  B111  = (xx >= x1[10])&(xx < x1[11]);
B112 = xx - xx;  B112  = (xx >= x1[11])&(xx < x1[12]);
B113 = xx - xx;  B113  = (xx >= x1[12])&(xx < x1[13]);
B114 = xx - xx;  B114  = (xx >= x1[13])&(xx < x1[14]);
B115 = xx - xx;  B115  = (xx >= x1[14])&(xx < x1[15]);
B116 = xx - xx;  B116  = (xx >= x1[15])&(xx < x1[16]);
```

```
B117 = xx - xx;  B117  = (xx >= x1[16])&(xx < x1[17]);
B118 = xx - xx;  B118  = (xx >= x1[17])&(xx < x1[18]);
B119 = xx - xx;  B119  = (xx >= x1[18])&(xx < x1[19]);
B120 = xx - xx;  B120  = (xx >= x1[19])&(xx < x1[20]);

# Order 2
B21 = (xx-x1[0])/(x1[1]-x1[0])*B11+(x1[2]-xx)/(x1[2]-x1[1])*B12;
B22 = (xx-x1[1])/(x1[2]-x1[1])*B12+(x1[3]-xx)/(x1[3]-x1[2])*B13;
B23 = (xx-x1[2])/(x1[3]-x1[2])*B13+(x1[4]-xx)/(x1[4]-x1[3])*B14;
B24 = (xx-x1[3])/(x1[4]-x1[3])*B14+(x1[5]-xx)/(x1[5]-x1[4])*B15;
B25 = (xx-x1[4])/(x1[5]-x1[4])*B15+(x1[6]-xx)/(x1[6]-x1[5])*B16;
B26 = (xx-x1[5])/(x1[6]-x1[5])*B16+(x1[7]-xx)/(x1[7]-x1[6])*B17;
B27 = (xx-x1[6])/(x1[7]-x1[6])*B17+(x1[8]-xx)/(x1[8]-x1[7])*B18;
B28 = (xx-x1[7])/(x1[8]-x1[7])*B18+(x1[9]-xx)/(x1[9]-x1[8])*B19;
B29 =   (xx-x1[8])/(x1[9]-x1[8])*B19  +(x1[10]-xx)/(x1[10]-x1[9])*B110;
B210 = (xx-x1[9])/(x1[10]-x1[9])*B110+(x1[11]-xx)/(x1[11]-x1[10])*B111;
B211 = (xx-x1[10])/(x1[11]-x1[10])*B111+(x1[12]-xx)/(x1[12]-x1[11])*B112;
B212 = (xx-x1[11])/(x1[12]-x1[11])*B112+(x1[13]-xx)/(x1[13]-x1[12])*B113;
B213 = (xx-x1[12])/(x1[13]-x1[12])*B113+(x1[14]-xx)/(x1[14]-x1[13])*B114;
B214 = (xx-x1[13])/(x1[14]-x1[13])*B114+(x1[15]-xx)/(x1[15]-x1[14])*B115;
B215 = (xx-x1[14])/(x1[15]-x1[14])*B115+(x1[16]-xx)/(x1[16]-x1[15])*B116;
B216 = (xx-x1[15])/(x1[16]-x1[15])*B116+(x1[17]-xx)/(x1[17]-x1[16])*B117;
B217 = (xx-x1[16])/(x1[17]-x1[16])*B117+(x1[18]-xx)/(x1[18]-x1[17])*B118;
B218 = (xx-x1[17])/(x1[18]-x1[17])*B118+(x1[19]-xx)/(x1[19]-x1[18])*B119;
B219 = (xx-x1[18])/(x1[19]-x1[18])*B119+(x1[20]-xx)/(x1[20]-x1[19])*B120;

# Order 3
B31 = (xx-x1[0])/(x1[2]-x1[0])*B21    +(x1[3]-xx)/(x1[3]-x1[1])*B22;
B32 = (xx-x1[1])/(x1[3]-x1[1])*B22    +(x1[4]-xx)/(x1[4]-x1[2])*B23;
B33 = (xx-x1[2])/(x1[4]-x1[2])*B23    +(x1[5]-xx)/(x1[5]-x1[3])*B24;
B34 = (xx-x1[3])/(x1[5]-x1[3])*B24    +(x1[6]-xx)/(x1[6]-x1[4])*B25;
B35 = (xx-x1[4])/(x1[6]-x1[4])*B25    +(x1[7]-xx)/(x1[7]-x1[5])*B26;
B36 = (xx-x1[5])/(x1[7]-x1[5])*B26    +(x1[8]-xx)/(x1[8]-x1[6])*B27;
B37 = (xx-x1[6])/(x1[8]-x1[6])*B27    +(x1[9]-xx)/(x1[9]-x1[7])*B28;
B38 = (xx-x1[7])/(x1[9]-x1[7])*B28    +(x1[10]-xx)/(x1[10]-x1[8])*B29;
B39 =   (xx-x1[8])/(x1[10]-x1[8])*B29  +(x1[11]-xx)/(x1[11]-x1[9])*B210;
B310 = (xx-x1[9])/(x1[11]-x1[9])*B210+(x1[12]-xx)/(x1[12]-x1[10])*B211;
B311 = (xx-x1[10])/(x1[12]-x1[10])*B211+(x1[13]-xx)/(x1[13]-x1[11])*B212;
B312 = (xx-x1[11])/(x1[13]-x1[11])*B212+(x1[14]-xx)/(x1[14]-x1[12])*B213;
B313 = (xx-x1[12])/(x1[14]-x1[12])*B213+(x1[15]-xx)/(x1[15]-x1[13])*B214;
B314 = (xx-x1[13])/(x1[15]-x1[13])*B214+(x1[16]-xx)/(x1[16]-x1[14])*B215;
B315 = (xx-x1[14])/(x1[16]-x1[14])*B215+(x1[17]-xx)/(x1[17]-x1[15])*B216;
B316 = (xx-x1[15])/(x1[17]-x1[15])*B216+(x1[18]-xx)/(x1[18]-x1[16])*B217;
B317 = (xx-x1[16])/(x1[18]-x1[16])*B217+(x1[19]-xx)/(x1[19]-x1[17])*B218;
B318 = (xx-x1[17])/(x1[19]-x1[17])*B218+(x1[20]-xx)/(x1[20]-x1[18])*B219;

# Order 4
B41 = (xx-x1[0])/(x1[3]-x1[0])*B31     +(x1[4]-xx)/(x1[4]-x1[1])*B32;
B42 = (xx-x1[1])/(x1[4]-x1[1])*B32     +(x1[5]-xx)/(x1[5]-x1[2])*B33;
B43 = (xx-x1[2])/(x1[5]-x1[2])*B33     +(x1[6]-xx)/(x1[6]-x1[3])*B34;
B44 = (xx-x1[3])/(x1[6]-x1[3])*B34     +(x1[7]-xx)/(x1[7]-x1[4])*B35;
B45 = (xx-x1[4])/(x1[7]-x1[4])*B35     +(x1[8]-xx)/(x1[8]-x1[5])*B36;
B46 = (xx-x1[5])/(x1[8]-x1[5])*B36     +(x1[9]-xx)/(x1[9]-x1[6])*B37;
B47 = (xx-x1[6])/(x1[9]-x1[6])*B37    +(x1[10]-xx)/(x1[10]-x1[7])*B38;
B48 = (xx-x1[7])/(x1[10]-x1[7])*B38    +(x1[11]-xx)/(x1[11]-x1[8])*B39;
B49 =   (xx-x1[8])/(x1[11]-x1[8])*B39  +(x1[12]-xx)/(x1[12]-x1[9])*B310;
B410 = (xx-x1[9])/(x1[12]-x1[9])*B310+(x1[13]-xx)/(x1[13]-x1[10])*B311;
B411 = (xx-x1[10])/(x1[13]-x1[10])*B311+(x1[14]-xx)/(x1[14]-x1[11])*B312;
B412 = (xx-x1[11])/(x1[14]-x1[11])*B312+(x1[15]-xx)/(x1[15]-x1[12])*B313;
B413 = (xx-x1[12])/(x1[15]-x1[12])*B313+(x1[16]-xx)/(x1[16]-x1[13])*B314;
B414 = (xx-x1[13])/(x1[16]-x1[13])*B314+(x1[17]-xx)/(x1[17]-x1[14])*B315;
```

```
B415 = (xx-x1[14])/(x1[17]-x1[14])*B315+(x1[18]-xx)/(x1[18]-x1[15])*B316;
B416 = (xx-x1[15])/(x1[18]-x1[15])*B316+(x1[19]-xx)/(x1[19]-x1[16])*B317;
B417 = (xx-x1[16])/(x1[19]-x1[16])*B317+(x1[20]-xx)/(x1[20]-x1[17])*B318;

# Plotting
fig = plt.figure()
# row and column sharing
f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex='col', sharey='row')
ax1.plot(xx,B11,xx,B12,xx,B13,xx,B14,xx,B15,xx,B16,xx,B17,xx,B18,
     xx,B19,xx,B110,xx,B111,xx,B112,xx,B113,xx,B114,xx,B115,xx,B116,
     xx,B117,xx,B118,xx,B119,lw=1.5)
ax1.axis([-5, 5, -0.2, 1.2 ])
ax2.plot(xx,B21,xx,B22,xx,B23,xx,B24,xx,B25,xx,B26,xx,B27,xx,B28,
     xx,B29,xx,B210,xx,B211,xx,B212,xx,B213,xx,B214,xx,B215,xx,B216,
     xx,B217,xx,B218,xx,B219,lw=1.5)
ax2.axis([-5, 5, -0.2, 1.2 ])
ax3.plot(xx,B31,xx,B32,xx,B33,xx,B34,xx,B35,xx,B36,xx,B37,xx,B38,
     xx,B39,xx,B310,xx,B311,xx,B312,xx,B313,xx,B314,xx,B315,xx,B316,
     xx,B317,xx,B318,lw=1.5)
ax3.axis([-5, 5, -0.2, 1.2 ])
ax4.plot(xx,B41,xx,B42,xx,B43,xx,B44,xx,B45,xx,B46,xx,B47,xx,B48,
     xx,B49,xx,B410,xx,B411,xx,B412,xx,B413,xx,B414,xx,B415,xx,B416,
     xx,B417,lw=1.5)
ax4.axis([-5, 5, -0.2, 1.2 ])
plt.show()
fig.savefig('BSplineGraph101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 4.2.1의 결과와 같다. ■

4.2.6 직교스플라인

스플라인함수들이 직교성을 갖으면, 직교스플라인함수들(orthogonal splines)이라 한다. 이 소절에서는 Mason et al. [40]에서 다룬 B-스플라인함수들을 바탕으로 한 직교스플라인함수들을 살펴보자.

첫째, 1차 B-스플라인의 직교화에 대해서 살펴보자. 마디점들 $x_1, x_2, \cdots, x_n$과 외부마디점들(exterior knots) x_0, x_{n+1}이 다음 식을 만족한다고 하자.

$$x_0 < x_1 = a < x_2 < \cdots < x_{n-1} < x_n = b < x_{n+1} \tag{4.2.26}$$

이들을 마디점들로 갖는 1차 B-스플라인함수들을 $L_1, L_2, \cdots, L_n$이라 하면, L_k는 지지대 $[x_{k-1}, x_{k+1}]$에서 연속이다. 편의상 다음 식들이 성립한다고 가정하자.

$$L_k(x_k) = 1, \quad (k = 1, 2, \cdots, n) \tag{4.2.27}$$

이 B-스플라인함수들로부터 다음과 같은 직교스플라인들 $\{P_k \,|\, k = 1, 2, \cdots, n\}$을 유도하기로 하자.

$$P_1 = L_1 \tag{4.2.28}$$

$$P_k = L_k - a_{k-1}P_{k-1}, \quad (k = 2, 3, \cdots, n) \tag{4.2.29}$$

여기서 P_k는 지지대가 $[a, x_{k+1}]$인 1차스플라인이고, a_{k-1}은 미지 상수이다. 각 $r(\leq k-2)$에 대해서 P_r의 지지대는 $[a, x_{r+1}]$이고 또한 L_k의 지지대는 $[x_{k-1}, x_{k+1}]$이다. 따라서, 다음 식들이 성립한다.

$$\langle P_r, L_k \rangle \doteq \int_a^b P_r(x)L_k(x)dx = 0, \quad (r \leq k-2) \tag{4.2.30}$$

식 (4.2.29)와 식 (4.2.30)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\langle P_r, P_k \rangle = 0, \quad (r \leq k-2) \tag{4.2.31}$$

따라서, 직교스플라인들 $\{P_k\}$를 구하기 위해서는 다음 식을 만족하는 a_{k-1}을 구하면 된다.

$$\langle P_{k-1}, P_k \rangle = 0 \tag{4.2.32}$$

식 (4.2.32)를 다음과 같이 쓸 수 있다.

$$\langle P_{k-1}, L_k \rangle - a_{k-1}\|P_{k-1}\|^2 = 0 \tag{4.2.33}$$

여기서 $\|f\| \doteq \sqrt{< f, f >}$이다. 즉, 다음 식이 성립한다.

$$a_k = \frac{\langle P_k, L_{k+1} \rangle}{\|P_k\|^2} \tag{4.2.34}$$

식 (4.2.29)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\begin{aligned} \|P_k\|^2 &= \langle L_k, L_k \rangle - 2a_{k-1}\langle L_k, P_{k-1} \rangle + a_{k-1}a_{k-1}\|P_{k-1}\|^2 \\ &= \|L_k\|^2 - a_{k-1}\langle L_k, P_{k-1} \rangle \end{aligned} \tag{4.2.35}$$

여기서 두 번째 등호는 식 (4.2.33)에 의해서 성립한다. 식 (4.2.29)와 식 (4.2.31)에서 알 수 있듯이, 다음 식이 성립한다.

$$\langle P_{k-1}, L_k \rangle = \langle L_{k-1}, L_k \rangle \tag{4.2.36}$$

식 (4.2.35)와 식 (4.2.36)에서 알 수 있듯이, 다음 식이 성립한다.

$$\|P_k\|^2 = \|L_k\|^2 - a_{k-1} \langle L_{k-1}, L_k \rangle \tag{4.2.37}$$

식 (4.2.36)과 식 (4.2.37)를 식 (4.2.34)에 대입하면, 다음 식을 얻는다.

$$a_k = \frac{\langle L_k, L_{k+1} \rangle}{\|L_k\|^2 - a_{k-1} \langle L_{k-1}, L_k \rangle}, \quad (k = 2, 3, \cdots, n) \tag{4.2.38}$$

다음 식들이 성립한다.

$$\langle L_2 - P_2, P_1 \rangle = \langle L_2, P_1 \rangle = \langle L_2, L_1 \rangle \tag{4.2.39}$$

여기서 첫 번째 등호는 $\{P_k\}$의 직교성에 의해서, 그리고 두 번째 등호는 식 (4.2.36)에 의해서 성립한다. 식 (4.2.28)과 식 (4.2.36)를 식 (4.2.34)에 대입하면, 초기값이 다음과 같음을 알 수 있다.

$$a_1 = \frac{\langle L_1, L_2 \rangle}{\|L_1\|^2} \tag{4.2.40}$$

식 (4.2.17)에서 알 수 있듯이, 다음 식이 성립한다.

$$L_k(x) = \frac{x - x_{k-1}}{x_k - x_{k-1}} 1_{[x_{k-1}, x_k)}(x) + \frac{x_{k+1} - x}{x_{k+1} - x_k} 1_{[x_k \le x < x_{k+1})}(x) \tag{4.2.41}$$

식 (4.2.41)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\|L_1\|^2 = \frac{1}{3} [x_2 - x_1] \tag{4.2.42}$$

$$\|L_n\|^2 = \frac{1}{3} [x_n - x_{n-1}] \tag{4.2.43}$$

$$\|L_k\|^2 = \frac{1}{3} [x_{k+1} - x_{k-1}], \quad (k = 2, 3, \cdots, n-1) \tag{4.2.44}$$

$$\langle L_k, L_{k+1} \rangle = \frac{1}{6} [x_{k+1} - x_k], \quad (k = 1, 2, \cdots, n-1) \tag{4.2.45}$$

식 (4.2.42) ∼ 식 (4.2.45)를 식 (4.2.38)과 식 (4.2.40)에 대입하면, 다음 식들을 얻는다.

$$a_1 = \frac{1}{2}, \quad a_k = \frac{x_{k+1} - x_k}{2\left[x_{k+1} - x_{k-1}\right] - a_{k-1}\left[x_k - x_{k-1}\right]}, \quad (k = 2, 3, \cdots, n) \tag{4.2.46}$$

만약 마디점들이 등간격이라면, 식 (4.2.46)을 다음과 같이 쓸 수 있다.

$$a_1 = \frac{1}{2} \tag{4.2.47}$$

$$a_k = \frac{1}{4 - a_{k-1}}, \quad (k = 2, 3, \cdots, n) \tag{4.2.48}$$

다음 방정식의 두 근들은 $\alpha = 2 - \sqrt{3}$과 $\beta = 2 + \sqrt{3}$이다.

$$\alpha = \frac{1}{4 - \alpha} \tag{4.2.49}$$

초기값 $a_1 = 0.5$를 점화식 (4.2.48)에 대입하면, 다음 식이 성립함을 알 수 있다.

$$\lim_{n \to \infty} a_n = \alpha = 2 - \sqrt{3} \tag{4.2.50}$$

식 (4.2.50)이 성립함을 확인하기 위해서, 다음 MATLAB 프로그램 `RecursiveFormula4OrthogonalSpline101.m`을 실행하라.

```
% ------------------------------------------------------------------
%  Filename: RecursiveFormula4OrthogonalSpline101.m
%  Recursive Formula for Orthogonal B-Spline
%  Programmed by CBS
%------------------------------------------------------------------
close all; clear all; format long
alpha = 2 - sqrt(3)
a1(1) = 1/2;
for n1=2:1:20;
    a1(n1) = 1/(4-a1(n1-1));
end
x1 = a1;
y1 = 1./(4-x1);
xa = eps:0.01:0.6;
ya = 1./(4-xa);
set(gca,'fontsize',11,'fontweigh','bold')
hold on
plot(xa,xa,'g-',xa,ya,'g-','linewidth',2)
plot([a1(1) a1(1) a1(2) a1(2) a1(3) a1(3) a1(4) a1(4)], ...
     [0     a1(2) a1(2) a1(3) a1(3) a1(4) a1(4) a1(5)], ...
     'k-','linewidth',1.5)
axis([0 0.6 0 0.6 ])
dx = 0.005; dy = 0.015;
text(a1(1)+dx,0+dy,'(a(1),0)')
text(a1(1)+dx,a1(2)-dy,'(a(1),a(1))')
text(a1(2)+dx,a1(2)+dy,'(a(2),a(2))')
```

```
text(a1(2)+dx,a1(3)-0.5*dy,'(a(2),a(3))')
text(alpha-0.04,alpha+0.02,'(\alpha,\alpha)')
text(0.1,0.245,'y = 1/(4-x)')
text(0.1,0.085,'y = x')
plot([a1(1) a1(1) a1(2) a1(2) a1(3) a1(3) a1(4) a1(4)], ...
     [0     a1(2) a1(2) a1(3) a1(3) a1(4) a1(4) a1(5)], ...
     'k*','linewidth',1)
hold off
saveas(gcf,'RecursiveFormula4OrthogonalSpline101.jpg')
save('RecursiveFormula4OrthogonalSpline101.txt','a1','-ascii')
% End of program
%-------------------------------------------------------------------
```

이 MATLAB프로그램 RecursiveFormula4OrthogonalSpline101.m을 실행하면, 그림 4.2.4을 출력한다. 이 그림에서 확인할 수 있듯이, 식 (4.2.50)이 성립한다.

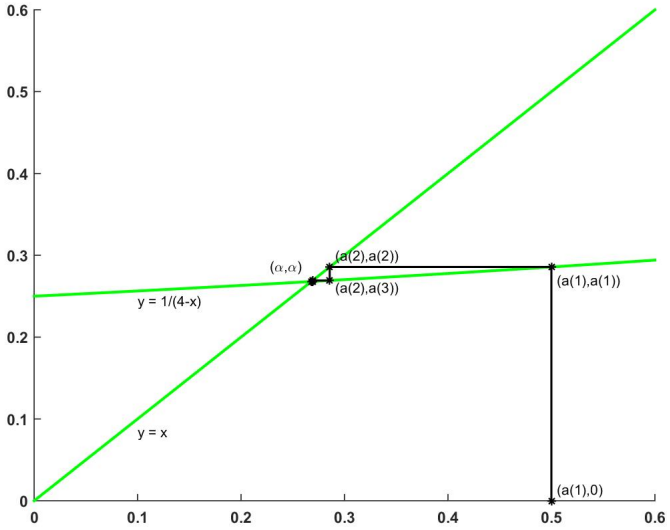


그림 4.2.4. 점화식의 수렴

예제 4.2.3 Python을 사용해서 그림 4.2.4을 다시 출력하기 위해서, 다음 Python프로그램 RecursiveFormula4OrthogonalSpline101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

alpha = 2 - np.sqrt(3)
aa = np.zeros(20);
aa[0] = 1/2;
for nn in range(1,20):
    aa[nn] = 1/(4-aa[nn-1]);
xa = np.ones(20) - aa;
```

```
15
16
17 yy = 1/(4.0-aa);
18 MachEpss = np.finfo(float).eps  # OR 7./3 - 4./3 -1
19 xa = np.arange(2*MachEpss,0.6+MachEpss,0.01);
20 ya = 1/(4-xa);
21
22 # Plotting
23 fig = plt.figure()
24 plt.plot(xa,xa,'g-',lw=2)
25 plt.hold(True); plt.plot(xa,ya,'g-',lw=2)
26 aax = np.array([ aa[0], aa[0], aa[1], aa[1], aa[2], aa[2], aa[3], aa[3] ]);
27 aay = np.array([     0, aa[1], aa[1], aa[2], aa[2], aa[3], aa[3], aa[4] ]);
28 plt.hold(True); plt.plot(aax,aay,'k-',lw=1.5)
29 plt.axis([ 0, 0.6, 0, 0.6 ]);
30 dx = 0.005; dy = 0.015;
31 plt.text(aa[0]+dx,0+dy,r'(a(1),0)')
32 plt.text(aa[0]+dx,aa[1]-dy,r'(a(1),a(1))')
33 plt.text(aa[1]+dx,aa[1]+dy,u'(a(2),a(2))')
34 plt.text(aa[1]+dx,aa[2]-1.5*dy,r'(a(2),a(3))')
35 # plt.text(alpha-0.04,alpha+0.02,'(\alpha,\alpha)')
36 plt.text(0.1,0.270,u'y = 1/(4-x)')
37 plt.text(0.1,0.085,r'y = x')
38 abx = np.array([ aa[0], aa[0], aa[1], aa[1], aa[2], aa[2], aa[3], aa[3] ]);
39 aby = np.array([     0, aa[1], aa[1], aa[2], aa[2], aa[3], aa[3], aa[4] ]);
40 plt.hold(True); plt.plot(abx,aby,'k*',lw=1.5)
41 plt.show()
42 fig.savefig('RecursiveFormula4OrthogonalSpline101Py.png')
43
44 # End of Program
```

이 Python프로그램을 수행한 결과는 그림 4.2.4와 같다. ■

둘째, 3차B-스플라인의 직교화에 대해서 살펴보자. 마디점들 $x_1, x_2, \cdots, x_n$과 외부마디점들 $x_{-2}, x_{-1}, x_0, x_{n+1}, x_{n+3}, x_{n+3}$이 다음 식을 만족한다고 하자.

$$x_{-2} < x_{-1} < x_0 < x_1 = a < x_2 < \cdots < x_{n-1} < x_n = b < x_{n+1} < x_{n+2} < x_{n+3} \quad (4.2.51)$$

이들을 마디점들로 갖는 정규화된 3차B-스플라인함수들을 $B_0, B_1, \cdots, B_n, B_{n+1}$이라 하면, B_k는 지지대 $[x_{k-2}, x_{k+2}]$에서 연속이다. 이 B-스플라인함수들로부터 다음 식들을 만족하는 직교스플라인들 $\{P_k \,|\, k = 0, 1, \cdots, n, n+1\}$을 유도하기로 하자.

$$\langle P_k, P_j \rangle = 0, \quad (k \neq j) \quad (4.2.52)$$

$$P_0 = B_0, \quad P_1 = B_1 - a_{1,1}P_0, \quad P_2 = B_2 - a_{2,1}P_1 - a_{2,2}P_0 \quad (4.2.53)$$

$$P_k = B_k - a_{k,1}P_{k-1} - a_{k,2}P_{k-2} - a_{k,3}P_{k-3}, \quad (k = 3, 4, \cdots, n+1) \quad (4.2.54)$$

여기서 P_k는 지지대가 $[a, x_{k+2}]$인 3차스플라인이다. 편의상, 다음 식들이 성립한다고 가정하자.

$$a_{0,1} = a_{0,2} = a_{0,3} = a_{1,2} = a_{1,3} = a_{2,3} = 0 \tag{4.2.55}$$

이러한 가정 하에서는 식 (4.2.53)과 식 (4.2.54)를 점화식 (4.2.54) 하나로 나타낼 수 있다. 다음 상수들을 정의하자.

$$n_k \doteq \|P_k\|^2 = \langle P_k, P_k \rangle, \quad (k = 0, 1, \cdots, n, n+1) \tag{4.2.56}$$

$$b_{k,l} \doteq \langle B_k, B_{k-l} \rangle, \quad (l = 0, 1, 2, 3) \tag{4.2.57}$$

편의상, 다음 식들이 성립한다고 가정하자.

$$b_{0,1} = b_{0,2} = b_{0,3} = b_{1,2} = b_{1,3} = b_{2,3} = 0 \tag{4.2.58}$$

식 (4.2.52)와 식 (4.2.54)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\langle B_k - a_{k,1}P_{k-1} - a_{k,2}P_{k-2} - a_{k,3}P_{k-3}, P_{k-r} \rangle = \langle P_k, P_{k-r} \rangle = 0, \quad (r = 1, 2, 3) \tag{4.2.59}$$

즉, 다음 식들이 성립한다.

$$\langle B_k, P_{k-r} \rangle = a_{k,r} n_{k-r}, \quad (r = 1, 2, 3) \tag{4.2.60}$$

식 (4.2.54)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\langle B_{k-r} - a_{k-r,1}P_{k-r-1} - a_{k-r,2}P_{k-r-2} - a_{k-r,3}P_{k-3}, B_k \rangle = \langle P_{k-r}, B_k \rangle, \quad (r = 1, 2, 3) \tag{4.2.61}$$

따라서, 다음 식들이 성립한다.

$$\langle B_{k-3}, B_k \rangle = \langle P_{k-3}, B_k \rangle \tag{4.2.62}$$

$$\langle B_{k-2}, B_k \rangle - a_{k-2,1} \langle P_{k-3}, B_k \rangle = \langle P_{k-2}, B_k \rangle \tag{4.2.63}$$

$$\langle B_{k-1}, B_k \rangle - a_{k-1,1} \langle P_{k-2}, B_k \rangle - a_{k-1,2} \langle P_{k-3}, B_k \rangle = \langle P_{k-1}, B_k \rangle \tag{4.2.64}$$

식 (4.2.62) ∼ 식 (4.2.64) 그리고 식 (4.2.60)에서 알 수 있듯이, 다음 식들이 성립한다.

$$a_{k,3} = \frac{b_{k,3}}{n_{k-3}} \tag{4.2.65}$$

$$a_{k,2} = \frac{b_{k,2} - a_{k-2,1}b_{k,3}}{n_{k-2}} \tag{4.2.66}$$

$$a_{k,1} = \frac{b_{k,1} - a_{k-1,1}a_{k,2}n_{k-2} - a_{k-1,2}b_{k,3}}{n_{k-1}} \tag{4.2.67}$$

식 (4.2.54)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\begin{aligned} n_k &= \langle P_k, P_k \rangle \\ &= \langle B_k - a_{k,1}P_{k-1} - a_{k,2}P_{k-2} - a_{k,3}P_{k-3}, B_k - a_{k,1}P_{k-1} - a_{k,2}P_{k-2} - a_{k,3}P_{k-3} \rangle \\ &= b_{k,0} + \sum_{r=1}^{3} \left[-2a_{k,r} \langle B_k, P_{k-r} \rangle + a_{k,r}^2 n_{k-r} \right] \end{aligned} \tag{4.2.68}$$

즉, 다음 식이 성립한다.

$$n_k = b_{k,0} - \sum_{r=1}^{3} a_{k,r}^2 n_{k-r} \tag{4.2.69}$$

우리의 목표는 각 k에 대해서 계수들 $a_{k,1}, a_{k,2}, a_{k,3}$을 구하는 것이다. 따라서, 먼저 주어진 B-스플라인을 사용해서 $b_{k,0}, b_{k,1}, b_{k,2}, b_{k,3}$를 구한다. 다음으로 식 (4.2.65) ∼ 식 (4.2.67)을 사용해서 $a_{k,1}, a_{k,2}, a_{k,3}$를 구한다. 이 값들을 식 (4.2.69)에 대입해서 n_k를 구해서, 다음 단계 계산에 사용한다.

4.2.7 스플라인의 오차

다음 명제는 스플라인함수의 오차에 관한 것이다.

명제 4.2.1

만약 함수 f가 $C^4[a,b]$급이고, 또한 만약 $h \geq \max_i \{x_i - x_{i-1}\}$이면, 구분적 Hermite 보간함수 $\hat{f}_2$는 다음 식들을 만족한다.

$$\begin{aligned} \|f - \hat{f}_2\|_\infty &\leq \frac{5}{384}\|f^{(4)}\|_\infty h^4 \\ \|f' - \hat{f}_2'\|_\infty &\leq \left[\frac{\sqrt{3}}{216} + \frac{1}{24}\right] \|f^{(4)}\|_\infty h^3 \end{aligned}$$

명제 4.2.1의 증명은 Prenter [46, p.112]를 참조하라. 명제 4.2.1에서 알 수 있듯이, 2차스플라인함수는 빠른 속도로 수렴함을 알 수 있다. 일반적으로, $C^{k+1}[a.b]$급 함수 f의 k차 스플라인함수의 수렴속도는 $O(n^{-k-1})$이다.

근사기법으로서 스플라인함수가 아주 유용한 이유는 다음과 같다. 첫째, 일반적으로 사용하는 스플라인은 구분적으로 3차함수이기 때문에 많은 계산량을 필요로 하지 않는다. 둘째, 근사시키고자 하는 함수 f가 높은 차수의 도함수를 가져야 할 필요가 없다. 명제 4.2.1에서 알 수 있듯이, 함수 f의 5차 도함수 $f^{(5)}$가 존재 여부는 3차스플라인함수의 오차에 영향을 미치지 않는다.

제4.3절 1차스플라인과 2차스플라인

4.3.1 1차스플라인

구간 $[x_0, x_n]$에서 가장 간단한 스플라인함수 $S_1(x)$는 다음과 같다.

$$S_1(x) = \sum_{i=0}^{n-1} \left(f_i + \frac{f_{i+1} - f_i}{x_{i+1} - x_i}[x - x_i] \right) 1_{[x_i, x_{i+1})}(x) \tag{4.3.1}$$

이 $S_1(x)$를 선형스플라인함수(linear spline function)라 부른다. 이 선형스플라인함수는 앞에서 구한 1차 선형보간함수와 동일하다. 각 마디점 $x_i, (i = 1, 2, \cdots, n-1)$에서 이 $S_1(x)$는 연속이지만 일반적으로 미분불가능하다. 마찬가지로 n차 스플라인함수를 다음과 같이 쓸 수 있다.

$$S_n(x) = \sum_i \alpha_i B_{i,n}(x) \tag{4.3.2}$$

예제 4.3.1 다음과 같은 Runge함수 f를 살펴보자.

$$f(x) = \frac{1}{1 + x^2} \tag{1}$$

구간 $[-5, 5]$에서 이 Runge함수 f의 선형스플라인함수를 그리기 위해서, 다음 MATLAB 프로그램 LinearSpline101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: LinearSpline101.m
% Linear Spline with n = 10 using m-file interp1
% Programmed by CBS
% ------------------------------------------------------------------------
function LinearSpline
```

```
clear all, close all
runge = inline('1./( 1+x.^2 )','x');
x1 = [-5:1:5]' ;     % x(i-1)
y1 = runge(x1);      % y(i-1)
xx = -5:0.02:5;
yy = interp1(x1,y1,xx,'linear');
ytrue = runge(xx);
err = max(abs(yy-ytrue))
% Plotting
plot(xx,ytrue,'k-', xx,yy,'r--',x1,y1,'ko','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xtick',-5:1:5)
legend('original function','linear spline','knots')
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
axis( [ -5 5 0 1.2 ])
saveas(gcf,'LinearSpline101.jpg')
save('LinearSpline101.txt','xx','yy','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 LinearSpline101.m에서는 MATLAB함수 interp1에 모수로 'linear'를 지정하였다. 이는 선형스플라인함수를 사용하기 위한 것이다.

이 MATLAB프로그램을 실행하면 알 수 있듯이, 구간 $[-5,5]$에서 함수 $f(x)$와 구분적 선형스플라인함수 사이의 최대오차 $\max_x |f(x) - S(x)|$는 0.0674이다. 또한, 이 MATLAB 프로그램을 실행하면, 그림 4.3.1을 출력한다. 그림 4.3.1에서 Runge함수 $f(x)$는 흑색 실선으로, 그리고 선형스플라인함수는 적색 긴점선으로 그려져 있다. 그림 4.3.1과 그림 4.1.2이 동일함은 명백하다. ■

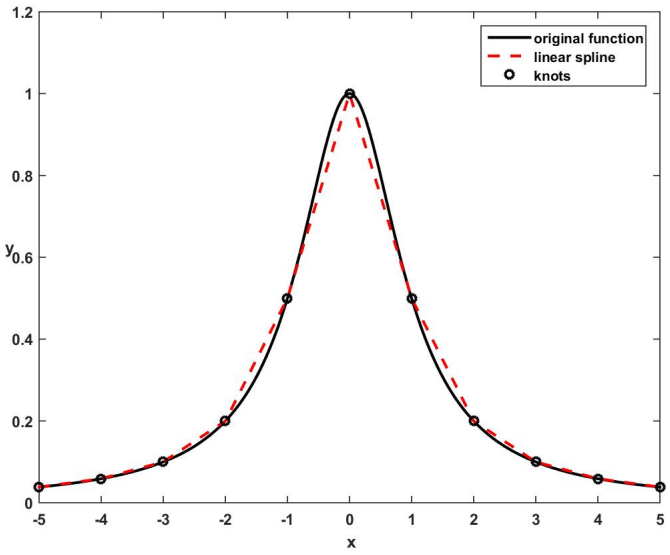


그림 4.3.1. 선형스플라인함수

예제 4.3.2 Python을 사용해서 예제 4.3.1을 다시 다루기 위해서, 다음 Python프로그램 LinearSpline101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

runge = lambda x: 1/( 1+x**2 );
x1 = np.linspace(-5,5,11);
y1 = runge(x1);
flinear = interp1d(x1, y1, kind='linear')
xx = np.linspace(-5,5,1001);
yy = flinear(xx);
ytrue = runge(xx);
err = max(np.abs(yy-ytrue))

# Plotting
fig = plt.figure()
plt.plot(xx,ytrue,'k-', lw=2, label='original function')
plt.plot(xx,yy,'r--', lw=2, label='linear spline')
plt.plot(x1,y1,'ko', lw=2, label='knots')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([-5, 5, 0, 1.2 ])
plt.show()
fig.savefig('LinearSpline101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 4.3.1의 결과와 같다. ■

4.3.2 2차스플라인

소구간 $[x_i, x_{i+1}]$에서 다음 함수를 정의하자.

$$S_i(x) \doteq a_i + b_i[x - x_i] + c_i[x - x_i]^2 \tag{4.3.3}$$

구간 $[x_0, x_n]$에서 2차스플라인함수(quadratic spline function)가 다음과 같다고 하자.

$$S(x) \doteq \sum_{i=0}^{n-1} S_i(x) 1_{[x_i, x_{i+1})} \tag{4.3.4}$$

정의에서 알 수 있듯이, 다음 식들이 성립한다.

$$S_i(x_{i+1}) = f_{i+1} = S_{i+1}(x_{i+1}), \quad (i = 0, 1, \cdots, n-1) \tag{4.3.5}$$

$$S_i'(x_{i+1}) = f_{i+1}' = S_{i+1}'(x_{i+1}), \quad (i = 0, 1, \cdots, n-1) \tag{4.3.6}$$

식 (4.3.6)을 식 (4.3.3)에 대입하면, 다음 식들이 성립함을 알 수 있다.

$$c_i = \frac{2[f_{i+1}' - f_i']}{x_{i+1} - x_i}, \quad (i = 0, 1, \cdots, n-1) \tag{4.3.7}$$

식 (4.3.5)∼식 (4.3.7)을 식 (4.3.3)에 대입하면, 각 $i(= 0, 1, \cdots, n-1)$에 대해서 다음 식이 성립함을 알 수 있다.

$$S_i(x) = \frac{f_{i+1}' - f_i'}{2[x_{i+1} - x_i]}[x - x_i]^2 + f_i'[x - x_i] + f_i \tag{4.3.8}$$

또한, 식 (4.3.8)을 식 (4.3.5)에 대입하면, 다음 식들을 얻는다.

$$f_{i+1}' = -f_i' + \frac{f_{i+1} - f_i}{2[x_{i+1} - x_i]}, \quad (i = 0, 1, \cdots, n-1) \tag{4.3.9}$$

예제 4.3.3 다음과 같은 Runge함수를 살펴보자.

$$f(x) = \frac{1}{1 + x^2} \tag{1}$$

구간 $[-5, 5]$에서 이 Runge함수 f의 2차스플라인함수를 그리기 위해서, 다음 MATLAB 프로그램 QuadraticSpline101.m을 실행하자.

```
% ------------------------------------------------------------------------
% Filename: QuadraticSpline101.m
% Quadratic Spline with n = 10
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, clf
runge = inline('1./(1+x.^2)','x');
drunge = inline('-2*x./(1+x.^2).^2','x'); % derivative
x1 = [-5:1:5]' ;    % x(i-1)
n = length(x1); deltax = x1(2)-x1(1);
y1 = runge(x1);     % y(i-1)
Tdf1 = drunge(x1);
% quadratic spline function
df1(1) = drunge(-5);
for i=1:n-1
```

```
    df1(i+1) = -df1(i)+2*(y1(i+1)-y1(i))/(x1(i+1)-x1(i));
end
xx1 = [-5:0.01:5]'; S1 = xx1-xx1;
nn = length(xx1);
deltaxx = xx1(2)-xx1(1);
multi = round(deltax/deltaxx);
for i=1:n-1
    b1(i) = df1(i);
    c1(i) = (df1(i+1)-df1(i))/(x1(i+1)-x1(i))/2;
    Tb1(i) = Tdf1(i);
    Tc1(i) = (Tdf1(i+1)-Tdf1(i))/(x1(i+1)-x1(i))/2;
    for ii = 0:multi-1
        d1 = (i-1)*multi+ii+1;
        dx = -5+(d1-1)*deltaxx;
        S1(d1) = y1(i)+b1(i)*(dx-x1(i))+c1(i)*(dx-x1(i)).^2;
        TS1(d1) = y1(i)+ Tb1(i)*(dx-x1(i)) ...
                  +Tc1(i)*(dx-x1(i)).^2;
        ytrue(d1) = runge(dx);
    end
end
S1(nn) = y1(n); TS1(nn) = y1(n); ytrue(nn) = runge(x1(n));
errS = max(abs(S1'-ytrue))
errTS = max(abs(TS1-ytrue))
% Plotting
plot(xx1,ytrue,'k-', xx1,S1,'r--',xx1,TS1,'b-.', ...
     x1,y1,'ko','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xtick',-5:1:5)
legend('original function','quadratic spline',...
       'true quad spline','knots')
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
axis([ -5 5 -0.2 1.2 ])
saveas(gcf,'QuadraticSpline101.jpg')
save('QuadraticSpline101.txt','xx1','S1','-ascii')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 QuadraticSpline101.m을 실행하면, 그림 4.3.2를 출력한다. 이 MATLAB프로그램에서는 두 가지 방법들을 사용해서, 2차스플라인함수들을 구하였다. 그 차이는 미분값들을 구하는 방법들에 의한 것이다. 점화식 (4.3.9)를 적용해서 구한 미분값들을 바탕으로 그린 2차스플라인함수가 적색 긴점선으로 그려져 있다. 이 2차스플라인함수는 x가 커질수록 진동하고, 이 스플라인함수에 의한 최대오차 $\max_x |f(x) - S(x)|$는 0.2730이다. 이는 점화식 (4.3.9)가 수치적 안정성을 제공하지 못함을 의미한다. 반면에, 진짜 미분값들을 적용해서 구한 2차스플라인함수가 청색 반점선으로 그려져 있다. 이 2차스플라인함수는 1차미분값이 양수에서 음수로 바뀌는 $x = 0$ 부근에서 좋은 근사식을 제공하지 못함을 알 수 있다. 또한, 이 스플라인함수에 의한 최대오차 $\max_x |f(x) - S(x)|$는 0.2500이다. ■

예제 4.3.4 Python을 사용해서 예제 4.3.3을 다시 다루기 위해서, 다음 Python프로그램

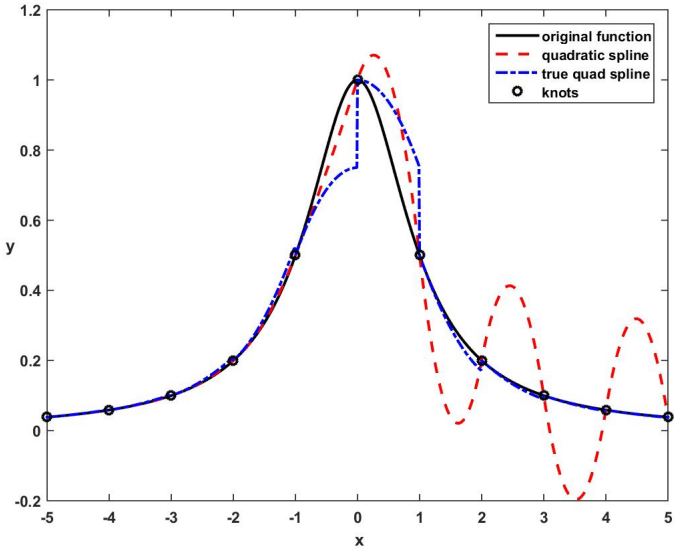


그림 4.3.2. 2차스플라인함수

QuadraticSpline101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

runge = lambda x: 1/( 1+x**2 );
x1 = np.linspace(-5,5,11);
y1 = runge(x1);
fQuadratic = interp1d(x1, y1, kind='quadratic')
xx = np.linspace(-5,5,1001);
yy = fQuadratic(xx);
ytrue = runge(xx);
err = max(np.abs(yy-ytrue))

# Plotting
fig = plt.figure()
plt.plot(xx,ytrue,'k-', lw=2, label='original function')
plt.plot(xx,yy,'r--', lw=2, label='quadratic spline')
plt.plot(x1,y1,'ko', lw=2, label='knots')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([-5, 5, -0.2, 1.2 ])
plt.show()
fig.savefig('QuadraticSpline101Py.jpg')

# End of Program
```

이 Python프로그램 QuadraticSpline101.Py를 실행하면, 그림 4.3.3를 출력한다. 그림 4.3.3에서 알 수 있듯이, 이 Python프로그램은 점 $x = 0$를 기준으로 2차스플라인함수를

구하였다. ■

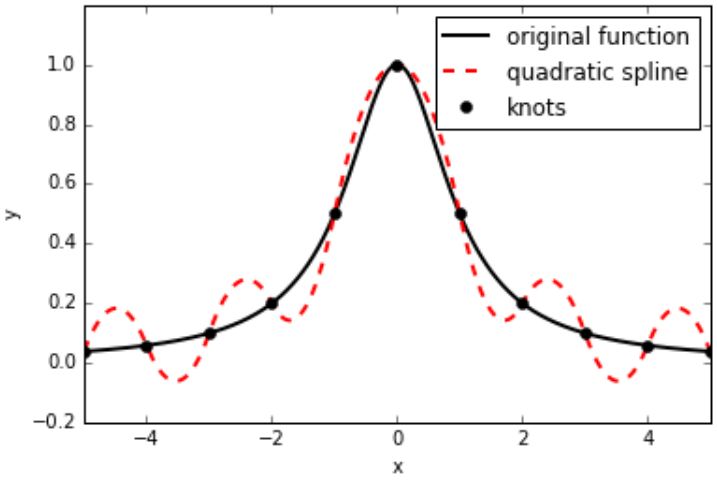


그림 4.3.3. Python에 의한 2차스플라인함수

제4.4절 3차스플라인

현실적으로 가장 널리 사용되는 스플라인함수는 3차스플라인함수(cubic spline function)이다. 이 절에서는 이 3차스플라인함수에 대해서 살펴보자.

다음과 같은 구간 $[a,b]$의 분할 Π를 살펴보자.

$$\Pi \doteq \{a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b\} \tag{4.4.1}$$

여기서 x_i는 마디점이다. 소구간 $[x_i, x_{i+1}]$에서 다음 함수를 정의하자.

$$S_i(x) \doteq a_i + b_i[x - x_i] + c_i[x - x_i]^2 + d_i[x - x_i]^3 \tag{4.4.2}$$

구간 $[x_0, x_n]$에서 3차스플라인함수 $S(x)$를 다음과 같이 정의하자.

$$S(x) \doteq \sum_{i=0}^{n-1} S_i(x) 1_{[x_i, x_{i+1})} \tag{4.4.3}$$

정의에서 알 수 있듯이, 다음 식들이 성립한다.

$$S_i(x_i) = y_i, \quad (i = 0, 1, \cdots, n) \tag{4.4.4}$$

$$S_i(x_{i+1}) = S_{i+1}(x_{i+1}), \quad (i = 0, 1, \cdots, n-2) \tag{4.4.5}$$

$$S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}), \quad (i = 0, 1, \cdots, n-2) \tag{4.4.6}$$

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}), \quad (i = 0, 1, \cdots, n-2) \tag{4.4.7}$$

식 (4.4.2)와 식 (4.4.3)에서 알 수 있듯이, 3차스플라인함수 $S(x)$를 구하기 위해서는 $\{(a_i, b_i, c_i, d_i) \,|\, i = 0, 1, \cdots, n-1\}$를 구해야 한다. 따라서, 미지수는 $4n$개이다. 즉, $4n$개 조건들을 먼저 결정해야 한다. 식 (4.4.4)에서 $[n+1]$개, 식 (4.4.5)에서 $[n-1]$개, 식 (4.4.6)에서 $[n-1]$개, 그리고 식 (4.4.7)에서 $[n-1]$개 등 $[4n-2]$개 조건들이 형성된다. 따라서, 2개 조건들을 더 구해야 한다. 이를 끝점조건이라고 한다. 어떻게 끝점조건을 정하느냐에 따라 다른 3차스플라인함수가 구해진다.

스플라인함수 $S(x)$는 식 (4.4.7)을 만족하는 구분적 3차함수이므로, $S''(x)$는 연속인 구분적 선형함수이다. 따라서, 다음 식이 성립한다.

$$S_i''(x) = \frac{x - x_i}{x_{i+1} - x_i} S_i''(x_{i+1}) + \frac{x_{i+1} - x}{x_{i+1} - x_i} S_i''(x_i) \tag{4.4.8}$$

다음 변수들을 정의하자.

$$h_i \doteq x_{i+1} - x_i, \quad g_i \doteq S''(x_i) \tag{4.4.9}$$

다음 식이 성립한다.

$$S_i''(x) = \frac{g_i}{h_i}[x_{i+1} - x] + \frac{g_{i+1}}{h_i}[x - x_i], \quad (x_i \le x \le x_{i+1}) \tag{4.4.10}$$

식 (4.4.10)의 양변을 두 번 적분하면, 다음 식을 얻는다.

$$S_i(x) = \frac{1}{6}\frac{g_i}{h_i}[x_{i+1} - x]^3 + \frac{1}{6}\frac{g_{i+1}}{h_i}[x - x_i]^3 + \alpha_i[x_{i+1} - x] + \beta_i[x - x_i], \quad (x_i \le x \le x_{i+1}) \tag{4.4.11}$$

여기서 α_i와 β_i는 적분상수들이다. 식 (4.4.11)에 점 $x = x_i$와 점 $x = x_{i+1}$를 대입하면, 다음

식들을 얻는다.

$$\alpha_i = \frac{1}{h_i}\left[y_i - \frac{1}{6}g_i h_i^2\right], \quad \beta_i = \frac{1}{h_i}\left[y_{i+1} - \frac{1}{6}g_{i+1}h_i^2\right] \tag{4.4.12}$$

식 (4.4.12)를 식 (4.4.11)에 대입하면, 각 $x(\in [x_i, x_{i+1}])$에 대해서 다음 식이 성립함을 알 수 있다.

$$\begin{aligned} S_i(x) = &\frac{1}{6}\frac{g_i}{h_i}[x_{i+1} - x]^3 + \frac{1}{6}\frac{g_{i+1}}{h_i}[x - x_i]^3 \\ &+ \frac{1}{h_i}\left[y_i - \frac{1}{6}g_i h_i^2\right][x_{i+1} - x] + \frac{1}{h_i}\left[y_{i+1} - \frac{1}{6}g_{i+1}h_i^2\right][x - x_i] \end{aligned} \tag{4.4.13}$$

식 (4.4.13)에서 미지인 모수들은 $\{g_i\}$ 뿐이다. 지금부터 이 모수들을 구하기로 하자. 식 (4.4.13)의 양변을 미분하면, 다음 식이 성립함을 알 수 있다.

$$\begin{aligned} S_i'(x) = &-\frac{1}{2}\frac{g_i}{h_i}[x_{i+1} - x]^2 + \frac{1}{2}\frac{g_{i+1}}{h_i}[x - x_i]^2 \\ &- \frac{1}{h_i}\left[y_i - \frac{1}{6}g_i h_i^2\right] + \frac{1}{h_i}\left[y_{i+1} - \frac{1}{6}g_{i+1}h_i^2\right] \end{aligned} \tag{4.4.14}$$

식 (4.4.14)에 점 $x = x_i$를 대입하면, 다음 식을 얻는다.

$$S_i'(x_i) = -\frac{g_i}{3}h_i - \frac{g_{i+1}}{6}h_i + \frac{y_{i+1} - y_i}{h_i} \tag{4.4.15}$$

식 (4.4.14)에서 알 수 있듯이, 다음 식이 성립한다.

$$\begin{aligned} S_{i-1}'(x) = &-\frac{1}{2}\frac{g_{i-1}}{h_{i-1}}[x_i - x]^2 + \frac{1}{2}\frac{g_i}{h_{i-1}}[x - x_{i-1}]^2 \\ &- \frac{1}{h_{i-1}}\left[y_{i-1} - \frac{1}{6}g_{i-1}h_{i-1}^2\right] + \frac{1}{h_{i-1}}\left[y_i - \frac{1}{6}g_i h_{i-1}^2\right] \end{aligned} \tag{4.4.16}$$

식 (4.4.16)에 점 $x = x_i$를 대입하면, 다음 식을 얻는다.

$$S_{i-1}'(x_i) = \frac{g_i}{3}g_{i-1} + \frac{g_{i-1}}{6}h_{i-1} + \frac{y_i - y_{i-1}}{h_{i-1}} \tag{4.4.17}$$

식 (4.4.15)와 식 (4.4.17)을 식 (4.4.6)에 대입하면, 다음 식들을 얻는다.

$$h_{i-1}g_{i-1} + 2[h_{i-1} + h_i]g_i + h_i g_{i+1} = e_i, \quad (i = 1, 2, \cdots, n-1) \tag{4.4.18}$$

여기서 e_i 는 다음과 같다.

$$e_i \doteq 6\left[\frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right], \quad (i = 1, 2, \cdots, n-1) \tag{4.4.19}$$

다음 행렬과 벡터들을 정의하자.

$$H \doteq \begin{bmatrix} 2[h_0+h_1] & h_1 & 0 & \cdots & 0 & 0 & 0 \\ h_1 & 2[h_1+h_2] & h_2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & h_{n-3} & 2[h_{n-3}+h_{n-2}] & h_{n-2} \\ 0 & 0 & 0 & \cdots & 0 & h_{n-2} & 2[h_{n-2}+h_{n-1}] \end{bmatrix} \tag{4.4.20}$$

$$\boldsymbol{g} \doteq \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_{n-2} \\ g_{n-1} \end{bmatrix}, \quad \boldsymbol{e} \doteq \begin{bmatrix} e_1 - h_0 g_0 \\ e_2 \\ \vdots \\ e_{n-2} \\ e_{n-1} - h_{n-1} g_n \end{bmatrix} \tag{4.4.21}$$

따라서, 연립방정식 (4.4.18)을 다음과 같이 쓸 수 있다.

$$H\boldsymbol{g} = \boldsymbol{e} \tag{4.4.22}$$

여기서 H는 삼중대각행렬(tridiagonal matrix)임을 상기하라. 이 삼중대각행렬 H의 대각원소들이 충분히 크기 때문에, 행렬 H는 정칙이다. 따라서, 만약 g_0와 g_1이 주어지면, 방정식 (4.4.22)를 풀 수 있다. 이 g_0과 g_1은 끝점조건들로 주어진다. 방정식 (4.4.22)를 효율적으로 풀기 위해서는 Crout-Dolittle법을 사용하는 것이 좋다. 이에 대한 자세한 내용은 **IM&F10**[6]의 제3.8절을 참조하라. 방정식 (4.4.22)를 풀어서 구한 $\{g_i\}$를 식 (4.4.13)에 대입하면, 다음 식을 얻는다.

$$S_i(x) = a_i + b_i[x - x_i] + c_i[x - x_i]^2 + d_i[x - x_i]^3 \tag{4.4.23}$$

여기서 각 계수는 다음과 같다.

$$a_i = y_i, \quad b_i = \frac{y_{i+1} - y_i}{h_i} - \frac{1}{6}h_i[2g_i + g_{i+1}] \tag{4.4.24}$$

$$c_i = \frac{1}{2}g_i, \quad d_i = \frac{1}{6h_i}[g_{i+1} - g_i] \tag{4.4.25}$$

끝점조건들을 구하는 첫 번째 방법은 소구간 $[x_0, x_2]$에서 3차스플라인 하나를, 그리고 소구간 $[x_{n-2}, x_n]$에서 3차스플라인 하나를 사용하는 것이다. 이 경우에, 마디점들은 $x_0, x_2, \cdots, x_{n-2}, x_n$이다. 따라서, 이 조건을 무마디점조건(not-a-knot condition)이라고 부른다. 정의에서 알 수 있듯이, 3차스플라인은 3차 멱함수이다. 따라서, $S''(x_2)$와 $S''(x_1)$을 직선으로 외삽(extrapolation)해서 $S''(x_0)$를 구하고 또한 $S''(x_{n-2})$와 $S''(x_{n-1})$을 직선으로 외삽해서 $S''(x_n)$을 구하면 이 무마디점조건이 만족된다. 이 스플라인함수를 외삽스플라인함수(extrapolated spline function)라고 부른다. 점 x_1과 점 x_2에서 2차 도함수 $S''(x)$가 연속이다. 따라서, 이 외삽스플라인함수는 다른 스플라인함수에 비해서 평활하다. 외삽스플라인함수의 끝점조건들을 다음과 같이 쓸 수 있다.

$$\left[3h_0 + 2h_1 + \frac{h_0^2}{h_1}\right] g_1 + \left[h_1 - \frac{h_0^2}{h_1}\right] g_2 = \frac{y_1 - y_0}{h_0} \tag{4.4.26}$$

$$\left[h_{n-2} - \frac{h_{n-1}^2}{h_{n-2}}\right] g_{n-2} + \left[2h_{n-2} + 3h_{n-1} + \frac{h_{n-1}^2}{h_{n-2}}\right] g_{n-1} = \frac{y_n - y_{n-1}}{h_{n-1}} \tag{4.4.27}$$

예제 4.4.1 다음과 같은 Runge함수 f를 살펴보자.

$$f(x) = \frac{1}{1 + x^2} \tag{1}$$

구간 $[-5, 5]$에서 이 Runge함수 f의 외삽스플라인함수를 그리기 위해서, 다음 MATLAB 프로그램 CubicSpline101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: CubicSpline101.m
% Cubic Spline with n = 10 using MATLAB function interp1
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, clf
runge = inline('1./( 1+x.^2 )','x');
x1 = [-5:1:5]' ;    % x(i-1)
y1 = runge(x1);     % y(i-1)
xx = -5:0.02:5;
yy = interp1(x1,y1,xx,'spline');
ytrue = runge(xx);
err = max(abs(yy-ytrue))
```

```
% plot
plot(xx,ytrue,'k-',xx,yy,'r--',x1,y1,'ko','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xtick',-5:1:5)
legend('original function','cubic spline','knots')
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
axis( [ -5 5 0 1.2 ])
saveas(gcf,'CubicSpline101.jpg')
save('CubicSpline101.txt','xx','ytrue','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB 프로그램 CubicSpline101.m 에서는 MATLAB 함수 interp1.m 에 모수로 'spline'을 지정하였다. 이는 외삽스플라인함수를 출력하기 위한 것이다. MATLAB 함수 spline.m을 사용해도 같은 결과를 얻는다.

이 MATLAB 프로그램 CubicSpline101.m을 실행하면, 구간 $[-5, 5]$ 에서 함수 $f(x)$ 와 외삽스플라인함수 사이의 최대오차 $\max_x |f(x) - S(x)|$ 는 0.0220 임을 알 수 있다다. 또한, 이 MATLAB 프로그램을 실행하면, 그림 4.4.1를 출력한다. 그림 4.4.1에서 Runge 함수 $f(x)$ 는 흑색 실선으로, 그리고 선형스플라인함수는 적색 긴점선으로 그려져 있다. 외삽스플라인함수는 보간노드들을 3차 멱함수로 연결한 함수이다. 이 외삽스플라인함수의 2차 도함수는 각 점에서 연속임을 상기하라. 따라서, 이 외삽스플라인함수는 예제 4.1.3의 구분적 Hermite 보간함수보다 더 평활하다. ■

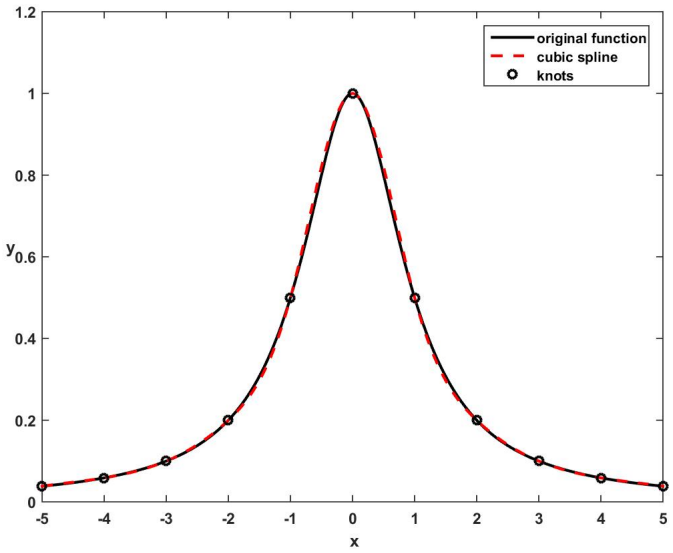


그림 4.4.1. 외삽스플라인함수

예제 4.4.2 Python을 사용해서 예제 4.4.1을 다시 다루기 위해서, 다음 Python 프로그램

CubicSpline101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt

runge = lambda x: 1/( 1+x**2 );
x1 = np.linspace(-5,5,11);
y1 = runge(x1);
fcubic = interp1d(x1, y1, kind='cubic')
xx = np.linspace(-5,5,1001);
yy = fcubic(xx);
ytrue = runge(xx);
err = max(np.abs(yy-ytrue))

# Plotting
fig = plt.figure()
plt.plot(xx,ytrue,'k-', lw=2, label='original function')
plt.plot(xx,yy,'r--', lw=2, label='cubic spline')
plt.plot(x1,y1,'ko', lw=2, label='knots')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper right', numpoints=1)
plt.axis([-5, 5, -0.2, 1.2 ])
plt.show()
fig.savefig('CubicSpline101Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 4.4.1의 결과와 같다. ■

끝점조건들을 구하는 두 번째 방법은 다음과 같이 각 끝점에서 1차 미분값을 지정하는 것이다.

$$S'(x_0) = f'_0, \quad S'(x_n) = f'_n \tag{4.4.28}$$

식 (4.4.28)을 만족하는 스플라인함수를 고정스플라인함수(clamped spline function)라고 부른다. 만약 함수 f의 도함수를 알고있는 경우에는 이 스플라인함수를 사용하는 것이 가장 효율적이다. 이 경우에는 다음 식들이 성립한다.

$$g_0 = \frac{3}{h_0}\left[\frac{y_1 - y_0}{h_0} - f'_0\right] - \frac{1}{2}g_1 \tag{4.4.29}$$

$$g_n = \frac{3}{h_{n-1}}\left[f'_n - \frac{y_n - y_{n-1}}{h_{n-1}}\right] - \frac{1}{2}g_{n-1} \tag{4.4.30}$$

예제 4.4.3 다음과 같은 Runge함수 f를 살펴보자.

$$f(x) = \frac{1}{1+x^2} \tag{1}$$

구간 $[-5, 5]$에서 이 Runge함수 f의 고정스플라인함수를 그리기 위해서, 다음 MATLAB 프로그램 ClampedCubicSpline101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: ClampedCubicSpline101.m
% Clamped Cubic Spline with n = 10 using MATLAB function interp1
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, clf
runge = inline('1./( 1+x.^2 )','x');
drunge = inline('-2*x./( 1+x.^2 ).^2','x');
% input and intermediate values
x1 = [-5:1:5]' ;    % {x(i-1)}
y1 = runge(x1);     % {y(i-1)}
deltax = x1(2)-x1(1);
n = length(x1); n1 = n-1;
df0 = drunge(x1(1));
dfn = drunge(x1(n));
h = diff(x1);
ydiff = diff(y1)./h;
aaa = h(2:n1-1);
bbb = 2*( h(1:n1-1)+h(2:n1) );
ccc = h(2:n1);
rhs = 6*diff(ydiff);
% end-point constraints
bbb(1) = bbb(1)-h(1)/2;
bbb(n1-1) = bbb(n1-1)-h(n1)/2;
rhs(1) = rhs(1) - 3*(ydiff(1)-df0);
rhs(n1-1) = rhs(n1-1) - 3*(dfn-ydiff(n1));
% applying Crout-Dolittle method
for ii = 2:n1-1
    dum = aaa(ii-1)/bbb(ii-1);
    bbb(ii) = bbb(ii) - dum*ccc(ii-1);
    rhs(ii) = rhs(ii) - dum*rhs(ii-1);
end
g(n1) = rhs(n1-1)/bbb(n1-1);
for ii = n1-2:-1:1
    g(ii+1) = ( rhs(ii)-ccc(ii)*g(ii+2) )/bbb(ii);
end
g(1) = 3*( ydiff(1)-df0 )/h(1) - g(2)/2;
g(n) = 3*( dfn-ydiff(n1) )/h(n1) - g(n1)/2;
for ii = 0:n1-1
    a1(ii+1) = y1(ii+1);
    b1(ii+1) = ydiff(ii+1)-h(ii+1)*(2*g(ii+1)+g(ii+2))/6;
    c1(ii+1) = g(ii+1)/2;
    d1(ii+1) = (g(ii+2)-g(ii+1))/(6*h(ii+1));
end
% Calculating cubic spline values
deltaxx = deltax/50;
ndum = round(deltax/deltaxx)
xx1 = [-5:1/50:5]';
for ii=0:n1-1
```

```
    for jj = 0:ndum-1
        ijdum = ii*ndum+jj+1;
        xdum = jj*deltaxx;
        yy1(ijdum) = a1(ii+1) + xdum*( b1(ii+1) ...
                    + xdum*(c1(ii+1)+xdum*d1(ii+1)));
    end
end
yy1(n1*ndum+1) = y1(n);
ytrue = runge(xx1);
err = max(abs(yy1'-ytrue))
% Plotting
plot(xx1,ytrue,'k-',xx1,yy1,'r--',x1,y1,'ko','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold','xtick',-5:1:5)
legend('original function','clamped cubic spline','knots')
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
axis( [ -5 5 0 1.2 ])
saveas(gcf,'ClampedCubicSpline101.jpg')
save('ClampedCubicSpline101.txt','yy1','ytrue','-ascii')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 ClampedCubicSpline101.m에서는 고정스플라인함수를 그리기 위해서, 끝점조건들 (4.4.29)과 식 (4.4.30)을 만족하는 연립방정식 (4.4.22)를 풀었다. 또한, Crout-Dolittle법을 적용해서 삼중대각행렬을 효율적으로 분해하였다.

이 MATLAB프로그램 ClampedCubicSpline101.m을 실행하면, 구간 $[-5,5]$에서 함수 $f(x)$와 외삽스플라인함수 사이의 최대오차 $\max_x |f(x) - S(x)|$는 0.0220임을 알 수 있다. 또한, 이 MATLAB프로그램을 실행하면, 그림 4.4.2을 출력한다. 그림 4.4.2에서 Runge함수 $f(x)$는 흑색 실선으로, 그리고 선형스플라인함수는 적색 긴점선으로 그려져 있다. 그림 4.4.2의 고정스플라인함수는 그림 4.4.1의 외삽스플라인함수와 크게 달라보이지 않는다. ■

예제 4.4.4 Python을 사용해서 예제 4.4.3을 다시 다루기 위해서, 다음 Python프로그램 ClampedCubicSpline101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

runge = lambda x: 1/( 1+x**2 );
drunge = lambda x: -2*x/( 1+x**2 )**2;

# input and intermediate values
x1 = np.linspace(-5,5,11);
y1 = runge(x1);
deltax = x1[1]-x1[0];
n = len(x1); n1 = n-1;
```

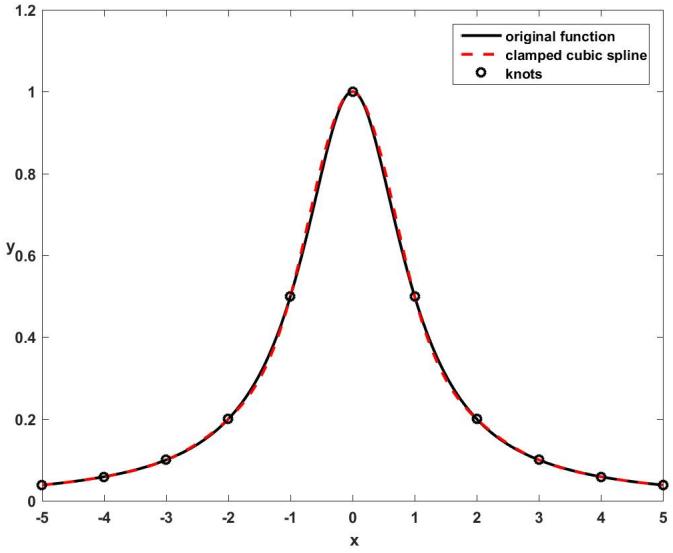


그림 4.4.2. 고정스플라인함수

```
df0 = drunge(x1[0]);
dfn = drunge(x1[n-1]);
h = np.diff(x1);
ydiff = np.diff(y1)/h;
aaa = h[1:n1-1]
bbb = 2*( h[0:n1-1] + h[1:n1] );
ccc = h[1:n1];
rhs = 6*np.diff(ydiff);

# end-point constraints
bbb[0] = bbb[0]-h[0]/2;
bbb[n1-2] = bbb[n1-2]-h[n1-1]/2;
rhs[0] = rhs[0] - 3*(ydiff[0]-df0);
rhs[n1-2] = rhs[n1-2] - 3*(dfn-ydiff[n1-1]);

# applying Crout-Dolittle method
for ii in range(1,n1-1):
    dum = aaa[ii-1]/bbb[ii-1];
    bbb[ii] = bbb[ii] - dum*ccc[ii-1];
    rhs[ii] = rhs[ii] - dum*rhs[ii-1];
g = [0.0]*n;
g[n1-1] = rhs[n1-2]/bbb[n1-2];
for ii in range(n1-3,-1,-1):
    g[ii+1] = ( rhs[ii]-ccc[ii]*g[ii+2] )/bbb[ii];
g[0] = 3*( ydiff[0]-df0 )/h[0] - g[1]/2;
g[n-1] = 3*( dfn-ydiff[n1-1] )/h[n1-1] - g[n1-1]/2;

a1 = [0.0]*n1; b1 = [0.0]*n1; c1 = [0.0]*n1; d1 = [0.0]*n1;
for ii in range(0,n1):
    a1[ii] = y1[ii];
    b1[ii] = ydiff[ii]-h[ii]*(2*g[ii]+g[ii+1])/6;
    c1[ii] = g[ii]/2;
    d1[ii] = (g[ii+1]-g[ii])/(6*h[ii]);

# Calculating cubic spline values
deltaxx = deltax/50;
ndum = int(round(deltax/deltaxx))   # integer part
```

```
xx1 = np.linspace(-5,5,501);
yy1 = [0.0]*501;
for ii in range(0,n1):
    for jj in range(0,ndum):
        ijdum = ii*ndum+jj;
        xdum = jj*deltaxx;
        yy1[ijdum] = a1[ii] + xdum*(b1[ii]+xdum*(c1[ii]+xdum*d1[ii]));
yy1[n1*ndum] = y1[n-1];
ytrue = runge(xx1);
err = max(np.abs(yy1-ytrue));

# Plotting
fig = plt.figure()
plt.plot(xx1,ytrue,'k-', lw=2, label='original function')
plt.plot(xx1,yy1,'r--', lw=2, label='clamped spline')
plt.plot(x1,y1,'ko', lw=2, label='knots')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper left', numpoints=1)
plt.axis([-5, 5, 0, 1.2 ])
plt.show()
fig.savefig('ClampedCubicSpline101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 4.4.3의 결과와 같다. ■

예제 4.4.5 Python을 사용해서 예제 4.4.3을 다시 다루기 위해서, 다음 Python프로그램 ClampedCubicSpline102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy.interpolate import CubicSpline
import matplotlib.pyplot as plt

runge = lambda x: 1/( 1+x**2 );
x1 = np.linspace(-5,5,11);
y1 = runge(x1);
fclamped = CubicSpline(x1,y1,axis=0,bc_type='clamped',extrapolate=None)
xx = np.linspace(-5,5,1001);
yy = fclamped(xx);
ytrue = runge(xx);
err = max(np.abs(yy-ytrue))

# Plotting
fig = plt.figure()
plt.plot(xx,ytrue,'k-', lw=2, label='original function')
plt.plot(xx,yy,'r--', lw=2, label='clamped cubic')
plt.plot(x1,y1,'ko', lw=2, label='knots')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper left', numpoints=1)
plt.axis([-5, 5, -0.2, 1.2 ])
plt.show()
```

```
fig.savefig('ClampedCubicSpline102Py.jpg')

# End of Program
```

이 Python프로그램에서는 Python모듈 scipy.interpolate.CubicSpline의 옵션 'clapmped'를 사용해서 고정스플라인함수를 구한다. 이 Python프로그램을 수행한 결과는 예제 4.4.3의 결과와 같다. ■

끝점조건들을 구하는 세 번째 방법은 다음 조건들을 부여하는 것이다.

$$g_0 \doteq S''(x_0) = 0, \quad g_n \doteq S''(x_n) = 0 \tag{4.4.31}$$

식 (4.4.31)을 만족하는 스플라인함수는 자연스플라인함수(natural spline function)이다. 자연스플라인함수가 특별히 관심을 끄는 이유는 앞에서 설명했듯이 총곡률을 최소화하기 때문이다. 자연스플라인함수의 끝점조건들을 다음과 같이 쓸 수 있다.

$$2[h_0 + h_1]g_1 + h_1 g_2 = \frac{y_1 - y_0}{h_0} \tag{4.4.32}$$

$$h_{n-2}g_{n-2} + 2[h_{n-2} + h_{n-1}]g_{n-1} = \frac{y_n - y_{n-1}}{h_{n-1}} \tag{4.4.33}$$

예제 4.4.6 다음과 같은 Runge함수 f를 살펴보자.

$$f(x) = \frac{1}{1+x^2} \tag{1}$$

구간 $[-5, 5]$에서 이 Runge함수 f의 자연스플라인함수를 그리기 위해서, 다음 MATLAB 프로그램 NaturalCubicSpline101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: NaturalCubicSpline101.m
% Natural Cubic Spline with n = 10
% using Spline Toolbox function csapi
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, clf
runge = @(x) 1./( 1+x.^2 );
x1 = [-5:1:5];      % x(i-1)
y1 = runge(x1);     % y(i-1)
xx = -5:0.02:5;
yy = csapi(x1,y1,xx);
ytrue = runge(xx);
err = max(abs(yy-ytrue))
```

```
% Plotting
plot(xx,ytrue,'k-',xx,yy,'r--',x1,y1,'ko','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold','xtick',-5:1:5)
legend('original function','natural cubic spline','knots')
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
axis( [ -5 5 0 1.2 ])
saveas(gcf,'NaturalCubicSpline101.jpg')
save('NaturalCubicSpline101.txt','yy','ytrue','-ascii')
% End of program
% --------------------------------------------------------------------
```

이 MATLAB프로그램 NaturalCubicSpline101.m에서는 자연스플라인함수를 그리기 위해서, MATLAB Toolbox Spline의 함수 csapi를 사용하였다.

이 MATLAB프로그램 NaturalCubicSpline101.m을 실행하면, 구간 $[-5, 5]$에서 함수 $f(x)$와 외삽스플라인함수 사이의 최대오차 $\max_x |f(x) - S(x)|$는 0.0220 임을 알 수 있다. 또한, 이 MATLAB프로그램을 실행하면, 그림 4.4.3을 출력한다. 그림 4.4.3에서 Runge함수 $f(x)$는 흑색 실선으로, 그리고 선형스플라인함수는 적색 긴점선으로 그려져 있다. 그림 4.4.3의 자연스플라인함수는 그림 4.4.2의 외삽스플라인함수와 크게 달라보이지 않는다. ■

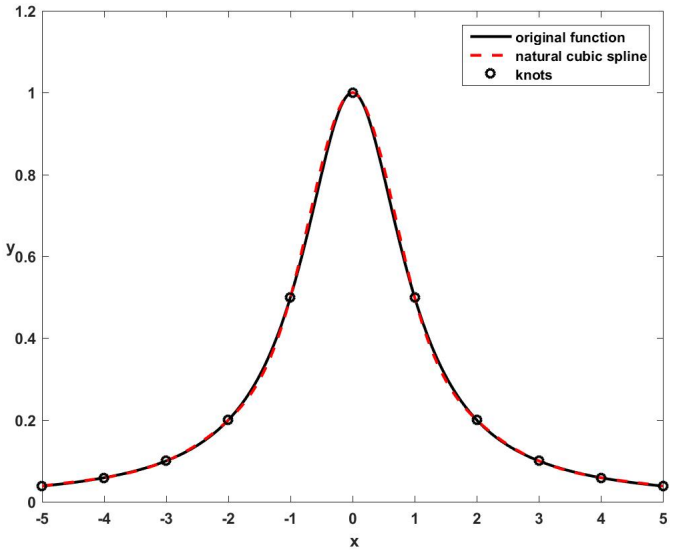


그림 4.4.3. 자연스플라인함수

예제 4.4.7 Python을 사용해서 예제 4.4.6을 다시 다루기 위해서, 다음 Python프로그램 NaturalCubicSpline101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
```

```
"""

import numpy as np
from scipy.interpolate import CubicSpline
import matplotlib.pyplot as plt

runge = lambda x: 1/( 1+x**2 );
x1 = np.linspace(-5,5,11);
y1 = runge(x1);
fnatural = CubicSpline(x1,y1,axis=0,bc_type='natural',extrapolate=None)
xx = np.linspace(-5,5,1001);
yy = fnatural(xx);
ytrue = runge(xx);
err = max(np.abs(yy-ytrue))

# Plotting
fig = plt.figure()
plt.plot(xx,ytrue,'k-', lw=2, label='original function')
plt.plot(xx,yy,'r--', lw=2, label='natural cubic')
plt.plot(x1,y1,'ko', lw=2, label='knots')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper left', numpoints=1)
plt.axis([-5, 5, -0.2, 1.2 ])
plt.show()
fig.savefig('NaturalCubicSpline102Py.jpg')

# End of Program
```

이 Python프로그램에서는 Python모듈 scipy.interpolate.CubicSpline의 옵션 'natural'를 사용해서 자연스플라인함수를 구한다. 이 Python프로그램을 수행한 결과는 예제 4.4.6의 결과와 같다. ■

제4.5절 정규화

4.5.1 Pareto효율

어떤 자원배분상태에서 다른 사람에게 손해가 가지 않으면서 어떤 한 사람에게 이득이 되는 변화를 만들어내는 것이 불가능할 때, 이 자원배분상태를 Pareto효율성(Pareto efficiency) 또는 Pareto최적성(Pareto optimality)이라 한다. 이탈리아의 엔지니어 겸 경제학자였던 Vilfredo Pareto (1848–1923)가 경제효율성과 수입분배(income distribution)를 설명하는데 이 개념을 도입했다. 오늘날에는 경제학, 공학, 생명과학 등에서 이 개념이 사용되고 있다. 만약 어떤 다른 구성원의 희생없이 한 구성원의 상황이 나아지는 자원배분이 있다면, 이러한 변화량을 Pareto개선(Pareto improvement)이라 한다. 따라서, 더 이상 Pareto개선이 없는 상태를 Pareto효율적 또는 Pareto최적이라고 한다. 만약 자원배분상태가 Pareto효율적이지

않다면, Pareto개선 가능성이 있다. 즉, 재분배를 통해서 다른 구성원의 복지를 줄이지 않고 어떤 구성원의 복지를 개선할 수 있다. 이상적인 조건 하에서 자유시장체제는 Pareto효율적 균형상태로 수렴한다. 즉, Walrasian균형은 Pareto효율적이다. 이를 후생경제학(welfare economics)의 근본정리라 부른다. 일반적으로 경제학에서 나타나는 정리의 증명에서 그러하듯이, 이 수리적 증명은 강한 가정을 바탕으로 한다. 즉, 이 증명에서는 시장에 가능한 모든 상품들이 존재하므로 외부효과가 없고, 시장은 완전한 균형상태에 있으며, 시장은 완전경쟁적이고, 거래비용을 무시할 수 있으며, 시장참여자들에게는 완전한 정보가 주어진다고 가정한다. Greenwald-Stigilitz정리 [25]에서 알 수 있듯이, 완전한 정보가 주어지지 않거나 불완전시장(incomplete market)에서는 시장은 Pareto효율적이 아니다. 따라서, 실제로는 경제가 이러한 이상적 조건들에서 얼마나 떨어져 있느냐가 경제정책에 반영되어야 한다.

비효율적 배분상태에서 효율적 배분상태로 움직인다고 해서 반드시 Pareto개선이 이루어지는 것은 아니라는 점에 유의해야 한다. Pareto효율을 달성하기 위한 배분상태의 변화때문에 손해를 입는 대상을 없게 하려면, 일부 구성원에게 보상을 해주어야 하는 경우도 있다. 예를 들어, 경제정책을 바꾸어서 독점을 제거함으로써 시장이 경쟁적이고 효율적이 되면, 독점자는 상황이 나빠지나 이러한 독점자의 손실은 다른 구성원들의 효율성 증가분에 의해 상쇄되고도 남을 것이다. 이러한 경우에는 다른 구성원들의 효율성을 증가시키면서, 즉 Pareto개선을 시키면서, 독점자 자신은 손실을 보상 받을 수 있다. 실제 세계에서 이러한 보상은 의도하지 않은 결과를 낳는다. 이러한 보상은 독점자가 그러한 보상을 예측하고 그에 따라 행동을 변화시킴으로써, 시간 흐름에 따라 정책적 유인구조를 왜곡시킬 수도 있다.

Pareto효율성을 쉽게 나타내기 위해서, Pareto프론티어 $P(Y)$를 정의하자. 함수 $f : \mathbb{R}^n \to \mathbb{R}^m$과 가능한 배분들의 컴팩트집합(compact set) $X\,(\subset \mathbb{R}^n)$에 대해서 다음 집합을 정의하자.

$$Y \doteq \{\boldsymbol{y} \in \mathbb{R}^m : \ \boldsymbol{y} = f(\boldsymbol{x}),\ \boldsymbol{y} \in X\ \} \tag{4.5.1}$$

집합 $Y\,(\subset \mathbb{R}^m)$도 가능한 배분들의 컴팩트집합이라고 하자. 만약 배분 $\boldsymbol{y}''\,(\in \mathbb{R}^m)$가 다른 배분 $\boldsymbol{y}'\,(\in \mathbb{R}^m)$보다 선호되면, 식 $\boldsymbol{y}'' \succ \boldsymbol{y}'$로 표기하자. Pareto프론티어는 다음과 같이 정의된다.

$$P(\boldsymbol{y}) \doteq \left\{\boldsymbol{y}' \in Y \mid \{\boldsymbol{y}'' \in Y \mid \boldsymbol{y}'' \succ \boldsymbol{y}',\ \boldsymbol{y}'' \neq \boldsymbol{y}'\ \} = \emptyset\right\} \tag{4.5.2}$$

즉, Pareto프론티어는 Pareto개선 여지가 없는 집합이다.

일반적인 최적화문제에서는 목적함수가 하나이지만, 때때로 목적함수들을 두 개 이상 동시에 고려해야 하는 경우가 있다. 예를 들어, 갓 고등학교를 졸업한 학생들에게 경제원칙이

무엇이냐고 물어보면, 대부분 최소한 희생으로 최대 효과를 얻는 것이라고 한다. 이론적으로 이 문제의 해는 존재하지 않는다. 또한, 어떤 엔지니어가 기계성능을 최대화시키는 동시에 개발비용을 최소화하고자 원하는 경우가 있다. 이 경우 이론적으로 가능할지는 몰라도 이런 해가 존재할 확률은 0일 것이다. 검정문제에서 나오는 Type I오차와 Type II오차를 동시에 최소화하는 것은 이론적으로 불가능하다. 그 이유는 Type I오차와 Type II오차는 서로 역관계를 이루기 때문이다. 즉, 이 두 종류의 오차들을 동시에 최소화할 수 없다. Grenander 현상이란 추정량의 편의와 분산이 서로 역관계를 이룬다는 것이다. 따라서, 편의와 분산을 동시에 최소화할 수는 없다. 이렇게 목적함수가 복수인 최적화문제를 다목적최적화문제 (multi-objective optimization problem)라고 하자. 다목적최적화문제에서 최적해는 목적함수들의 Pareto프론티어 상에 존재할 것이다.

상식적으로 생각해 보아도 Pareto프론티어는 볼록집합(convec set)의 경계점들로 이루어진다. 편의상, 여기서는 각 목적함수를 최대화가 아닌 최소화하는 경우를 살펴보자. 이러한 다목적최적화문제를 푸는 방법들은 다양하다. 첫째, Σ-제약조건법은 어떤 목적함수를 제외한 다른 목적함수들의 최대수준을 미리 정해놓고 그 제약조건 하에서 그 목적함수를 최소화한다. 이 경우에 최대수준을 조금씩 바꾸면서 최적화를 수행하고, 이러한 과정을 모든 목적함수들에 대해서 수행한다. 둘째, 등호제약조건법(equality constraint method)은 한 목적함수를 제외한 나머지 목적함수들에 동일한 값을 부여하고, 그 목적함수를 최소화한다. 셋째, 목적함수들의 가중합(weighted sum of the objective functions) 또는 가중제곱합(weighted sum of squared norm)을 최소화한다.. 그 이외에도 시뮬레이티드어닐링(simulated annealing)이나 유전자알고리즘(genetic algorithm) 등 메타휴리스틱스(meta heuristics) 기법을 사용할 수도 있다.

4.5.2 평활화

어떤 최적화문제는 입력데이터에 매우 민감해서, 입력데이터에 약간의 변화가 있어도 결과가 크게 달라지는 현상을 보인다. 이러한 최적화문제는 적절하지 못하게 설정된 문제(ill-posed problem)이다. 이 경우에는 해에 평활조건이나 노름의 한계 등과 같은 제약을 가함으로써, 안정적인 해를 얻고자 한다. 이러한 과정을 정규화(regularization)이라 한다. 대표적인 정규화로는 Tikhonov정규화, 최대엔트로피를 사용한 정규화, 그리고 Kullback-Leibler정보수를 최소화하는 정규화 등이 있다.

앞에서 자연스플라인(natural spline)을 정의했다. Reinsch (1967)는 자연스플라인이

데이터를 평활화하는 문제의 해답이 됨을 증명했고, 이 증명은 변분법(calculus of variations)에 나오는 Euler-Lagrange방정식을 바탕으로 한다.

다음 식들을 만족하는 집합 $\{(x_i, y_i, \delta y_i) \mid i = 0, 1, \cdots, n\}$ 를 살펴보자.

$$x_0 < x_1 < \cdots < x_n, \quad \delta y_i > 0, \quad (i = 0, 1, \cdots, n) \tag{4.5.3}$$

여기서 n은 3이상인 자연수이다. 주어진 상수 $S(\geq 0)$에 대해서, 다음과 같은 최적화문제를 살펴보자.

$$\text{Minimize} \int_{x_0}^{x_n} \left[g''(x)\right]^2 dx \tag{4.5.4}$$

$$\text{subject to} \sum_{i=0}^{n} \left[\frac{g(x_i) - y_i}{\delta y_i}\right]^2 \leq S. \tag{4.5.5}$$

여기서 g는 $C^2[x_0, x_n]$급이라고 가정하자. 상수 S는 편의상 도입한 잉여항(redundant constant)이다. 이 잉여항이 있으므로써, 평활화의 정도를 조절하는 δy_i를 묵시적으로 재조정(rescaling)할 수 있다. 이 S값을 가중값들 $\left\{1/\left[\delta y_i^2\right]\right\}$에 의존해서 결정한다. 만약 y_i의 표준편차가 존재한다면, 이 값을 δy_i로 사용할 수 있다. 이 경우에 $S \approx n + 1$이다. 또한, 다음 식들이 성립한다고 가정하자.

$$\lim_{h \downarrow 0} g''(x_0 + h) = 0 \tag{4.5.6}$$

$$\lim_{h \downarrow 0} g''(x_n - h) = 0 \tag{4.5.7}$$

식 (4.5.4)와 (4.5.5)으로 구성된 최적화문제의 해를 f라 하고, 다음과 같은 표기법을 사용하기로 하자.

$$f^{(k)}(x_i)_+ \doteq \lim_{h \downarrow 0} f^{(k)}(x_i + h) \tag{4.5.8}$$

$$f^{(k)}(x_i)_- \doteq \lim_{h \downarrow 0} f^{(k)}(x_i - h) \tag{4.5.9}$$

Euler-Lagrange방정식에서 알 수 있듯이, 어떤 상수 $p(\geq 0)$에 대해서 최적해 f는 다음 식들을

만족한다.

$$f(x_i)_- - f(x_i)_+ = 0, \quad (i = 1, 2, \cdots, n-1) \tag{4.5.10}$$

$$f'(x_i)_- - f'(x_i)_+ = 0, \quad (i = 1, 2, \cdots, n-1) \tag{4.5.11}$$

$$f''(x_i)_- - f''(x_i)_+ = 0, \quad (i = 0, 1, , \cdots, n) \tag{4.5.12}$$

$$f'''(x_i)_- - f'''(x_i)_+ = 2p\frac{f(x_i) - y_i}{[\delta y_i]^2}, \quad (i = 0, 1, , \cdots, n) \tag{4.5.13}$$

$$f^{(4)}(x) = 0, \quad (x_i < x < x_{i+1},\ i = 0, 1, \cdots, n-1) \tag{4.5.14}$$

식 (4.5.10) $\sim$ 식 (4.5.14)에서 알 수 있듯이, 최적해 f는 3차 멱함수이며 또한 f, f' 그리고 f''는 연속이다. 따라서, 최적해 f는 3차스플라인이다. 편의상 다음과 같이 표기하자.

$$f''(x_0)_- = f'''(x_0)_- = 0 \tag{4.5.15}$$

$$f''(x_n)_+ = f'''(x_n)_+ = 0 \tag{4.5.16}$$

보조정리 4.5.1

만약 함수 g가 매듭들이 $x_0, x_1, \cdots, x_n$ 인 자연스플라인이고 함수 $h(x)$가 $C^2[x_0, x_n]$ 급이면, 다음 식이 성립한다.

$$\int_{x_0}^{x_n} g''(x)h''(x)dx = \sum_{i=0}^{n} \left[g'''(x_i)_+ - g'''(x_i)_- \right] h(x_i)$$

증명. 부분적분을 이용하면, 다음 식들이 성립함을 알 수 있다.

$$\begin{aligned}
&\int_{x_0}^{x_n} g''(x)h''(x)dx \\
&= \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} g''(x)h''(x)dx \\
&= \sum_{i=0}^{n-1} \left[g''(x_{i+1})h'(x_{i+1}) - g''(x_i)h'(x_i) - \int_{x_i}^{x_{i+1}} g'''(x)h'(x)dx \right] \qquad (1)
\end{aligned}$$

다음 식들이 성립한다.

$$\begin{aligned}
&\int_{x_0}^{x_n} g''(x)h''(x)dx \\
&= \left[g''(x_n)h'(x_n) - g''(x_0)h'(x_0)\right] - \sum_{i=0}^{n-1}\int_{x_i}^{x_{i+1}} g'''(x)h'(x)dx \\
&= -\sum_{i=0}^{n-1}\int_{x_i}^{x_{i+1}} g'''(x)h'(x)dx \qquad (2)
\end{aligned}$$

여기서 첫 번째 등호는 식 (1)에 의해서, 그리고 두 번째 등호는 식 (4.5.6)과 식 (4.5.7)에 의해서 성립한다. 함수 g가 스플라인이므로, 각 소구간 (x_i, x_{i+1})에서 3차 도함수 g'''는 상수이다. 따라서, 식 (2)를 다음과 같이 쓸 수 있다.

$$\int_{x_0}^{x_n} g''(x)h''(x)dx = -\sum_{i=0}^{n-1} g'''(x_i)_+ \left[h(x_{i+1}) - h(x_i)\right] \qquad (3)$$

각 소구간 (x_i, x_{i+1})에서 3차 도함수 g'''는 상수이므로, 식 $g'''(x_i)_+ = g'''(x_{i+1})_-$이 성립한다. 이 식을 식 (3)에 적용하면, 다음 식을 얻는다.

$$\int_{x_0}^{x_n} g''(x)h''(x)dx = \sum_{i=0}^{n-1} g'''(x_i)_+ h(x_i) - \sum_{i=1}^{n} g'''(x_i)_- h(x_i) \qquad (4)$$

식 (4.5.6)과 식 (4.5.7)을 식 (4)에 적용하면, 다음 식을 얻는다.

$$\int_{x_0}^{x_n} g''(x)h''(x)dx = \sum_{i=0}^{n} \left[g'''(x_i)_+ - g'''(x_i)_-\right] h(x_i) \qquad (5)$$

■

명제 4.5.1

제약조건이 식 (4.5.5)인 최적화문제 (4.5.4)의 해는 매듭들이 $x_0, x_1, \cdots, x_n$인 자연스플라인이다.

증명. 여기서는 Euler-Lagrange방정식을 사용하지 않고, 이 명제를 증명하기로 하자.

함수 f가 이 최적화문제의 해이고, 함수 g는 다음 식들을 만족하는 자연스플라인이라고

하자.

$$f(x_i) = g(x_i), \quad (i = 0, 1, \cdots, n) \tag{1}$$

다음 식이 성립함은 자명하다.

$$\sum_{i=0}^{n} \left[\frac{g(x_i) - y_i}{\delta y_i} \right]^2 \leq S \tag{2}$$

즉, 함수 g는 이 최적화문제의 제약조건을 만족한다. 다음 식이 성립한다.

$$\begin{aligned} &\int_{x_0}^{x_n} [f''(x)]^2 dx \\ &= \int_{x_0}^{x_n} [f''(x) - g''(x)]^2 dx + \int_{x_0}^{x_n} [g''(x)]^2 \, dx + 2 \int_{x_0}^{x_n} [f''(x) - g''(x)] g''(x) dx \end{aligned} \tag{3}$$

보조정리 4.5.1에서 함수 $h(x)$ 대신 함수 $f(x) - g(x)$를 대입하면, 다음 식을 얻는다.

$$\int_{x_0}^{x_n} g''(x)[f''(x) - g''(x)]dx = \sum_{i=0}^{n} \left[g'''(x_{i+1})_+ - g'''(x_{i+1})_- \right] [f''(x_i) - g''(x_i)] \tag{4}$$

식 (1)과 식 (4)에서 알 수 있듯이, 다음 식이 성립한다.

$$\int_{x_0}^{x_n} [f''(x) - g''(x)] g''(x) dx = 0 \tag{5}$$

식 (3)과 식 (5)에서 알 수 있듯이, 다음 식이 성립한다.

$$\int_{x_0}^{x_n} [f''(x)]^2 dx = \int_{x_0}^{x_n} [f''(x) - g''(x)]^2 dx + \int_{x_0}^{x_n} [g''(x)]^2 dx \tag{6}$$

다음 부등식이 성립한다고 가정하자.

$$\int_{x_0}^{x_n} [f''(x) - g''(x)]^2 dx > 0 \tag{7}$$

식 (6)과 식 (7)에서 알 수 있듯이, 다음 식이 성립한다.

$$\int_{x_0}^{x_n} [f''(x)]^2 dx > \int_{x_0}^{x_n} [g''(x)]^2 dx \tag{8}$$

식 (2)와 식 (7)에서 알 수 있듯이, 함수 f는 최적해가 아니다. 따라서, 다음 식이 성립해야

한다.

$$\int_{x_0}^{x_n} [f''(x) - g''(x)]^2 dx = 0 \tag{9}$$

식 (9)에서 알 수 있듯이, 다음 식이 성립한다.

$$f''(x) - g''(x) = 0, \quad (x \in [x_0, x_n]) \tag{10}$$

따라서, 다음 식을 만족하는 상수들 a와 b가 존재한다.

$$f(x) - g(x) = a + bx, \quad (x \in [x_0, x_n]) \tag{11}$$

가정에 의해서 $n \geq 3$이다. 따라서, 식 (1), 식 (11) 그리고 항등식정리에 의해서 다음 식이 성립함을 알 수 있다.

$$f(x) = g(x), \quad (x \in [x_0, x_n]) \tag{12}$$

즉, 최적해 f는 매듭들이 $x_0, x_1, \cdots, x_n$인 자연스플라인이다. ■

변분법을 사용하지 않고, 제약조건이 식 (4.5.5)인 최적화문제 (4.5.4)가 식 (4.5.13)을 만족함을 증명하자. 다음과 같은 분리정리가 성립한다. 벡터들 $\boldsymbol{u} \neq \mathbf{0}$와 $\boldsymbol{v} \neq \mathbf{0}$가 선형독립이면, 다음 식들을 만족하는 벡터 $\boldsymbol{w}$가 존재한다.

$$\boldsymbol{w}^t \boldsymbol{u} > 0, \quad \boldsymbol{w}^t \boldsymbol{v} < 0 \tag{4.5.17}$$

이 분리정리의 증명은 **IM&F11**[6]의 제4.4절을 참조하라 식 (4.5.17)의 벡터 $\boldsymbol{w}$가 의미하는 바는 다음과 같다. 만약 함수 g가 최적해가 아니면, 제약조건 (4.5.5)을 만족하면서 목적함수 $\int_{x_0}^{x_n} [g''(x)]^2 \, dx$를 감소시키는 함수 $\tilde{g}$가 존재한다.

명제 4.5.2

만약 $S > 0$이면, 제약조건이 식 (4.5.5)인 최적화문제 (4.5.4)의 해 f에 대해서 다음 식을 만족하는 상수 $p(\geq 0)$가 존재한다.

$$f'''(x_i)_- - f'''(x_i)_+ = 2p \frac{f(x_i) - y_i}{[\delta y_i]^2}, \quad (i = 0, 1, , \cdots, n)$$

증명. 매듭들이 $x_0, x_1, \cdots, x_n$ 인 자연스플라인 g에 대해서, 다음 식들을 만족하는 상수 $p(\geq 0)$가 존재하지 않는다고 가정하자.

$$g'''(x_i)_- - g'''(x_i)_+ = 2p\frac{g(x_i)-y_i}{[\delta y_i]^2}, \quad (i=0,1,,\cdots,n) \tag{1}$$

다음 벡터들을 정의하자.

$$\boldsymbol{u} \doteq \begin{bmatrix} g'''(x_0)_- - g'''(x_0)_+ \\ g'''(x_1)_- - g'''(x_1)_+ \\ \vdots \\ g'''(x_n)_- - g'''(x_n)_+ \end{bmatrix}, \quad \boldsymbol{v} \doteq \begin{bmatrix} \frac{g(x_0)-y_0}{[\delta y_1]^2} \\ \frac{g(x_1)-y_1}{[\delta y_2]^2} \\ \vdots \\ \frac{g(x_n)-y_n}{[\delta y_n]^2} \end{bmatrix} \tag{2}$$

만약 $\boldsymbol{u}$ 또는 $\boldsymbol{v}$가 영벡터이면, 식 (1)을 만족하는 상수 p가 존재한다. 따라서, $\boldsymbol{u}$도 $\boldsymbol{v}$도 영벡터가 아니다. 식 (1)을 만족하는 상수 p가 존재하지 않는다고 가정했으므로, $\boldsymbol{u}$와 $\boldsymbol{v}$는 선형독립이다. 식 (4.5.17)에서 알 수 있듯이, 다음 식들을 만족하는 벡터 $\boldsymbol{w} \doteq [w_0,\ w_1,\ \cdots, w_n\ ]^t$가 존재한다.

$$\sum_{i=0}^{n} w_i \left[g'''(x_i)_- - g'''(x_i)_+\right] > 0, \quad \sum_{i=0}^{n} w_i \frac{g(x_i)-y_i}{[\delta y_i]^2} < 0 \tag{3}$$

구간 $[x_0\,,\, x_n\,]$에서 C^2급인 함수 $h(x)$가 다음 식들을 만족한다고 하자.

$$h(x_i) = w_i, \quad (i=0,1,\cdots,n) \tag{4}$$

식 (3)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\sum_{i=0}^{n} h(x_i) \left[g'''(x_i)_- - g'''(x_i)_+\right] > 0, \quad \sum_{i=0}^{n} h(x_i) \frac{g(x_i)-y_i}{[\delta y_i]^2} < 0 \tag{5}$$

다음 식이 성립한다.

$$\begin{aligned} &\int_{x_0}^{x_n} [g''(x) + \lambda h''(x)]^2 dx \\ &= \int_{x_0}^{x_n} [g''(x)]^2 dx + \lambda^2 \int_{x_0}^{x_n} [h''(x)]^2\, dx + 2\lambda \int_{x_0}^{x_n} g''(x)h''(x)dx \end{aligned} \tag{6}$$

보조정리 4.5.1과 식 (5)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\int_{x_0}^{x_n} g''(x)h''(x)dx = \sum_{i=0}^{n} \left[g'''(x_i)_+ - g'''(x_i)_- \right] h(x_i) < 0 \tag{7}$$

따라서, 다음 식들을 만족하는 λ_1이 존재한다.

$$0 < \lambda_1 < \frac{-2\int_{x_0}^{x_n} g''(x)h''(x)dx}{\int_{x_0}^{x_n}[g''(x)]^2dx} \tag{8}$$

식 (8)을 만족하는 λ_1에 대해서, 다음 식이 성립한다.

$$\int_{x_0}^{x_n}[g''(x) + \lambda_1 h''(x)]^2dx < \int_{x_0}^{x_n}[g''(x)]^2dx \tag{9}$$

같은 방법을 사용해서, 다음 식들을 만족하는 λ_2가 존재함을 증명할 수 있다.

$$0 < \lambda_2 < \frac{-2\sum_{i=0}^{n} h(x_i)\frac{g(x_i)-y_i}{[\delta y_i]^2}}{\sum_{i=0}^{n}\left\{\frac{h(x_i)}{[\delta y_i]^2}\right\}^2} \tag{10}$$

식 (10)을 만족하는 λ_2에 대해서, 다음 식이 성립한다.

$$\sum_{i=0}^{n}\left\{\frac{g(x_i) + \lambda_2 h(x_i) - y_i}{[\delta y_i]^2}\right\}^2 < \sum_{i=0}^{n}\left\{\frac{g(x_i) - y_i}{[\delta y_i]^2}\right\}^2 \tag{11}$$

만약 g가 제약조건 (4.5.5)을 만족하면, 식 (11)에서 알 수 있듯이 다음 식이 성립한다.

$$\sum_{i=0}^{n}\left\{\frac{g(x_i) + \lambda_2 h(x_i) - y_i}{[\delta y_i]^2}\right\}^2 < S \tag{12}$$

각 $\lambda(\in (0, \min\{\lambda_1, \lambda_2\})$에 대해서 함수 $f(x) \doteq g(x) + \lambda h(x)$는 제약조건 (12)를 만족하고, 또한 식 (9)에서 알 수 있듯이 다음 식이 성립한다.

$$\int_{x_0}^{x_n}[f''(x)]^2dx < \int_{x_0}^{x_n}[g''(x)]^2dx \tag{13}$$

즉, 매듭들이 $x_0, x_1, \cdots, x_n$인 자연스플라인 g는 최적화문제 (4.5.4)의 해가 아니다. 이는 명제 4.5.1과 모순이다. 따라서, 매듭들이 $x_0, x_1, \cdots, x_n$인 자연스플라인 g에 대해서 식 (1)을 만족하는 상수 $p(\geq 0)$가 존재한다. ■

4.5.3 정규화문제

정규화(regularization)란 예측모형을 선택하는데 있어서 모수들의 개수를 과도하게(overfitting) 사용하는 것을 방지하는 방법이다. 정규화기법은 모형에 정보를 추가해서, 간결하고(parsimonious), 정밀한(accurate) 모형을 선택함으로써 다중공선성(multicollinearity)이나 불필요한 예측변수들(redundant predictors)을 제거한다. 정규화기법에서는 모형에 모수를 추가함으로써 발생하는 복잡성에 벌칙을 부여한다. 대표적인 정규화기법에는 Tikhonov정규화, 능형회귀(ridge regression),, LASSO(least absolute shrinkage and selection operator) 등이 있다.

제약조건이 (4.5.5)인 최적화문제 (4.5.4)의 최적해 f가 다음 식을 만족한다고 하자.

$$\sum_{i=0}^{n}\left[\frac{f(x_i)-y_i}{\delta y_i}\right]^2 < S \tag{4.5.18}$$

명제 4.5.2의 증명의 식 (11)을 유도하는 방법을 사용하면, 각 $\lambda\,(\in(0,\lambda_3))$에 대해서 함수 $f_\lambda \doteq [1-\lambda]f$가 제약조건 (4.5.5)를 만족하는 양수 λ_3가 존재함을 알 수 있다. 또한, 다음 식들이 성립함은 자명하다.

$$\int_{x_0}^{x_n}[f_\lambda''(x)]^2dx = [1-\lambda]^2\int_{x_0}^{x_n}[f''(x)]^2dx < \int_{x_0}^{x_n}[f''(x)]^2dx \tag{4.5.19}$$

따라서, 함수 f가 최적화문제 (4.5.4)의 최적해가 되기 위해서는 구간 $[x_0\,,\,x_n\,]$에서 $f''(x)\equiv 0$이어야 한다. 즉, 이 최적해 f는 다음 식을 만족하는 당연한 해(trivial solution)이다.

$$f(x) = y_0 + \frac{y_n-y_0}{x_n-x_0}[x_n-x_0] \tag{4.5.20}$$

당연한 최적해 (4.5.20)를 제외한 다른 최적해도 다음 식을 만족해야 한다.

$$\sum_{i=0}^{n}\left[\frac{f(x_i)-y_i}{\delta y_i}\right]^2 = S \tag{4.5.21}$$

제약조건이 (4.5.21)인 최적화문제 (4.5.4)는 다음과 같은 최적화문제가 된다.

$$\text{Minimize}\int_{x_0}^{x_n}\left[g''(x)\right]^2dx \text{ subject to } \sum_{i=0}^{n}\left[\frac{g(x_i)-y_i}{\delta y_i}\right]^2 = S. \tag{4.5.22}$$

이 최적화문제를 Lagrange승수법으로 풀 수 있다. 즉, 최적화문제 (4.5.22)를 다음과 같이 쓸

수 있다.

$$\text{Minimize} \left\{ \sum_{i=0}^{n} \left[\frac{g(x_i) - y_i}{\delta y_i} \right]^2 + \frac{1}{p} \int_{x_0}^{x_n} \left[g''(x) \right]^2 dx \right\}. \tag{4.5.23}$$

여기서 p는 최적화문제 (4.5.23)의 최적해 f_p에 해당하는 Lagrange승수이고, 이 최적해 f_p는 다음 조건을 만족한다.

$$\sum_{i=0}^{n} \left[\frac{f_p(x_i) - y_i}{\delta y_i} \right]^2 = S \tag{4.5.24}$$

최적화문제 (4.5.23)는 Tikhonov의 정규화문제(regularization problem)의 특수한 경우이고, p^{-1}를 정규화모수(regularization parameter)라고 한다. 또한, p를 선택하는 방법을 불일치원칙(discrepancy principle)이라 한다. 식 (4.5.23)를 유도하는 자세한 과정에 대해서는 Hanke & Scherzer [28, p. 514]를 참조하라.

명제 4.5.1에서 알 수 있듯이, 제약조건이 (4.5.21)인 최적화문제 (4.5.22)의 최적해 f는 매듭들이 $x_0, x_1, \cdots, x_n$인 자연스플라인이다. 함수 g를 다음과 같이 표기하자.

$$f(x) = \sum_{i=0}^{n-1} f_i(x) \tag{4.5.25}$$

여기서 $f_i(x)$는 다음과 같다.

$$f_i(x) = \begin{cases} a_i + b_i[x - x_i] + c_i[x - x_i]^2 + d_i[x - x_i]^3, & (x_i \le x < x_{i+1}) \\ 0, & (\text{otherwise}) \end{cases} \tag{4.5.26}$$

다음과 같이 변수들, 벡터들 그리고 행렬을 정의하자.

$$h_i \doteq x_{i+1} - x_i, \quad (i = 0, 1, \cdots, n-1) \tag{4.5.27}$$

$$\boldsymbol{a} \doteq \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \\ a_n \end{bmatrix}, \quad \boldsymbol{c} \doteq \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix}, \quad D \doteq \begin{bmatrix} \delta y_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \delta y_1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \delta y_{n-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & \delta y_n \end{bmatrix} \tag{4.5.28}$$

행렬 $T = [t_{i,j}]$는 차원이 $[n-1] \times [n-1]$인 삼중대각행렬(tridiagonal matrix)로서 다음과

같이 정의된다.

$$T \doteq \frac{1}{3}\begin{bmatrix} 2[h_0+h_1] & h_1 & 0 & \cdots & 0 & 0 & 0 \\ h_1 & 2[h_1+h_2] & h_2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & h_{n-3} & 2[h_{n-3}+h_{n-2}] & h_{n-2} \\ 0 & 0 & 0 & \cdots & 0 & h_{n-2} & 2[h_{n-2}+h_{n-1}] \end{bmatrix} \tag{4.5.29}$$

즉, 행렬 T의 0이 아닌 원소들은 다음과 같다.

$$t_{i,i} = \frac{2[h_{i-1}+h_i]}{3}, \quad t_{i,i+1} = t_{i+1,i} = \frac{h_i}{3} \tag{4.5.30}$$

행렬 T는 양정치이다. 또한, 행렬 $Q = [q_{i,j}]$ 는 차원이 $[n+1] \times [n-1]$ 인 삼중대각행렬로서 다음과 같이 정의된다.

$$Q \doteq \begin{bmatrix} \frac{1}{h_1} & 0 & 0 & \cdots & 0 & 0 & 0 \\ -\frac{1}{h_0}-\frac{1}{h_1} & \frac{1}{h_1} & 0 & \cdots & 0 & 0 & 0 \\ \frac{1}{h_1} & -\frac{1}{h_1}-\frac{1}{h_2} & \frac{1}{h_2} & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{h_{n-1}} & -\frac{1}{h_{n-1}}-\frac{1}{h_n} & \frac{1}{h_n} \\ 0 & 0 & 0 & \cdots & 0 & \frac{1}{h_n} & -\frac{1}{h_n}-\frac{1}{h_{n+1}} \\ 0 & 0 & 0 & \cdots & 0 & 0 & \frac{1}{h_{n+1}} \end{bmatrix} \tag{4.5.31}$$

즉, 행렬 Q의 0이 아닌 원소들은 다음과 같다.

$$q_{i,i} = -\frac{1}{h_{i-1}} - \frac{1}{h_i}, \quad q_{i+1,i} = \frac{1}{h_i}, \quad q_{i-1,i} = \frac{1}{h_i} \tag{4.5.32}$$

식 (4.5.26)과 식 (4.5.12)에서 알 수 있듯이, 다음 식들이 성립한다.

$$c_0 = c_n = 0, \quad d_i = \frac{c_{i+1} - c_i}{3h_i}, \ \ (i = 0,\, 1, \cdots, n-1) \tag{4.5.33}$$

식 (4.5.26)과 식 (4.5.10)에서 알 수 있듯이, 다음 식들이 성립한다.

$$b_i = \frac{a_{i+1} - a_i}{h_i} - c_i h_i - d_i h_i^2, \ \ (i = 0,\, 1, \cdots, n-1) \tag{4.5.34}$$

식 (4.5.26), 식 (4.5.11), 식 (4.5.33), 그리고 식 (4.5.34)에서 알 수 있듯이, 다음 식들이 성립한다.

$$T\boldsymbol{c} = Q^t \boldsymbol{a} \tag{4.5.35}$$

식 (4.5.26)과 식 (4.5.13)에서 알 수 있듯이, 다음 식들이 성립한다.

$$Q\boldsymbol{c} = pD^{-2}[\boldsymbol{y} - \boldsymbol{a}] \tag{4.5.36}$$

따라서 다음 식이 성립한다.

$$\boldsymbol{a} = \boldsymbol{y} - \frac{1}{p} D^2 Q \boldsymbol{c} \tag{4.5.37}$$

다음 식들이 성립한다.

$$Q^t D^2 Q \boldsymbol{c} = pQ^t[\boldsymbol{y} - \boldsymbol{a}] = pQ^t \boldsymbol{y} - pT\boldsymbol{c} \tag{4.5.38}$$

여기서 첫 번째 등호는 식 (4.5.36)에 의해서, 그리고 두 번째 등호는 식 (4.5.35)에 의해서 성립한다. 식 (4.5.38)을 다음과 같이 쓸 수 있다.

$$\left[Q^t D^2 Q + pT\right] \boldsymbol{c} = pQ^t \boldsymbol{y} \tag{4.5.39}$$

행렬 $Q^t D^2 Q + pT$는 5대각행렬(band matrix with five non-zero diagonals)임에 유의하라. 만약 p가 양수이면, $Q^t D^2 Q + pT$는 양정치행렬이다. Lagrange승수 p가 주어지면, 방정식 (4.5.39)을 풀어서 $\boldsymbol{c}$를 구한다. 이렇게 구한 $\boldsymbol{c}$를 식 (4.5.37)에 대입해서, $\boldsymbol{a}$를 구한다. 이 $\boldsymbol{c}$를 식 (4.5.33)에 대입해서, $d_0, d_1, \cdots, d_{n-1}$을 구한다. 이 값들을 식 (4.5.34)에 대입해서, $b_0, b_1, \cdots, b_{n-1}$을 구한다.

지금부터는 p를 결정하는 문제를 생각해보자. 식 (4.5.26)에서 알 수 있듯이, 식 $f(x_i) = a_i$

가 성립한다. 따라서, 다음 식들이 성립한다.

$$\sum_{i=0}^{n}\left[\frac{f(x_i)-y_i}{\delta y_i}\right]^2 = \|D^{-1}[\boldsymbol{y}-\boldsymbol{a}]\|^2 = \frac{1}{p^2}\|DQ\boldsymbol{c}\|^2 \tag{4.5.40}$$

여기서 두 번째 등호는 식 (4.5.37)에 의해서 성립한다. 식 (4.5.39)를 식 (4.5.40)에 대입하면, 다음 식을 얻는다.

$$F(p) \doteq \sqrt{\sum_{i=0}^{n}\left[\frac{f(x_i)-y_i}{\delta y_i}\right]^2} = \left\| DQ\left[Q^tD^2Q+pT\right]^{-1}Q^t\boldsymbol{y}\right\| \tag{4.5.41}$$

만약 $p \geq 0$이면, 함수 $F(p)$는 단조증가하는 볼록함수이며 또한 식 $\lim_{p\to\infty} F(p) = 0$이 성립한다. 따라서, 다음 방정식을 만족하는 양수 p는 단 하나 존재한다.

$$F(p) = \sqrt{S} \tag{4.5.42}$$

예제 4.5.1 Newton-Raphson법을 적용해서 방정식 (4.5.42)을 풀기 위해서, 다음 변수를 정의하자.

$$\boldsymbol{u} \doteq \frac{1}{p}\boldsymbol{c} = \left[Q^tD^2Q+pT\right]^{-1}Q^t\boldsymbol{y} \tag{1}$$

다음과 같이 행렬 Q^tD^2Q+pT를 Cholesky분해하자.

$$R^tR = Q^tD^2Q+pT \tag{2}$$

여기서 R은 상삼각행렬이다. 식 (1)과 식 (2)에서 알 수 있듯이, 다음 식이 성립한다.

$$R^tR\boldsymbol{u} = Q^t\boldsymbol{y} \tag{3}$$

방정식 (3)을 풀기 위해서는, 먼저 하삼각행렬 R^t의 성질을 이용해서 전향대입(forward substitution)으로 방정식 $R^t\boldsymbol{w} = Q^t\boldsymbol{y}$를 풀어서 $\boldsymbol{w}$를 구한다. 다음으로, 상삼각행렬 R의 성질을 이용해서 후향대입(backward substitution)으로 방정식 $R\boldsymbol{u} = \boldsymbol{w}$를 풀어서 $\boldsymbol{u}$를 구한다.

식 (4.5.41)과 식 (1)에서 알 수 있듯이, 다음 식이 성립한다.

$$F(p)^2 = \boldsymbol{u}^tQ^tD^2Q\boldsymbol{u} \tag{4}$$

따라서, 다음 식들이 성립한다.

$$F(p)\frac{dF}{dp} = \boldsymbol{u}^t Q^t D^2 Q \frac{d\boldsymbol{u}}{dp} = p\boldsymbol{u}^t T \left[Q^t D^2 Q + pT\right]^{-1} T\boldsymbol{u} - \boldsymbol{u}^t T\boldsymbol{u} \tag{5}$$

식 (2)와 식 (5)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{dF}{dp} = \frac{1}{F(p)} \left\{ p \left[\boldsymbol{u}^t T R^{-1}\right] \left[\boldsymbol{u}^t T R^{-1}\right]^t - \boldsymbol{u}^t T\boldsymbol{u} \right\} \tag{6}$$

식 (6)의 우변을 계산하기 위해서는 먼저 $\boldsymbol{x} \doteq \boldsymbol{u}^t T R^{-1}$를 계산한다. 즉, 후향대입법을 사용해서 방정식 $\boldsymbol{x}R = \boldsymbol{u}^t T$를 풀어서, $\boldsymbol{x}$를 구한다. 다음 Newton-Raphson식을 사용해서 방정식 (4.5.42)를 축차적으로 푼다.

$$p_{k+1} = p_k - \frac{F(p_k) - \sqrt{S}}{F'(p_k)}, \quad (k = 0, 1, \cdots) \tag{7}$$

여기서 $F'(p_k)$는 식 (6)을 이용해서 계산한다. ■

예제 4.5.2 다음과 같이 최적화문제를 살펴보자.

$$\text{Minimize} \sum_{i=0}^{n} \left[y_i - g(x_i)\right]^2 + \alpha \int_{x_0}^{x_n} \left[g''(x)\right]^2 dx. \tag{1}$$

자연스플라인공간의 기저(basis)를 $b_1(\cdot), b_2(\cdot), \cdots, b_n(\cdot)$라고 하고, 함수 $g(\cdot)$를 다음과 같이 이들의 선형결합으로 나타내자.

$$g(x) = b_1(x)\beta_1 + b_2(x)\beta_2 + \cdots + b_n(x)\beta_n \tag{2}$$

행렬 B의 (i,j) 원소 $b_{i,j}$와 행렬 Λ의 (i,j) 원소 $\lambda_{i,j}$를 다음과 같이 정의하자.

$$b_{i,j} \doteq b_j(x_i), \quad (i,j = 1, 2, \cdots, n) \tag{3}$$

$$\lambda_{i,j} \doteq \int_{x_0}^{x_n} b_i''(x)\, b_j''(x) dx, \quad (i,j = 1, 2, \cdots, n) \tag{4}$$

또한, 벡터 $\boldsymbol{y}$와 벡터 $\boldsymbol{\beta}$ 각각을 다음과 같이 정의하자.

$$\boldsymbol{y} \doteq [y_1, y_2, \cdots, y_n]^t \ , \quad \boldsymbol{\beta} \doteq [\beta_1, \beta_2, \cdots, \beta_n]^t \tag{5}$$

식 (1) 의 목적함수를 다음과 같이 쓸 수 있다.

$$F \doteq [\boldsymbol{y} - B\boldsymbol{\beta}]^t [\boldsymbol{y} - B\boldsymbol{\beta}] + \alpha \boldsymbol{\beta}^t \Lambda \boldsymbol{\beta} \tag{6}$$

목적함수 F 를 최소화하는 추정벡터 $\hat{\boldsymbol{\beta}}$ 는 다음과 같다.

$$\hat{\boldsymbol{\beta}} = [B^t B + \alpha \Lambda]^{-1} B^t \boldsymbol{y} \tag{7}$$

이에 대응하는 예측벡터 $\hat{\boldsymbol{y}}$ 는 다음과 같다.

$$\hat{\boldsymbol{y}} = B\hat{\boldsymbol{\beta}} = B[B^t B + \alpha \Lambda]^{-1} B^t \boldsymbol{y} \tag{8}$$

다음 식이 성립한다.

$$\hat{\boldsymbol{y}} = \left[I + \alpha (B^{-1})^t \Lambda B^{-1}\right]^{-1} \boldsymbol{y} \tag{9}$$

식 (7) 과 식 (9) 에서 알 수 있듯이, 이 최적화문제의 해는 능형회귀 (ridge regression) 와 관계가 있다. 즉, Tihkhonov정규화는 능형회귀추정이다. ■

4.5.4 B-스플라인회귀

B-스플라인의 정의에서 알 수 있듯이, 다음 식이 성립한다.

$$B_{i,q}(x) = 0, \quad (x \notin [x_i, x_{i+q+1})) \tag{4.5.43}$$

식 (4.2.18) 와 식 (4.5.43) 에서 알 수 있듯이, 각 q 에 대해서 q 차 스플라인들 $\{B_{j,q}(x) \,|\, j \in Z\}$ 은 서로 독립이다. 따라서, 구간 $[x_I, x_J]$ 에서 어떤 함수를 B-스플라인으로 나타내기 위해서는 서로 독립인 B-스플라인들 $B_{I-q,q}(x), B_{I-q+1,q}(x), \cdots, B_{J-2,q}(x), B_{J-1,q}(x)$ 를 기저 (basis) 로 사용한다. 즉, 구간 $[x_I, x_J]$ 에서 함수 $f(x)$ 를 다음과 같이 표기하자.

$$f(x) = \sum_{i=I-q}^{J-1} \alpha_i B_{i,q}(x) \tag{4.5.44}$$

앞에서 언급했듯이 B-스플라인들이 서로 독립이므로, 식 (4.5.44) 의 선형결합에서 계수들은 일의적으로 결정된다. 구간 $[x_I, x_J]$ 에서 마디점들이 $J - I + 1$ 개인데 반해서, 식 (4.5.44)

의 계수들은 $J-I+q$ 이다. 따라서, 회귀식 (4.5.44) 에서 설명변수들의 개수를 줄이거나 정규화시킬 필요가 있다. 이 소절에서는 정규화에 관해 다루고자 한다. 이에 대한 자세한 내용은 Eilers & Marx [20] 를 참조하라.

주어진 데이터세트 $\{(x_i, y_i) \,|\, i = 1, 2, \cdots, n\}$ 을 다음과 같은 q 차 스플라인함수들의 회귀식으로 나타내기로 하자.

$$y_i = \sum_{j=1}^{p} B_j(x_i)\, \alpha_j + \epsilon_i, \quad (i = 1, 2, \cdots, n) \tag{4.5.45}$$

여기서 $\{\epsilon_i \,|\, i = 1, 2, \cdots, n\}$ 는 서로 독립이며 평균이 0 이고 분산이 σ^2 인 확률변수들이다. 편의상, $b_{i,j} \doteq B_j(x_i)$ 로 표기하고, 다음 벡터들과 행렬을 정의하자.

$$\boldsymbol{y} \doteq \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad B \doteq \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,p} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,p} \\ \vdots & \vdots & & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,p} \end{bmatrix}, \quad \boldsymbol{\alpha} \doteq \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_p \end{bmatrix}, \quad \boldsymbol{\epsilon} \doteq \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix} \tag{4.5.46}$$

식 (4.5.45) 를 다음과 같이 쓸 수 있다.

$$\boldsymbol{y} = B\boldsymbol{\alpha} + \boldsymbol{\epsilon} \tag{4.5.47}$$

다음과 같이 오차제곱합 S 를 정의하자.

$$S \doteq \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} \left[y_i - \sum_{j=1}^{p} b_{i,j}\, \alpha_j \right]^2 \tag{4.5.48}$$

이 오차제곱합 S 를 최소화하는 $\hat{\boldsymbol{\alpha}}$ 를 최소제곱추정량이라 한다. 모수벡터 $\boldsymbol{\alpha}$ 의 최소제곱추정량 $\hat{\boldsymbol{\alpha}}$ 는 다음 정규방정식을 만족한다.

$$B^t B \hat{\boldsymbol{\alpha}} = B^t \boldsymbol{y} \tag{4.5.49}$$

만약 p 가 $J-I+1$ 보다 크면, 이 B-스플라인회귀에서는 다중공선성문제가 발생한다. 이를 해결하는 방법들 중 하나는 다음과 같이 오차제곱합에 벌칙함수 (penalty function) 를

추가해서 정규화하는 것이다.

$$S(\lambda) \doteq \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} \left[y_i - \sum_{j=1}^{p} b_{i,j}\, \alpha_j \right]^2 + \lambda \int_{x_{\min}}^{x_{\max}} \left[\sum_{j=1}^{p} B_j^{(k)}(x)\, \alpha_j \right]^2 dx \tag{4.5.50}$$

여기서 $\lambda(\geq 0)$는 적당한 상수이고, k는 적당한 자연수이다. 주어진 관찰점들의 개수가 n이므로, 이 데이터세트에 식 (4.5.50)을 적용하기 위해서 이 식을 다음과 같은 이산화 형태로 나타낸다.

$$S_\lambda \doteq \sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} \left[y_i - \sum_{j=1}^{p} b_{i,j}\, \alpha_j \right]^2 + \lambda \left[\sum_{j=k+1}^{p} \Delta^k\, \alpha_j \right]^2 \tag{4.5.51}$$

벌칙함수가 추가된 오차제곱합 S_λ를 최소화하는 벌칙함수추정량 $\hat{\boldsymbol{\alpha}}_\lambda$는 다음 식을 만족한다.

$$\left(B^t B + \lambda D_k^t D_k \right) \hat{\boldsymbol{\alpha}}_\lambda = B^t \boldsymbol{y} \tag{4.5.52}$$

여기서 D^k는 차분연산자 $\Delta^{(k)}$의 행렬표현이다. 만약 $\lambda > 0$이면, $D_k^t D_k$는 $[2k+1]$-대각행렬이다. 즉, 주대각원소들과 위로 k개 그리고 아래로 k개 대각원소들을 제외한 나머지 원소들은 0이다. 식 (4.5.51)처럼 벌칙함수가 추가된 오차제곱합을 최소화해서 결정되는 B-스플라인을 P-스플라인(penalized B-spline)이라 부른다.

앞에서는 $\{\epsilon_i \,|\, i = 1, 2, \cdots, n\}$이 서로 독립이며 평균이 0이고 분산이 σ^2인 확률변수들이라고 가정했다. 설사 이들이 서로 독립이라 하더라도, 분산이 상수라는 등분산성 가정은 현실적이 아닌 경우가 많다. 이러한 경우에는 일반화최소제곱추정법(genralized least squares method)을 사용해야 한다. 다음과 같은 표기법을 사용하자.

$$\mu_i \doteq E(y_i), \quad v_i \doteq \mathrm{Var}\left(y_i \,|\, \mu_i\right), \quad \eta_i \doteq \sum_{j=1}^{p} b_{i,j}\, \alpha_j \tag{4.5.53}$$

또한, 다음 식을 만족하는 함수 $g(\cdot)$가 존재한다고 가정하자.

$$\eta_i = g\left(\mu_i\right) \tag{4.5.54}$$

이 회귀모형의 대수우도함수(log-likelihood)를 $l(\boldsymbol{y}; \boldsymbol{\alpha})$로 표기하면, 벌칙우도함수(penalized likelihood function)는 다음과 같다.

$$L = l(\boldsymbol{y}; \boldsymbol{\alpha}) - \frac{\lambda}{2} \sum_{j=k+1}^{p} \left[\Delta^k\, \alpha_j \right]^2 \tag{4.5.55}$$

여기서 유의할 점은 식 (4.5.50)의 $S(\lambda)$를 최소화하지만, 식 (4.5.55)의 L을 최대화한다는 것이다. 만약 오차항들 $\{\epsilon_i\}$가 등분산성을 갖으면, 벌칙우도함수 L을 최소화하는 벌칙우도

추정량 $\tilde{\boldsymbol{\alpha}}$는 다음 정규방정식을 만족한다.

$$\lambda D_k^t D_k \tilde{\boldsymbol{\alpha}} = B^t[\boldsymbol{y} - \boldsymbol{\mu}] \tag{4.5.56}$$

만약 오차항들 $\{\epsilon_i\}$가 이분산성을 갖으면, 벌칙우도함수 L을 최소화하는 벌칙우도추정량 $\tilde{\boldsymbol{\alpha}}$는 다음 정규방정식을 만족한다.

$$\lambda D_k^t D_k \tilde{\boldsymbol{\alpha}} = B^t W[\boldsymbol{y} - \boldsymbol{\mu}] \tag{4.5.57}$$

여기서 $W = [w_{i,j}]$는 각 대각원소가 다음과 같은 $n \times n$ 대각행렬이다.

$$w_{i,i} \doteq \frac{1}{v_i}\left[\frac{\partial \mu_i}{\partial \eta_i}\right]^2 \tag{4.5.58}$$

식 (4.5.57)에서 알 수 있듯이, 다음 식이 성립한다.

$$\left(\lambda D_k^t D_k + B^t W B\right)\tilde{\boldsymbol{\alpha}} = B^t W[\boldsymbol{y} - \boldsymbol{\mu}] + B^t W B\tilde{\boldsymbol{\alpha}} \tag{4.5.59}$$

식 (4.5.59)을 바탕으로 다음과 같은 점화식을 고려할 수 있다.

$$\left(\lambda D_k^t D_k + B^t W^{(n)} B\right)\tilde{\boldsymbol{\alpha}}^{(n+1)} = B^t W^{(n)}[\boldsymbol{y} - \boldsymbol{\mu}^{(n)}] + B^t W^{(n)} B\,\tilde{\boldsymbol{\alpha}}^{(n)} \tag{4.5.60}$$

여기서 $W^{(n)}$과 $\boldsymbol{\mu}^{(n)}$은 $\tilde{\boldsymbol{\alpha}}^{(n)}$에서 계산된 것이다. 정규방정식 (4.5.56)의 해로 초기벡터 $\tilde{\boldsymbol{\alpha}}^{(0)}$를 사용할 수 있다.

다음과 같이 모자행렬(hat matrix) H를 정의하자.

$$H \doteq B\left(\lambda D_k^t D_k + B^t W B\right)^{-1} B^t W \tag{4.5.61}$$

또한 다음 행렬들을 정의하자.

$$Q_B \doteq B^t W B, \quad Q_\lambda \doteq \lambda D_k^t D_k \tag{4.5.62}$$

모자행렬 (hat matrix) H 는 다음 식들을 만족한다.

$$\begin{aligned}\mathrm{Tr}(H) &= \mathrm{Tr}\left((Q_B + Q_\lambda)^{-1} Q_B\right) \\ &= \mathrm{Tr}\left(Q_B^{1/2} (Q_B + Q_\lambda)^{-1} Q_B^{1/2}\right) = \mathrm{Tr}\left(\left(I + Q_B^{-1/2} Q_\lambda Q_B^{-1/2}\right)^{-1}\right) \end{aligned} \tag{4.5.63}$$

여기서 $\mathrm{Tr}(A)$ 는 정방행렬 A 의 트레이스 (trace) 이다. 다음과 같이 $p \times p$ 행렬 M 을 정의하자.

$$M \doteq \frac{1}{\lambda} Q_B^{-1/2} Q_\lambda Q_B^{-1/2} \tag{4.5.64}$$

행렬 M 의 고유값들을 $\nu_1, \nu_2, \cdots, \nu_p$ 로 표기하자. 식 (4.5.63) 을 다음과 같이 쓸 수 있다.

$$\mathrm{Tr}(H) = \mathrm{Tr}\left((I + \lambda M)^{-1}\right) = \sum_{j=1}^{p} \frac{1}{1 + \lambda \nu_j} \tag{4.5.65}$$

행렬 Q_λ 의 k 개 고유값들이 0 이므로, 행렬 M 의 k 개 고유값들도 0 이다. 따라서, 만약 λ 가 아주 커지면, $\mathrm{Tr}(H)$ 는 k 에 가까이 간다.

지금부터는 모자행렬 H 를 사용해서 λ 를 선택하는 방법에 대해서 살펴보자. 가장 대표적인 선택법은 Akaike정보통계량 (AIC) 를 이용하는 것이다. AIC는 다음과 같다.

$$\mathrm{AIC}(\lambda) = -2 \ln(\text{likelihood ratio}) + 2\mathrm{Tr}(H) \tag{4.5.66}$$

여기서 likelihood ratio 는 귀무가설 하에서 우도함수추정량을 완전모형 (full model 또는 saturated model) 의 우도함수추정량으로 나눈 것이다. 이렇게 정의된 AIC를 최소화하는 λ 를 택한다. 식 (4.5.61) 에서 정의한 모자행렬 (hat matrix) H 를 다음과 같이 쓸 수 있다.

$$\mathrm{Tr}(H) = \mathrm{Tr}\left(\left(\lambda D_k^t D_k + B^t W B\right)^{-1} B^t W B\right) \tag{4.5.67}$$

만약 $\lambda = 0$ 이면, 전혀 벌칙을 가하지 않는 것이다. 따라서, 이에 대응하는 모형을 완전모형으로 간주하고 추정한 적합값을 $\hat{y}_i$ 라 하고 이에 대응하는 잔차들의 평균제곱합을 $\hat{\sigma}_0^2$ 라고 하면, 식 (4.5.66) 을 다음과 같이 쓸 수 있다.

$$\mathrm{AIC}(\lambda) = \sum_{i=1}^{n} \frac{[y_i - \tilde{\mu}_i]^2}{\hat{\sigma}_0^2} + 2\mathrm{Tr}(H) - n \ln\left(2\pi \hat{\sigma}_0^2\right) \tag{4.5.68}$$

여기서 $\tilde{\mu}_i$ 는 모수가 $\lambda (\neq 0)$ 인 경우에 μ_i 를 추정한 값이다. 이 평균제곱합 $\hat{\sigma}_0^2$ 는 고정된

값이므로, AIC(λ) 대신 다음과 같이 변형된 AIC*(λ)를 사용해도 마찬가지이다.

$$\text{AIC}^*(\lambda) = \sum_{i=1}^{n} \frac{[y_i - \tilde{\mu}_i]^2}{\hat{\sigma}_0^2} + 2\text{Tr}(H) \tag{4.5.69}$$

다른 선택법으로는 교차검증법(cross-validation method)과 일반화교차검증법(generalized cross-validation method)이 있다. 교차검증통계량 CV는 다음과 같다.

$$\text{CV}(\lambda) = \sum_{i=1}^{n} \left[\frac{y_i - \tilde{y}_i}{1 - h_{ii}} \right]^2 \tag{4.5.70}$$

여기서 h_{ii}는 행렬 H의 대각원소이다. 또한, 일반화교차검증통계량 GCV는 다음과 같다.

$$\text{GCV}(\lambda) = \sum_{i=1}^{n} \left[\frac{y_i - \tilde{y}_i}{n - \sum_{i=1}^{n} h_{ii}} \right]^2 \tag{4.5.71}$$

교차검증통계량 CV나 일반화교차검증통계량 GCV가 최소가 되는 λ를 선택한다.

예제 4.5.3 B-스플라인을 사용한 회귀분석을 하는 예를 살펴보기 위해서, 다음 MATLAB 프로그램 `BSplineRegression102.m`을 실행하라.

```
% --------------------------------------------------------------------------
% Filename: BSplineRegression102.m
% B-Spline Regression
% Programmed by CBS
% --------------------------------------------------------------------------
clear all, close all, format short
% Inputs
x = [  1 2 3.5 4 5.5 6 ]';
rng(1299827)
y = 2.0*x + 0.1*randn(size(x))
% Data Structure
xLeft = 0;
xRight = 6;
ndx = 6;
bdeg = 3;
% Make B-spline Matrix
xx = [ xLeft; x ];
dxx = diff(xx);
dxmean = mean(dxx)
t = zeros(1,2*ndx+bdeg-2);
t(1:bdeg+1) = xLeft + dxmean*[-bdeg:0];
t(bdeg+2:bdeg+ndx) = 1:ndx-1;    % Input
t(bdeg+ndx+1:bdeg+2*ndx-2) = t(bdeg+ndx)+dxmean*(1:(ndx-2));
T = (0*x+1)*t;
X = x*(0*t+1);
[lengthx,lengtht] =size(T)
dx = dxx*ones(1,lengtht)
Bmat = zeros(lengthx,lengtht,bdeg+1);
```

```
Bmat(:,:,1) = (T <= X)&(X < (T+dx) )
for kk = 2:bdeg+1       % bdeg
    for jj=1:lengtht
       for ii=1:lengthx
           if jj+kk-1 > lengtht
              Bmat(ii,jj,kk) = 0;
           elseif jj+kk > lengtht
              Bmat(ii,jj,kk) = (x(ii)-t(jj))/(t(jj+kk-1)-t(jj)) ...
                               *Bmat(ii,jj,kk-1);
           else
              Bmat(ii,jj,kk) = (x(ii)-t(jj))/(t(jj+kk-1)-t(jj)) ...
                         *Bmat(ii,jj,kk-1) ...
                        + (t(jj+kk)-x(ii))/(t(jj+kk)-t(jj+1)) ...
                         *Bmat(ii,jj+1,kk-1);
           end
        end % ii
    end % jj
end
Bmat
% Regression
B = Bmat(:,:,end)
% Regression
lambda = 0.5
[ m, n] = size(B)
D = diff(eye(n))
beta = (B'*B+lambda*D'*D)\(B'*y)
yhat = B*beta
plot(x,y,'b:*',x,yhat,'k--o','LineWidth',1.5)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x'), ylabel('\bf y','rotation',0)
legend('Given Data','Fitted Data','location','NW')
saveas(gcf,'BSplineRegression102.jpg')
Q = inv(B'*B + lambda*D'*D)
SS = sum((y-yhat).^2)
tra = sum(diag(Q*(B'*B)))
gcv = SS/(m-tra)^2
save('BSplineRegression102.txt','x','t','B','beta','yhat','-ascii')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB 프로그램 `BSplineRegression102.m`에서는 설명변수 x의 관찰값들이 $\{1, 2, 3.5, 4, 5.5, 6\}$이고 종속변수 y의 값들은 다음과 같다.

$$y_i = 2.0x_i + \epsilon_i \tag{1}$$

이 관찰값들 $\{(x_i, y_i) \,|\, i = 1, 2, \cdots, 6\}$를 3차B-스플라인으로 추정한 결과가 그림 4.5.1에 수록되어 있다. 이 그림에서 알 수 있듯이, 중심에서는 B-스플라인회귀식이 잘 적합되나 양쪽 끝으로 갈수록 관찰값과 적합값의 차이가 커진다. 일반화최소제곱추정법을 사용해서 이러한 현상을 완화시킬 수 있다. ■

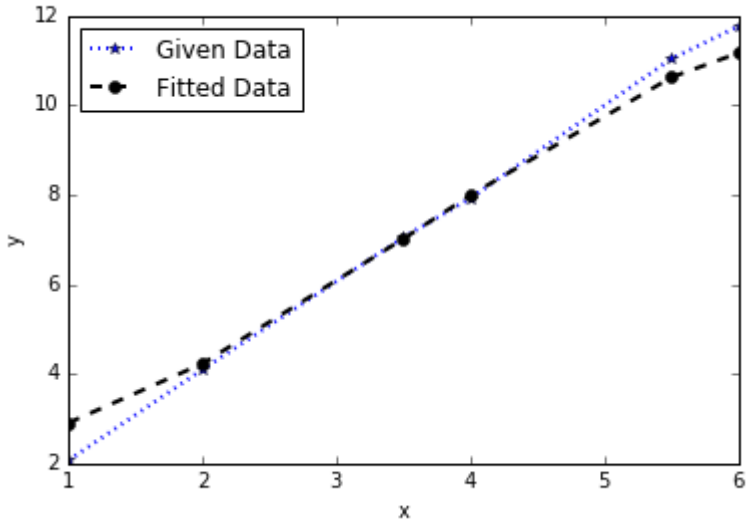


그림 4.5.1. B-스플라인회귀식

예제 4.5.4 Python을 사용해서 예제 4.5.3을 다시 다루기 위해서, 다음 Python프로그램 BSplineRegression102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
# from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
import random

# Data Structure
x = np.array([ 1, 2, 3.5, 4, 5.5, 6 ]);
random.seed(1299827)
y = 2.0*x + 0.1*np.random.randn(len(x));
xLeft = 0;
xRight = 6;
ndx = 6;
bdeg = 3;

# Make B-spline Matrix
xx = np.hstack((xLeft, x));
dxx = np.diff(xx);
dxmean = np.mean(dxx)
t = np.zeros((1,2*ndx+bdeg-2));
t101 = xLeft + dxmean*np.arange(-bdeg,1,1);
t102 = 1.0*np.arange(1,ndx,1);
t103 = t102[-1] + dxmean*np.arange(1,ndx-1,1);
t = np.hstack((t101,t102,t103));
x.shape = (len(x),1)
t.shape = (1,len(t))
T = np.matmul(0.0*x+1.0,t);
X = np.matmul(x,0.0*t+1.0);
[lengthx,lengtht] = T.shape
```

```
dxx.shape = (len(dxx),1)
dx = np.matmul(dxx,np.ones((1,lengtht)));
Bmat = np.zeros((lengthx,lengtht,bdeg+1));
Bmat[:,:,0] = (T <= X)&(X < (T+dx) );
for k1 in range(1,bdeg+1):
    for j1 in range(0,lengtht):
       for i1 in range(0,lengthx):
           print(j1,i1)
           if j1+k1+1 > lengtht:
              Bmat[i1,j1,k1] = 0;
           elif j1+k1+2 > lengtht:
              Bmat[i1,j1,k1] = (x[i1]-t[0,j1])/(t[0,j1+k1]-t[0,j1])*Bmat[i1,
    j1,k1-1];
           else:
              dumB1 = (x[i1]-t[0,j1])/(t[0,j1+k1]-t[0,j1])*Bmat[i1,j1,k1-1];
              dumB2 = (t[0,j1+k1+1]-x[i1])/(t[0,j1+k1+1]-t[0,j1+1])*Bmat[i1,
    j1+1,k1-1];
              Bmat[i1,j1,k1] = dumB1 + dumB2;
print(Bmat)

# Regression
B = Bmat[:,:,-1]
lambdaa = 0.5
[ m, n] = B.shape
D = np.transpose(np.diff(np.identity(n)))
XX = np.matmul(np.transpose(B),B) + lambdaa*np.matmul(np.transpose(D),D)
y.shape = (len(y),1)
Xy = np.matmul(np.transpose(B),y)
betaa = np.matmul(np.linalg.inv(XX),Xy)
print(betaa)
yhat = np.matmul(B,betaa)
print(yhat)
Q = np.linalg.inv(XX)
print(Q)
SS = np.sum((y-yhat)**2)
print(SS)
tracee = np.matrix.trace(np.matmul(Q,np.matmul(np.transpose(B),B)))
print(tracee)
gcv = SS/(m-tracee)**2
print(gcv)

# Plotting
fig = plt.figure()
plt.plot(x,y,'b:*',lw=2, label='Given Data')
plt.plot(x,yhat,'k--o',lw=2, label='Fitted Data')
plt.xlabel('x'); plt.ylabel('y')
plt.legend(loc='upper left', numpoints=1)
plt.axis([ 1, 6, 2, 12 ])
plt.show()
fig.savefig('BSplineRegression102.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 4.5.3의 결과와 같다. ■

제4.6절 예제들

예제 4.6.1 분할 $\Pi = [0, 0.25, 0.75, 1]$에서 함수 $f(x) = 3x^2 + 2x + 1$의 스플라인을 구하기 위해서, 다음 MATLAB프로그램 CScalculus101DY.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: CScalculus101DY.m
%  Cubic-Spline approximation, Integration, Differentiation
%  Programmed by CDY
%------------------------------------------------------------------------
clear all, close all, clc
format long g
% Function itself and Differentiation
f = @(x) (3*x.^2 + 2*x +1)
x = [ 0 ; 0.25 ; 0.75 ; 1 ]
y = f(x)
[ x xIndex ] = sort(x)
y = y(xIndex);              % sorting
PP = spline(x,f(x));
[breaks, coefs , l , k , d ] = unmkpp(PP)
NoInterp = 9;
xNew = linspace(0,1,NoInterp);
[ xNew xIndex ] = sort(xNew);            % sorting
for jj=1:length(xNew)
    for ii = 1:length(x)
        if xNew(jj) < x(ii)
            NoInt(jj) = ii-1;
            break
        end % if-end
        if xNew(end) == x(end)
            NoInt(NoInterp) = length(x)-1;
        end % if-end
    end % for-ii-end
end  % for-jj-end
for jj = 1:length(xNew)
    ii = NoInt(jj);
    xdel = xNew(jj) - x(ii);
    fspline(jj) = coefs(ii,1)*xdel^3 + coefs(ii,2)*xdel^2 ...
                  + coefs(ii,3)*xdel + coefs(ii,4);
    g(jj) = 3*coefs(ii,1)*xdel^2 + 2*coefs(ii,2)*xdel + coefs(ii,3);
end
% % Subplot True function and Spline
subplot(1,3,1)
fTrue = f(xNew);
plot(xNew,fspline,'ro',xNew,fTrue,'g-','LineWidth',1.5);
legend('Spline f','True f','location','NW')
set(gca,'fontsize',11,'fontweigh','bold','Xtick',[ 0 0.25 0.75 1])
title('\bf Spline Itself')
% Differentiation
gTrue = 6*xNew+2;
% % Subplot Spline differentiation
subplot(1,3,2)
plot(xNew,g,'ro',xNew,gTrue,'g-','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','Xtick',[ 0 0.25 0.75 1])
legend('Spline g','True g','location','NW')
title('\bf Differentiation')
% It should be exactly the same. (Because a polynomial of order 2)
% Integration
for jj=1:length(xNew)
```

```
    for ii = 1:length(x)
        if xNew(jj) < x(ii)
            NoInt(jj) = ii-1;
            break
        end % if-end
        if xNew(end) == x(end)
            NoInt(NoInterp) = length(x)-1;
        end % if-end
    end % for-ii-end
end  % for-jj-end
hApprox = zeros(1,length(x)+1);
for ii = 1:length(x)-1
    xdel = x(ii+1) - x(ii);
    hdel(ii+1) = 1/4*coefs(ii,1)*xdel^4 + 1/3*coefs(ii,2)*xdel^3 ...
               + 1/2*coefs(ii,3)*xdel^2 + coefs(ii,4)*xdel;
end
hApprox = cumsum(hdel);
hdelExtra = zeros(1,length(xNew));
hRefine = zeros(1,length(xNew));
for jj = 1:length(xNew)
    ii = NoInt(jj);
    xdel = xNew(jj) - x(ii);
    hdelExtra = 1/4*coefs(ii,1)*xdel^4 + 1/3*coefs(ii,2)*xdel^3 ...
               + 1/2*coefs(ii,3)*xdel^2 + coefs(ii,4)*xdel;
    hRefine(jj) = hApprox(ii) + hdelExtra;
end
hRefine
% Subplot Integral
subplot(1,3,3)
hTrue = xNew.*(1+ xNew.*(1+1*xNew));
plot(xNew,hRefine,'ro',xNew,hTrue,'g-','LineWidth',1.5);
legend('Spline h','True h','location','NW')
set(gca,'fontsize',11,'fontweigh','bold','Xtick',[ 0 0.25 0.75 1])
title('\bf Integration')
axis([ 0 1 0 3 ])
% It should be exactly the same. (Because a polynomial of ordeer 2)
% Plot
saveas(gcf,'CScalculus101DY.jpg')
% Save the Dataset
save('CScalculus101DY')
% End of Program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 함수 $f(x)$의 3차스플라인 $S(x)$가 다음과 같음을 알 수 있다.

$$S(x) = \begin{cases} 1 + 2x + 3x^2, & (0 \le x < 0.25) \\ 1.6875 + 3.5[x - 0.25] + 3[x - 0.25]^2, & (0.25 \le x < 0.75) \\ 4.1875 + 6.5[x - 0.75] + 3[x - 0.75]^2, & (0.75 \le x \le 1) \end{cases} \tag{1}$$

이 3차스플라인 $S(x)$의 미분은 다음과 같다.

$$S'(x) = \begin{cases} 2 + 6x, & (0 \le x < 0.25) \\ 3.5 + 6[x - 0.25], & (0.25 \le x < 0.75) \\ 6.5 + 6[x - 0.75], & (0.75 \le x \le 1) \end{cases} \tag{2}$$

이 3차스플라인 $S(x)$의 적분은 다음과 같다.

$$\begin{aligned} &\int_0^x S(t)dt \\ &= \begin{cases} x + x^2 + x^3, & (0 \le x < 0.25) \\ 0.328125 + 1.6875[x - 0.25] + 1.75[x - 0.25]^2 + [x - 0.25]^3, & (0.25 \le x < 0.75) \\ 1.734375 + 4.1875[x - 0.75] + 3.25[x - 0.75]^2 + [x - 0.75]^3, & (0.75 \le x \le 1) \end{cases} \end{aligned} \tag{3}$$

그림 4.6.1에 이 스플라인, 미분함수와 적분함수가 수록되어 있다. ■

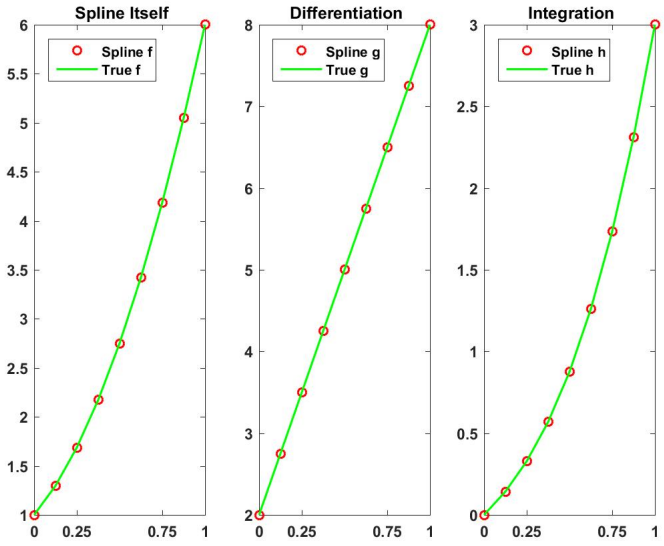


그림 4.6.1. 스플라인, 미분과 적분

예제 4.6.2 다음 함수를 살펴보자.

$$\phi(x) = \frac{1}{1 + \exp(200x)} = 1 - \frac{\exp(200x)}{1 + \exp(200x)}, \quad (-0.05 < x < 0.05) \tag{1}$$

구간 $[-1,\ -0.05]$에서 다음 함수를 정의하자.

$$p(x) = \left[1 + 2x\exp\left(-[x+1]^2\right)\right][x+0.05]^2 + \phi'(-0.05)[x+0.05] + \phi(-0.05) \tag{2}$$

여기서 $\phi(-0.05)$와 $\phi'(-0.05)$는 다음과 같다.

$$\phi(-0.05) = \frac{1}{1+e^{-10}}, \quad \phi'(-0.05) = -\frac{200e^{-10}}{[e^{-10}+1]^2} \tag{3}$$

구간 $[0.05,\ 1]$에서 다음 함수를 정의하자.

$$q(x) = \frac{1}{2}[x-0.05]^2\sin\left(32\pi[x-0.05]^2\right) + \phi'(0.05)[x-0.05] + \phi(0.05) \tag{4}$$

여기서 $\phi(0.05)$와 $\phi'(0.05)$는 다음과 같다.

$$\phi(0.05) = \frac{1}{1+e^{10}}, \quad \phi'(0.05) = -\frac{200e^{10}}{[e^{10}+1]^2} \tag{4}$$

다음 함수를 정의하자.

$$f(x) = \begin{cases} p(x), & (-1 \le x \le -0.05) \\ \phi(x), & (-0.05 < x < 0.05) \\ q(x), & (0.05 \le x \le 1) \end{cases} \tag{5}$$

함수 $f(x)$를 그리기 위해서, 다음 MATLAB프로그램 DifferentialExtension101DY.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: DifferentialExtension101DY.m
%  Extension of Continuously Differentiable Function
%  Programmed by CDY
%------------------------------------------------------------------------
clear all, close all, clc
format long g
% Original Function
xx = -0.2:0.001:0.2;
phii = 1./(1+exp(200*xx));
subplot(2,1,1)
plot(xx,phii,'k','Linewidth',1.5)
set(gca,'fontsize',11,'fontweigh','bold','ylim',[-0.8,1.2])
title('\bfFunction \phi (x)')
hold on, plot([-0.2 0.2],[0 0]), plot([0 0], [-1.2 1.2])
hold off
% Extension Function
```

```
syms X
phii = 1/(1+exp(200*X)), dphii = diff(phii,X)
xp = -0.05, xt = 0.05
phiixp = subs(phii,X,xp), phiixpX = double(phiixp)
phiixt = subs(phii,X,xt), phiixtX = double(phiixt)
dphiixp = subs(dphii,X,xp), dphxpX = double(dphiixp)
dphiixt = subs(dphii,X,xt), dphxtX = double(dphiixt)
phii = @(x) 1./(1+exp(200*x))
p = @(x) (1+2*x.*exp(-(x+1).^2)).*(x-xp).^2 + dphiixp*(x-xp) + phiixp
q = @(x) phiixt + dphiixt*(x-xt) + 1/2*(x-xt).^2.*sin(32*pi*(x-xt).^2)
xLow = -1:0.001:-0.05;
xCenter = -0.05:0.0005:0.05;
xHigh = 0.05:0.001:1;
xTotal = [xLow xCenter xHigh];
f = [ p(xLow) phii(xCenter) q(xHigh) ];
subplot(2,1,2)
plot(xTotal,f,'r-','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','Xtick',[-1:0.2:1], ...
    'ylim',[-0.8,1.2])
title('\bf Function f(x)')
saveas(gcf,'DifferentialExtension101DY.jpg')
save('DifferentialExtension101DY')
% End of Program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 그림 4.6.2에 함수 $\phi(x)$와 함수 $f(x)$가 그려진다.

■

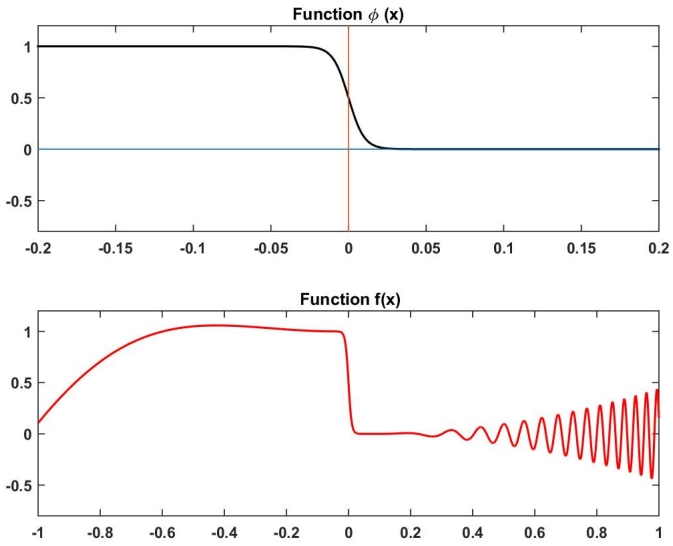


그림 4.6.2. 스플라인, 미분과 적분

예제 4.6.3 예제 4.6.2의 함수 $f(x)$에 대해서 다시 생각해보자. 각 n $\left(=2^3, 2^4, 2^5, 2^6, 2^7, 2^8\right)$에 대해서 구간 $[-1,1]$의 분할 $\Pi_n = \{-1 = x_0 < x_1 < \cdots < x_{n-1} < x_n = 1\}$을 살펴보자.

분할 Π_n 에서 함수 $f(x)$ 의 3차스플라인을 $S_n(x)$ 로 표기하자. 또한 최대절대오차(maximum absolute error)를 다음과 같이 표기하자.

$$\mathrm{MAE}_f(n) \doteq \max_{-1\le x\le 1} |S_n(x) - f(x)| \tag{1}$$

또한, 도함수 $g(x) = f'(x)$ 와 스플라인도함수 $S_n'(x)$ 의 최대절대오차와 정적분함수 $h(x) = \int_{-1}^{x} f(t)dt$ 와 스플라인정적분함수 $\int_{-1}^{x} S_n(t)dt$ 의 최대절대오차를 다음과 같이 표기하자.

$$\mathrm{MAE}_{D_f}(n) \doteq \max_{-1\le x\le 1} \left|S_n'(x) - g(x)\right| \tag{2}$$

$$\mathrm{MAE}_{I_f}(n) \doteq \max_{-1\le x\le 1} \left|\int_{-1}^{x} S_n(t)dt - h(x)\right| \tag{3}$$

이 함수들과 최대절대오차들을 그리기 위해서, 다음 MATLAB프로그램 DifferentialExtension102DY.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: DifferentialExtension102DY.m
%  Extension of Continuously Differentiable Function
%  Programmed by CDY
%------------------------------------------------------------------------
clear all, close all, clc
format long g

% % PART 1: Calculate True Function Values, Derivatives, and Integration
% Input parameters
xp = -0.05, xt = 0.05
p0 = 1/(exp(-10) + 1)
p1 = -(200*exp(-10))/(exp(-10) + 1)^2
q0 = 1/(exp(10) + 1)
q1 =  -(200*exp(10))/(exp(10) + 1)^2
% find the true functions, derivatives, integrations
syms X
varphi = 1/(1+exp(200*X))
dvarphi = diff(varphi,X)
Ivarphi = int(varphi,X)
p = (1+2*X*exp(-(X+1)^2))*(X-xp)^2 + p1*(X-xp) + p0
dp = diff(p,X), dp = simplify(dp)
Ip = int(p,X), Ip = simplify(Ip)
q = q0 + q1*(X-xt) + 1/2*(X-xt)^2*sin(32*pi*(X-xt)^2)
dq = diff(q,X), dq = simplify(dq)
Iq = int(q,X), Iq = simplify(Iq)
   %  Given points
nnn = 2^11  % nnn = 2048
rho1 = (1+xp)/2, rho2 = (xt-xp)/2, rho3 = 1 - rho1-rho2
m1 = round(rho1*(nnn+1)), m2 = round(rho2*(nnn+1)), m3 = nnn+1-m1-m2
   % x-points
xxp = linspace(-1,xp,m1);
xxvarphiDum = linspace(xp,xt,m2+2); xxvarphi = xxvarphiDum(2:end-1);
xxq = linspace(xt,1,m3);
xxx = [ xxp xxvarphi xxq ];                    % True x-points
xxx(1) = xxp(1); xxx(end) = xxq(end);
```

```
   % f
ffp = double(subs(p,X,xxp));
ffvarphi = double(subs(varphi,X,xxvarphi));
ffq = double(subs(q,X,xxq));
fff = [ ffp ffvarphi ffq ];                    % True f(x)
   % Derivatives
dfp = double(subs(dp,X,xxp));
dfvarphi = double(subs(dvarphi,X,xxvarphi));
dfq = double(subs(dq,X,xxq));
dff = [ dfp dfvarphi dfq ];                    % True f'(x)
   % Integration
Ifp = double(subs(Ip,X,xxp)) - double(subs(Ip,X,xxp(1)));
Ifvarphi = double(subs(Ivarphi,X,xxvarphi)) ...
           - double(subs(Ivarphi,X,xxp(end))) ...
           + Ifp(end) ...
           + 1/2*(xxvarphi(1)-xxp(end))*(ffvarphi(1)+ffp(end));
Ifq =  double(subs(Iq,X,xxq)) - double(subs(Iq,X,xxvarphi(end))) ...
       + Ifvarphi(end) ...
       + 1/2*(xxq(1)-xxvarphi(end))*(ffq(1)+ffvarphi(end));
Iff = [ Ifp Ifvarphi Ifq ];                    % Integral(x)
% Save DAta
save('SplineTrueValues','xxx','fff','dff','Iff', 'nnn')
% %%%%% End of Part I

% %% PART II: Spline, Derivative, and Integration
log2nMax = 8;               % n = 2^(log2nMax)
Delta = zeros(1,log2nMax);
for ii = 3:log2nMax
    nn = 2^ii;
    Delta(ii) = 1/nn;
%    disp('No of Given Points'), disp(nn+1)
    clear xgiven fgiven
    dumIndex = 1:2^(log2nMax-ii):length(xxx);
    xgiven = xxx(dumIndex);
    fgiven = fff(dumIndex);
   % Which interval a new x does belong?
    xInterpol = xxx;
    for jj=1:length(xInterpol)
        for i1 = 1:length(xgiven)
            if xInterpol(jj) < xgiven(i1)
               NoInt(jj) = i1-1;
               break
            end % if-end
            if xInterpol(end) == xgiven(end)
               NoInt(length(xInterpol)) = length(xgiven)-1;
            end % if-end
        end % for-ii-end
    end  % for-jj-end
   %  disp('Which Interval?'), disp(NoInt)
  % Finding Spline
    PP = spline(xgiven,fgiven);
    [breaks, coefs , l , k , d ] = unmkpp(PP);
  % Differentiation
   for j2 = 1:length(xInterpol)
       i2 = NoInt(j2);
       xdel = xInterpol(j2) - xgiven(i2);
       fSpline(j2) = coefs(i2,1)*xdel^3 + coefs(i2,2)*xdel^2 + coefs(i2,3)*
    xdel + coefs(i2,4);
       g(j2) = 3*coefs(i2,1)*xdel^2 + 2*coefs(i2,2)*xdel + coefs(i2,3);
   end
   % Maximum Absolute Difference Errors
```

```
    fAbsErr(ii) = max(abs(fff-fSpline));
    gAbsErr(ii) = max(abs(dff-g));
    % End of Differentiation Program
  % Integration
      % Approximation Part
    hApprox = zeros(1,length(xgiven)+1);
    for i3 = 1:length(xgiven)-1
        xdel = xgiven(i3+1) - xgiven(i3);
        hdel(i3+1) = 1/4*coefs(i3,1)*xdel^4 + 1/3*coefs(i3,2)*xdel^3 ...
                     + 1/2*coefs(i3,3)*xdel^2 + coefs(i3,4)*xdel;
    end
    hApprox = cumsum(hdel);
      % Refinement Part
    hdelExtra = zeros(1,length(xInterpol));
    hRefine = zeros(1,length(xInterpol));
    for j4 = 1:length(xInterpol)
        i4 = NoInt(j4);
        xdel = xInterpol(j4) - xgiven(i4);
        hdelExtra = 1/4*coefs(i4,1)*xdel^4 + 1/3*coefs(i4,2)*xdel^3 ...
               + 1/2*coefs(i4,3)*xdel^2 + coefs(i4,4)*xdel;
        hRefine(j4) = hApprox(i4) + hdelExtra;
    end
    % Maximum Absolute Difference Error
    IAbsErr(ii) = max(abs(Iff-hRefine));
    % End of Integration
end % for-ii-end
save('DifferentialExtension102DY')

% Plotting Spline and Maximum of abstract Errors
subplot(2,1,1)
  plot(xxx,fff,'g-',xInterpol,fSpline,'k--','LineWidth',1.5);
  legend('Function','Spline','location','SW')
  set(gca,'fontsize',11,'fontweigh','bold','xlim',[-1 1],'ylim',[-0.8 1.2])
  title('\bf Cubic Spline Function')
subplot(2,1,2)
  disp('Max Abs Error of Splines')
  disp(fAbsErr(3:end))
  loglog(Delta(3:end),fAbsErr(3:end),'k-o','LineWidth',1.5)
  set(gca,'fontsize',11,'fontweight','bold','xdir','reverse', ...
     'xlim',[2^-8 2^-3])
  set(gca,'xtick',[2^-8 2^-7 2^-6 2^-5 2^-4 2^-3 ])
  title('\bf Maximum Error of Splines')
  saveas(gcf,'DifferentialExtension102aDY.jpg')
% Plotting Derivative and Maximum of abstract Errors
figure
subplot(2,1,1)
  plot(xxx,dff,'g-',xInterpol,g,'k--','LineWidth',1.5);
  legend('Derivative','Spline','location','SW')
  set(gca,'fontsize',11,'fontweigh','bold','xlim',[-1 1])
  title('\bf Cubic Spline Derivative')
subplot(2,1,2)
  disp('Max Abs Error of Derivatives')
  disp(gAbsErr(3:end))
  loglog(Delta(3:end),gAbsErr(3:end),'k-o','LineWidth',1.5)
  set(gca,'fontsize',11,'fontweigh','bold','xdir','reverse', ...
     'xlim',[2^-8 2^-3])
  set(gca,'xtick',[2^-8 2^-7 2^-6 2^-5 2^-4 2^-3 ])
  title('\bf Maximum Error of Derivatives')
  saveas(gcf,'DifferentialExtension102bDY.jpg')
% Plotting Integral and Maximum of abstract Errors
figure
```

```
subplot(2,1,1)
  plot(xxx,Iff,'g-',xInterpol,hRefine,'k--','LineWidth',1.5);
  legend('Integral','Spline','location','SE')
  set(gca,'fontsize',11,'fontweigh','bold','xlim',[-1 1])
  title('\bf Cubic Spline Integral')
subplot(2,1,2)
  disp('Max Abs Error of Integrals')
  disp(IAbsErr(3:end))
  loglog(Delta(3:end),IAbsErr(3:end),'k-o','LineWidth',1.5)
  set(gca,'fontsize',11,'fontweigh','bold','xdir','reverse', ...
      'xlim',[2^-8 2^-3])
  set(gca,'xtick',[2^-8 2^-7 2^-6 2^-5 2^-4 2^-3 ])
  title('\bf Maximum Error of Integrals')
  saveas(gcf,'DifferentialExtension102cDY.jpg')
% End of Program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 함수 $f(x)$와 3차스플라인 $S_n(x)$가 그림 4.6.3의 위 그래프에 그려지고 아래 그래프에는 이들의 최대절대오차 $\mathrm{MAE}_f(n)$이 그려졌다. 이 그림에서 알 수 있듯이, 마디점들의 개수 n이 증가함에 따라 최대절대오차가 감소한다. 함수 $f(x)$와 3차스플라인 $S_n(x)$의 미분함수들이 그림 4.6.4의 위 그래프에 그려지고, 아래 그래프에는 이들의 최대절대오차 $\mathrm{MAE}_{D_f}(n)$가 그려졌다. 이 그림에서 알 수 있듯이, 마디점들의 개수 n이 증가함에 따라 최대절대오차가 감소한다. 함수 $f(x)$와 3차스플라인 $S_n(x)$의 적분함수들이 그림 4.6.5의 위 그래프에 그려지고 아래 그래프에는 이들의 최대절대오차 $\mathrm{MAE}_{I_f}(n)$가 그려졌다. 이 그림에서 알 수 있듯이, 마디점들의 개수 n이 증가함에 따라 최대절대오차는 증가하지 않는다. 특히, n이 2^5 이상이면 최대절대오차는 감소하지 않는다. ■

예제 4.6.4 예제 4.6.3에 대해서 다시 생각해보자. 여기서는 분할의 마디점들이 $n+1 = 2^8+1$인 경우를 살펴보자. 소구간 $[-1,-0.05]$에서 등간격인 m_1개 마디점들을, 소구간 $(-0.05,0.05)$에서 등간격인 m_2개 마디점들을, 그리고 소구간 $[0.05,1]$에서 등간격인 m_3개 마디점들을 선택하기로 하자. 여기서 m_1, m_2와 m_3는 다음 식들을 만족한다고 하자.

$$m_1 \approx [n+1]\rho_1, \quad m_2 \approx [n+1]\rho_2, \quad m_3 \approx [n+1]\rho_3 \tag{1}$$

따라서 다음 식들이 성립한다.

$$\rho_1 > 0, \quad \rho_2 > 0, \quad \rho_3 > 0, \quad \rho_1+\rho_2+\rho_3 = 1 \tag{2}$$

상수들 ρ_1, ρ_2, ρ_3이 어떻게 변화함에 따라 최대절대오차가 어떻게 변하는지를 살펴보기

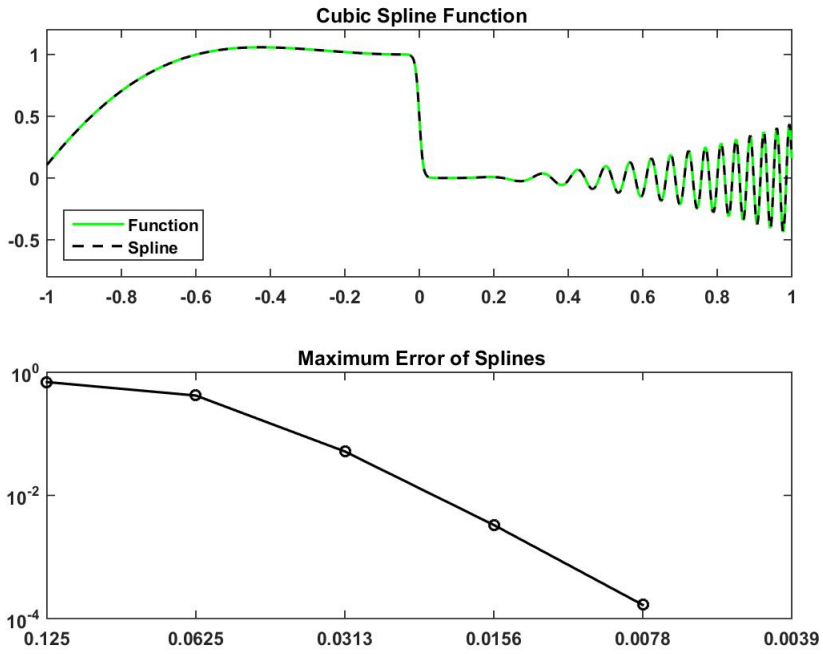


그림 4.6.3. 스플라인함수와 최대절대오차

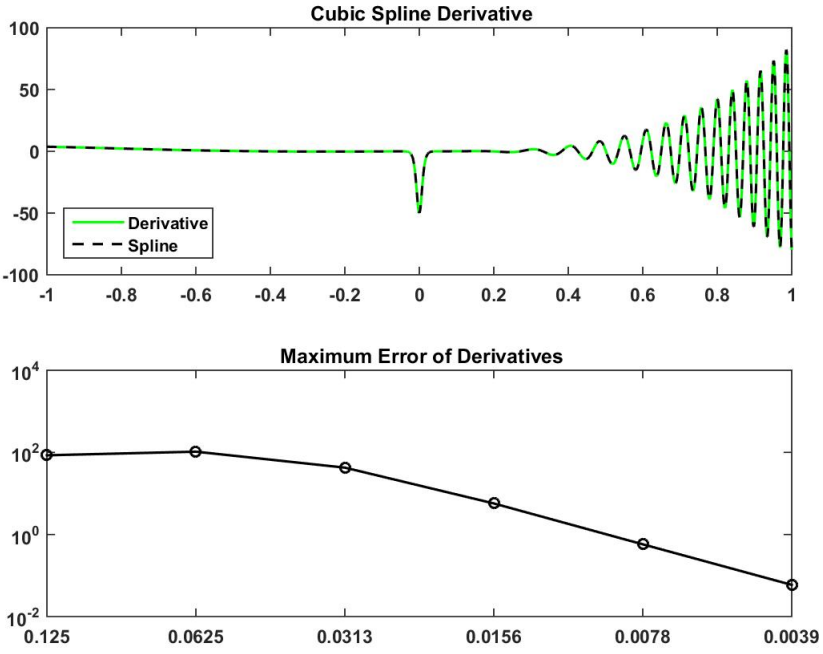


그림 4.6.4. 스플라인미분과 최대절대오차

위해서, 다음 MATLAB프로그램 DifferentialExtension103DY.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: DifferentialExtension103DY.m
%  Extension of Continuously Differentiable Function
```

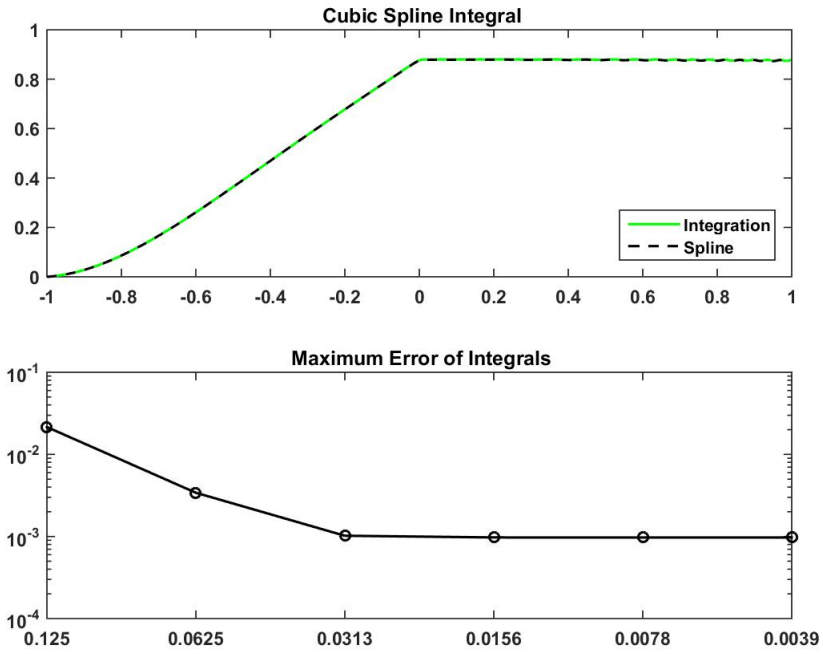


그림 4.6.5. 스플라인적분과 최대절대오차

```
%  Cubic-Spline approximation
%  The ratio among the intervals vs. Minimum Absolute Error
%  True Values for n = 2^11
%  Programmed by CDY
%-------------------------------------------------------------------------
clear all, close all, clc, format long g
% Calculate the function values
xp = -0.05, xt = 0.05
p0 = 1/(exp(-10) + 1)
p1 = -(200*exp(-10))/(exp(-10) + 1)^2
q0 = 1/(exp(10) + 1)
q1 =  -(200*exp(10))/(exp(10) + 1)^2
syms X
varphi = 1/(1+exp(200*X))
p = (1+2*X*exp(-(X+1)^2))*(X-xp)^2 + p1*(X-xp) + p0
q = q0 + q1*(X-xt) + 1/2*(X-xt)^2*sin(32*pi*(X-xt)^2)
% load true data with n +1 = 2^11 +1 = 2048 + 1
Sdata = dlmread('SplineTrueValues.dat');
xxx = Sdata(1,:);
fff = Sdata(2,:);
dff = Sdata(3,:);
Iff = Sdata(4,:);
log2nMax = 8;   % n = 2^(log2nMax)
Delta = zeros(1,log2nMax);

for ii = log2nMax:log2nMax    % log2nMax = 8
    for irho1 = 1:19
        for irho3 = 1:20
            rho1 = irho1*0.025;
            rho3 = irho3*0.025;
            rho2 = 1 - rho1 - rho3;

    nn = 2^ii;
```

```
    Delta(ii) = 1/nn;
  %  disp('No of Given Points'), disp(nn+1)
    clear xold fold
    dumIndex = 1:2^(log2nMax-ii):length(xxx);

   %  Given Point Ratios
m1 = ceil(rho1*(nn+1)); m2 = floor(rho2*(nn+1)); m3 = nn+1-m1-m2;
   % x-points
xoldp = linspace(-1,xp,m1);
xoldvarphiDum = linspace(xp,xt,m2+2); xoldvarphi = xoldvarphiDum(2:end-1);
xoldq = linspace(xt,1,m3);
xold = [ xoldp xoldvarphi xoldq ];                    % True x-points
xold(end) = 1;
   % f
foldp = double(subs(p,X,xoldp));
foldvarphi = double(subs(varphi,X,xoldvarphi));
foldq = double(subs(q,X,xoldq));
fold = [ foldp foldvarphi foldq ];                    % True f-points

   % Which interval a new x does belong?
    xNew = xxx;
    for jj=1:length(xNew)
        for i1 = 1:length(xold)
            if xNew(jj) < xold(i1)
               NoInt(jj) = i1-1;
               break
            end % if-end
            if xNew(end) == xold(end)
               NoInt(length(xNew)) = length(xold)-1;
            end % if-end
        end % for-ii-end
    end  % for-jj-end
   %  disp('Which Interval?'), disp(NoInt)

  % Finding Spline
    PP = spline(xold,fold);
    [breaks, coefs , l , k , d ] = unmkpp(PP);

  % Differentiation
   for j2 = 1:length(xNew)
       i2 = NoInt(j2);
       xdel = xNew(j2) - xold(i2);
       fSpline(j2) = coefs(i2,1)*xdel^3 + coefs(i2,2)*xdel^2 ...
                     + coefs(i2,3)*xdel + coefs(i2,4);
       g(j2) = 3*coefs(i2,1)*xdel^2 + 2*coefs(i2,2)*xdel + coefs(i2,3);
   end
   % Maximum Absolute Difference Errors
   fAbsErr(irho1,irho3) = max(abs(fff-fSpline));
   gAbsErr(irho1,irho3) = max(abs(dff-g));
   % End of Differentiation Program

   % Integration
      % Approximation Part
    hApprox = zeros(1,length(xold)+1);
    for i3 = 1:length(xold)-1
        xdel = xold(i3+1) - xold(i3);
        hdel(i3+1) = 1/4*coefs(i3,1)*xdel^4 + 1/3*coefs(i3,2)*xdel^3 ...
                     + 1/2*coefs(i3,3)*xdel^2 + coefs(i3,4)*xdel;
    end
    hApprox = cumsum(hdel);
      % Refinement Part
```

```
    hdelExtra = zeros(1,length(xNew));
    hRefine = zeros(1,length(xNew));
    for j4 = 1:length(xNew)
        i4 = NoInt(j4);
        xdel = xNew(j4) - xold(i4);
        hdelExtra = 1/4*coefs(i4,1)*xdel^4 + 1/3*coefs(i4,2)*xdel^3 ...
               + 1/2*coefs(i4,3)*xdel^2 + coefs(i4,4)*xdel;
        hRefine(j4) = hApprox(i4) + hdelExtra;
    end
    % Maximum Absolute Difference Error
    IAbsErr(irho1,irho3) = max(abs(Iff-hRefine));
    % End of Integration
        end % for-irho3
    end  % for-irho1
end % for-ii-end

% Result
disp('Spline Error'), disp(fAbsErr)
disp('Derivative Error'), disp(gAbsErr)
disp('Integral Error'), disp(IAbsErr)
 figure
 subplot(2,2,1)
 contour(fAbsErr,30), colorbar, set(gca,'fontsize',11,'fontweigh','bold')
 xlabel('\bf \rho_{1} \times 40'), ylabel('\bf \rho_{3} \times 40')
 title('MAE of Function')
 subplot(2,2,2)
 contour(gAbsErr,30), colorbar, set(gca,'fontsize',11,'fontweigh','bold')
 xlabel('\bf \rho_{1} \times 40'), ylabel('\bf \rho_{3} \times 40')
 title('MAE of Derivative')
 subplot(2,2,3)
 contour(IAbsErr,30), colorbar, set(gca,'fontsize',11,'fontweigh','bold')
 xlabel('\bf \rho_{1} \times 40'), ylabel('\bf \rho_{3} \times 40')
 title('MAE of Integral')
saveas(gcf,'DifferentialExtension103DY.jpg')
save('DifferentialExtension103DY')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 DifferentialExtension103DY.m을 실행하면, 마디점들 사이 간격이 어떻게 변함에 따라 최대절대오차가 변화하는 형태를 보여주는 그림 4.6.6이 그려진다. 그림 4.6.6의 좌측상단 그래프는 X 축이 $40 \times \rho_1$ 이고 Y 축이 $40 \times \rho_3$ 인 함수 $f(x)$ 와 3차스플라인 $S_n(x)$ 사이 최대절대오차 $\mathrm{MAE}_f(n)$ 를 그린 등위도(contour map)가 그려진다. 이 그래프에서 알 수 있듯이, 이 최대절대오차는 ρ_1 에 크게 의존하고 ρ_2 와 ρ_3 에는 크게 의존하지 않는다. 즉, 이 최대절대오차는 $p(x)$ 의 정의역 $[-1, -0.05]$ 를 얼마나 잘게 나누냐에 크게 의존한다. 우측상단의 그래프는 X 축이 $40 \times \rho_1$ 이고 Y 축이 $40 \times \rho_3$ 인 도함수 $g(x) = f'(x)$ 와 스플라인도함수 $S'_n(x)$ 사이 최대절대오차 $\mathrm{MAE}_{D_f}(n)$ 를 그린 등위도가 그려진다. 이 그래프에서 알 수 있듯이, 이 최대절대오차에 크게 의존하고 ρ_2 와 ρ_3 에는 크게 의존하지 않는다. 즉, 이 최대절대오차는 $p(x)$ 의 정의역 $[-1, -0.05]$ 를 얼마나 잘게 나누냐에 크게 의존한다. 좌측하단의 그래프는 X 축이 $40 \times \rho_1$ 이고 Y 축이 $40 \times \rho_3$ 인 정적분함수 $h(x) = \int_0^x f(t)dt$ 와 스플라인적분함수 $\int_0^x S_n(t)dt$ 사이 최대절대오차 $\mathrm{MAE}_{I_f}(n)$ 를 그린 등위도가 그려진다. 이 그래프에서 알 수

있듯이, 이 최대절대오차는 ρ_1에 크게 의존하고 ρ_2와 ρ_3에는 크게 의존하지 않는다. 즉, 이 최대절대오차는 $p(x)$의 정의역 $[-1, -0.05]$를 얼마나 잘게 나누냐에 크게 의존한다. ■

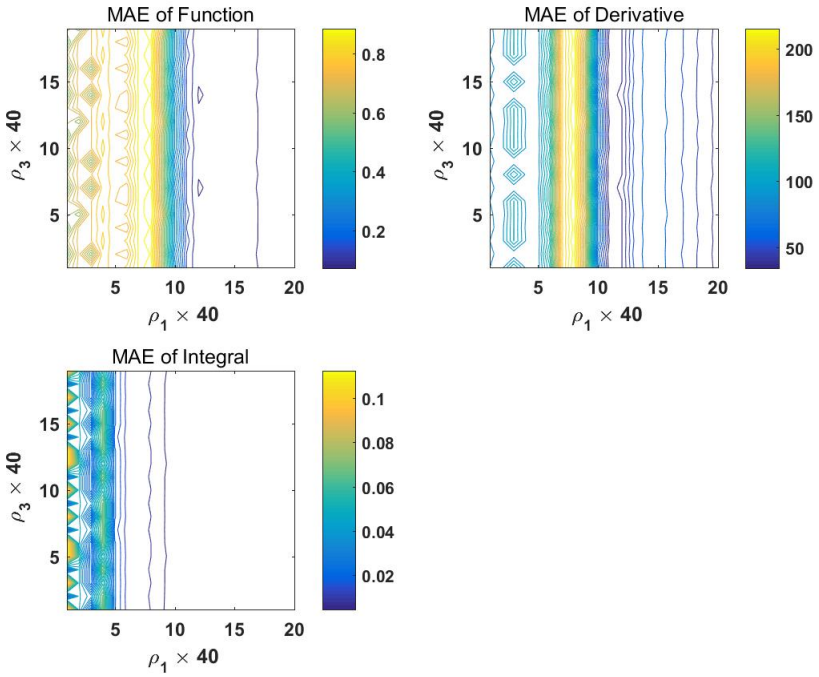


그림 4.6.6. 마디점 간격과 최대절대오차

예제 4.6.5 예제 4.6.3을 다시 생각해보자. 소구간 $[-1, -0.05]$에서 등간격인 m_1개 마디점들을, 소구간 $(-0.05, 0.05)$에서 등간격인 m_2개 마디점들을, 그리고 소구간 $[0.05, 1]$에서 등간격인 m_3개 마디점들을 선택하기로 하자. 여기서 m_1, m_2와 m_3는 다음 식들을 만족한다고 하자.

$$m_1 \approx 0.1875[n+1], \quad m_2 \approx 0.2750[n+1], \quad m_3 \approx 0.5375[n+1] \tag{1}$$

고정된 비율들 $[\alpha_1, \alpha_2, \alpha_3] = [0.1875, 0.2750, 0.5375]$ 그리고 각 n $(= 2^3, 2^4, 2^5, 2^6, 2^7, 2^8)$에 따라 최대절대오차가 어떻게 변하는지를 살펴보기 위해서, 다음 MATLAB프로그램 DifferentialExtension104DY.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: DifferentialExtension104DY.m
%  Extension of Continuously Differentiable Function
%  Cubic-Spline approximation
%  rho_1 = 0.1875,  rho_2 = 0.2750,  rho_3 = 0.5375
%  Number of Knots vs. Minimum Absolute Error
```

```
%  True Values for n = 2^11
%  Programmed by CDY
%-------------------------------------------------------------------------
clear all, close all, clc
format long g
% Calculate the function values
xp = -0.05, xt = 0.05
p0 = 1/(exp(-10) + 1)
p1 = -(200*exp(-10))/(exp(-10) + 1)^2
q0 = 1/(exp(10) + 1)
q1 =  -(200*exp(10))/(exp(10) + 1)^2
syms X
varphi = 1/(1+exp(200*X))
p = (1+2*X*exp(-(X+1)^2))*(X-xp)^2 + p1*(X-xp) + p0
q = q0 + q1*(X-xt) + 1/2*(X-xt)^2*sin(32*pi*(X-xt)^2)
%
% load true data with n +1 = 2^11 +1 = 2048 + 1
Sdata = dlmread('SplineTrueValues.dat');
xxx = Sdata(1,:);
fff = Sdata(2,:);
dff = Sdata(3,:);
Iff = Sdata(4,:);
log2nMax = 8;   % n = 2^(log2nMax)
Delta = zeros(1,log2nMax);
rho1 = 7.5*0.025
rho3 = 11*0.025           %  rho3 = 11*0.025
rho2 = 1 - rho1 - rho3

for ii = 3:log2nMax
    nn = 2^ii;
    Delta(ii) = 1/nn;
 %   disp('No of Given Points'), disp(nn+1)
    clear xold fold
    dumIndex = 1:2^(log2nMax-ii):length(xxx);

   %  Given Point Ratios
m1 = max(round(rho1*(nn+1)),1);
m2 = max(round(rho2*(nn+1)),1);
m3 = nn+1-m1-m2;
   % x-points
xoldp = linspace(-1,xp,m1);
xoldvarphiDum = linspace(xp,xt,m2+2); xoldvarphi = xoldvarphiDum(2:end-1);
xoldq = linspace(xt,1,m3);
xold = [ xoldp xoldvarphi xoldq ];                  % True x-points
xold(end) = 1;
   % f
foldp = double(subs(p,X,xoldp));
foldvarphi = double(subs(varphi,X,xoldvarphi));
foldq = double(subs(q,X,xoldq));
fold = [ foldp foldvarphi foldq ];                   % True f-points

   % Which interval a new x does belong?
    xNew = xxx;
    for jj=1:length(xNew)
        for i1 = 1:length(xold)
            if xNew(jj) < xold(i1)
               NoInt(jj) = max(i1-1,1);
               break
            end % if-end
            if xNew(end) == xold(end)
               NoInt(length(xNew)) = length(xold)-1;
```

```
            end % if-end
        end % for-ii-end
    end  % for-jj-end
   %  disp('Which Interval?'), disp(NoInt)
  % Finding Spline
    PP = spline(xold,fold);
    [breaks, coefs , l , k , d ] = unmkpp(PP);
  % Differentiation
   for j2 = 1:length(xNew)
       i2 = NoInt(j2);
       xdel = xNew(j2) - xold(i2);
       fSpline(j2) = coefs(i2,1)*xdel^3 + coefs(i2,2)*xdel^2 ...
                   + coefs(i2,3)*xdel + coefs(i2,4);
       g(j2) = 3*coefs(i2,1)*xdel^2 + 2*coefs(i2,2)*xdel + coefs(i2,3);
   end
   % Maximum Absolute Difference Errors
   fAbsErr(ii) = max(abs(fff-fSpline));
   gAbsErr(ii) = max(abs(dff-g));
   % End of Differentiation Program

   % Integration
      % Approximation Part
    hApprox = zeros(1,length(xold)+1);
    for i3 = 1:length(xold)-1
        xdel = xold(i3+1) - xold(i3);
        hdel(i3+1) = 1/4*coefs(i3,1)*xdel^4 + 1/3*coefs(i3,2)*xdel^3 ...
                   + 1/2*coefs(i3,3)*xdel^2 + coefs(i3,4)*xdel;
    end
    hApprox = cumsum(hdel);
      % Refinement Part
    hdelExtra = zeros(1,length(xNew));
    hRefine = zeros(1,length(xNew));
    for j4 = 1:length(xNew)
        i4 = NoInt(j4);
        xdel = xNew(j4) - xold(i4);
        hdelExtra = 1/4*coefs(i4,1)*xdel^4 + 1/3*coefs(i4,2)*xdel^3 ...
               + 1/2*coefs(i4,3)*xdel^2 + coefs(i4,4)*xdel;
        hRefine(j4) = hApprox(i4) + hdelExtra;
    end
    % Maximum Absolute Difference Error
    IAbsErr(ii) = max(abs(Iff-hRefine));
    % End of Integration

end % for-ii-end

% % Result
% Spline
subplot(2,2,1)
  disp('Max Abs Error of Splines')
  disp(fAbsErr(3:end))
  loglog(Delta(3:end),fAbsErr(3:end),'k-o','LineWidth',1.5)
  set(gca,'fontsize',11,'fontweigh','bold','xdir','reverse', ...
      'xlim',[2^-8 2^-3])
  set(gca,'xtick',[2^-8  2^-6  2^-4  ])
  % set(gca,'xtick',[2^-8 2^-7 2^-6 2^-5 2^-4 2^-3 ])
  title('\bf MAE of Splines')
% Derivative
subplot(2,2,2)
  disp('Max Abs Error of Derivatives')
  disp(gAbsErr(3:end))
  loglog(Delta(3:end),gAbsErr(3:end),'r-o','LineWidth',1.5)
```

```
  set(gca,'fontsize',11,'fontweigh','bold','xdir','reverse', ...
      'xlim',[2^-8 2^-3])
  set(gca,'xtick',[2^-8  2^-6  2^-4  ])
  % set(gca,'xtick',[2^-8 2^-7 2^-6 2^-5 2^-4 2^-3 ])
  title('\bf MAE of Derivatives')
% Integral
subplot(2,2,3)
  disp('Max Abs Error of Integrals')
  disp(IAbsErr(3:end))
  loglog(Delta(3:end),IAbsErr(3:end),'b-o','LineWidth',1.5)
  set(gca,'fontsize',11,'fontweigh','bold','xdir','reverse', ...
      'xlim',[2^-8 2^-3])
  set(gca,'xtick',[2^-8  2^-6  2^-4  ])
  % set(gca,'xtick',[2^-8 2^-7 2^-6 2^-5 2^-4 2^-3 ])
  title('\bf MAE of Integrals')
saveas(gcf,'DifferentialExtension104DY.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 DifferentialExtension104DY.m을 실행하면, n이 어떻게 변함에 따라 최대절대오차가 변화하는 형태를 보여주는 그림 4.6.7이 그려진다. 이 그래프에서 마디점들의 개수 n이 증가함에 따라 이 최대절대오차가 감소하는 경향을 보인다. 그림 4.6.7의 좌측상단 그래프는 X 축에는 2^{-n}을 Y 축에는 함수 $f(x)$와 3차스플라인 $S_n(x)$ 사이 최대절대오차 $\mathrm{MAE}_f(n)$를 그린 것이다. 우측상단 그래프는 함수 $f(x)$와 3차스플라인 $S_n(x)$의 미분함수들의 최대절대오차 $\mathrm{MAE}_{D_f}(n)$가 그려진다. 이 그래프에서 마디점들의 개수 n이 증가함에 따라 이 최대절대오차가 감소하는 경향을 보인다. 좌측하단 그래프는 함수 $f(x)$와 3차스플라인 $S_n(x)$의 적분함수들 사이 최대절대오차 $\mathrm{MAE}_{I_f}(n)$가 그려진다. 이 그래프에서 마디점들의 개수 n이 증가함에 따라 이 최대절대오차가 감소하는 경향을 보인다. 그림 4.6.5의 그래프들은 예제 4.6.3의 그래프들과 비슷하다. 즉, 마디점들의 개수들 m_1, m_2와 m_3의 비율이 고정되면 최대절대오차는 n에 상관없이 비슷한 형태를 보인다. ■

제4.7절 변동성곡면

서울대학교 금융경제연구원은 2015년 11월 28일 홍콩과 싱가폴에서 활동하는 퀀트들을 초대해서 '나는 *Qunat*다 *3*'를 개최하였다. 이 과정에서 변동성곡면(volatility surface)에 대한 포스터세션도 열었다. 이 절은 이 포스터세션에 참가한 서울대학교 산업공학과 박사과정 이민혁군의 보고서를 약간 수정한 것이다.

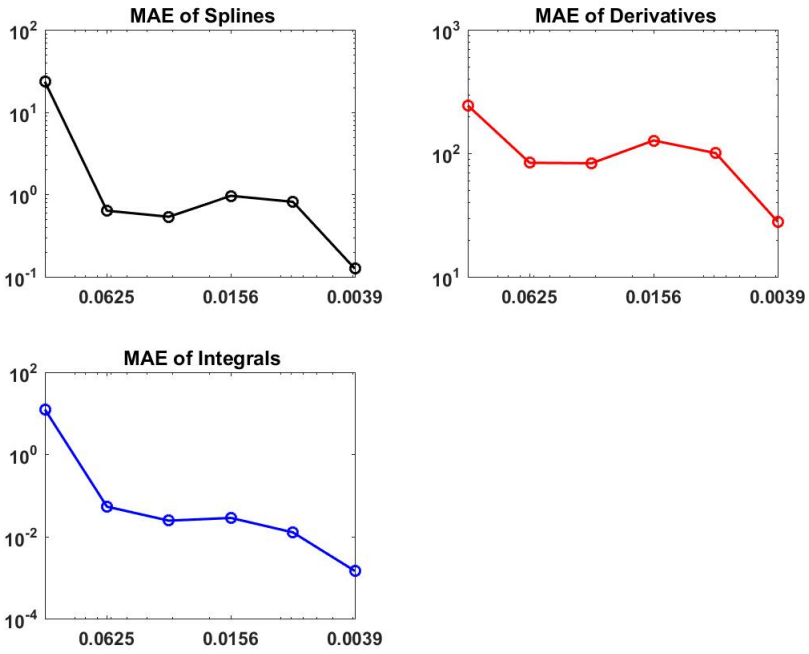


그림 4.6.7. 마디점의 개수와 최대절대오차

4.7.1 변동성곡면 추정문제

금융시장에서 거래할 때 가장 중요한 정보는 변동성(volatility)이다. 예를 들어, 옵션거래에서 원자산의 변동성을 아는 것은 매우 중요하다. Black-Scholes옵션가치평가식을 구성하는 요소는 원자산, 행사가격, 잔여기간, 무위험이자율 그리고 변동성이다. 이 중에서 원자산, 행사가격, 잔여기간, 무위험이자율은 시장에서 쉽게 알 수 있다. 따라서, Black-Scholes옵션가치를 안다는 것은 원자산의 변동성을 아는 것과 동치이다. 실제 옵션시장에서는 옵션가격이 아닌 변동성으로 거래한다. 변동성을 추정하는 것은 그리 간단한 일이 아니다. 많은 학자들과 실무자들이 변동성을 추정하는 다양한 방법들을 제시하고 있으나 아직도 많은 사람들이 이의없이 받아들이는 변동성추정법은 없는 것 같다. 본 절에서는 Fengler & Hin [21]이 제시한 방법을 약간 변형해서 변동성곡면(volatility surface)을 추정하고자 한다.

변동성곡면를 구하기 위해서는 먼저 옵션가격데이터를 Black-Scholes옵션가치평가식에 대입해서 내재변동성(implied volatility)을 구한다. 이렇게 구해진 변동성들을 내삽해서(interpolating) 변동성곡면를 구할 수 있다. 여기서는 앞에서 설명한 B-스플라인을 사용해서 내삽을 행하기로 하자. 옵션가격은 행사가격(strike price)과 잔여기간(tenor)의 함수이다. 따라서 어떤 옵션의 시장가격이 자신의 행사가격이나 잔여기간 뿐아니라 이들과 비슷한 행사가격이나 잔여기간에 해당하는 옵션가격에 영향을 받는다고 할 수 있다. 따라서 옵션가격이나

내재변동성을 내삽하는데 B-스플라인은 적절한 방법이라고 할 수 있다.

다음과 같은 텐서곱(tensor product) B-스플라인을 사용하기로 하자.

$$s(x,y)=\sum_{j_1=0}^{q_1}\sum_{j_2=0}^{q_2}\theta_{j_1,j_2}B_{j_1,p_1}(x)B_{j_2,p_2}(y) \tag{4.7.1}$$

주어진 데이터에 B-스플라인을 적합시키기 위해서는 최적화과정을 거쳐야 한다. 즉, 주어진 데이터와 B-스플라인 사이 오차제곱합을 최소화해야한다. 점 $[x_i,y_i]$에서 관측값이 z_i라고 하자. 주어진 데이터세트 $\{[x_i,y_i,z_i]:i=1,2,\cdots\}$에 대해서 다음 목적함수를 최소화하기로 하자.

$$\arg\min_s\left\{\frac{1}{n}\sum_{i=1}^{n}[z_i-s(x_i,y_i)]^2+\lambda\,(\text{Sum of squared curvature})\right\} \tag{4.7.2}$$

이 목적함수에서 식 $\lambda=0$이 성립한다면 일반적인 B-스플라인이다. 그러나, 식 $\lambda>0$가 성립하면, B-스플라인은 평활해진다. 즉, 식 (4.7.2)의 목적함수는 벌칙B-스플라인이고, 식 (4.7.2)를 만족하는 B-스플라인 $\hat{s}(x,y)$이 우리가 구하고자하는 변동성곡률이다.

이 변동성곡면은 콜옵션가격을 나타내는 곡면이므로, 콜옵션가격들 사이에 재정이 존재하지 않을 조건을 만족해야 한다. 이 절에서는 Carr & Madan [13]이 제시한 무재정조건을 사용하기로 하자. 즉, 행사가격이 K이고 잔여기간이 τ인 콜옵션가격 $C(K,\tau)$가 다음 조건들을 만족한다고 가정하자.

(a) 잔여기간 τ가 고정되었을 때 콜옵션가격 $C(K,\tau)$는 행사가격 K에 대해 볼록하다.

(b) 각 (K,τ)에 대해서 다음 부등식들이 성립한다.

$$[S_t-K]^+\le C(K,\tau)\le S_t$$

(c) 각 잔여기간 τ에 대해서 다음 식이 성립한다.

$$C(K,0)=[S_t-K]^+$$

(d) 각 잔여기간 τ에 대해서 다음 식이 성립한다.

$$\lim_{K\to\infty}C(K,\tau)=0$$

(e) 각 행사가격 K에 대해서, 잔여기간 τ가 증가함에 따라 $C(K,\tau)$는 감소하지 않는다.

4.7.2 데이터세트

이 최적화문제에 사용된 데이터는 2015.09.15. KOSPI200 Option 데이터이다. 선물가격은 $F = 233.10$, 이자율은 $r = 0.016$이고 사용한 콜가격과 Black-Scholes옵션가치평가식을 사용해서 구한 내재변동성값(implied volatility value)은 아래 표와 같다. 왼쪽 표는 Call option price table, 오른쪽 표는 Implied volatility table이다. 가로축은 잔여기간이고 세로축은 행사가격이다.

4.7.3 결과

[1] 행사가격에 대한 콜옵션가격곡선

행사가격에 대한 콜옵션가격곡선(call option price curve with strike price unde fixed maturity)을 살펴보자. 그림 4.7.1는 잔여기간(tenor, time to maturity)이 0.0630인 콜옵션가격들을 3차(order 3) B-스플라인을 사용해서 무재정조건을 만족하는 곡선에 적합시킨 그래프이다. 이때 λ값은 100으로 설정하였다. 청색 점은 실제 콜옵션가격이고 적색 선은 3차 B-스플라인으로 적합시킨 그래프이다. 적색 선에서 보이는 울퉁불퉁한 부분이 B-스플라인을 사용해서 평활하게 적합되었음을 알 수 있다.

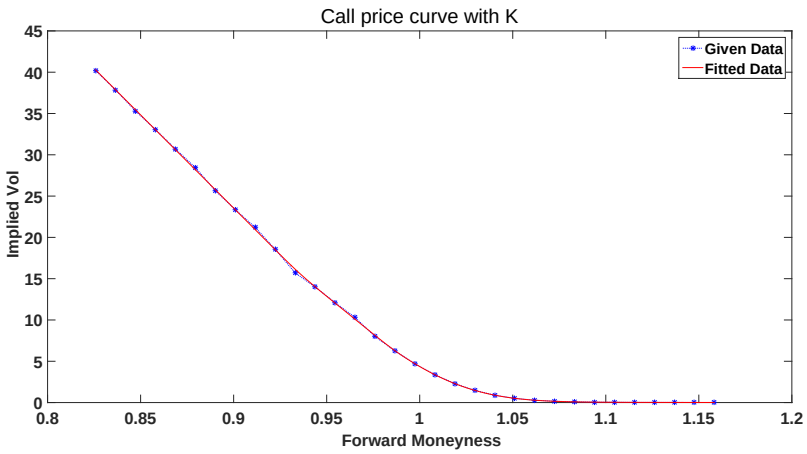


그림 4.7.1. 행사가격에 대한 콜옵션가격곡선

그림 4.7.2는 콜옵션가격과 그림 4.7.1의 적합된 콜옵션가격곡선의 차이를 그린 것이다.

[2] 행사가격과 잔여기간에 대한 콜옵션가격곡면

	0.0630	0.1589	0.2356
192.5	40.2	41	42.55
195	37.85	38.8	
197.5	35.3	36.5	
200	33.05		
202.5	30.7	31.7	
205	28.45	29.45	
207.5	25.65	27.15	
210	23.35		
212.5	21.2		
215	18.55	20.5	
217.5	15.7		
220	14		17.6
222.5	12.1		
225	10.3		
227.5	8	10.6	
230	6.25	8.35	10.6
232.5	4.7	7.33	9.3
235	3.37	6.08	7.72
237.5	2.29	4.62	6.4
240	1.47	3.7	5.26
242.5	0.89	2.63	4.32
245	0.51	1.89	3.3
247.5	0.29	1.32	2.6
250	0.16	0.9	1.97
252.5	0.09	0.59	1.46
255	0.05	0.39	1.12
257.5	0.04	0.27	0.81
260	0.02	0.18	0.6
262.5	0.02	0.11	0.44
265	0.02	0.09	0.34
267.5	0.01	0.06	0.27
270	0.01	0.05	0.2

(a) 콜옵션

	0.0630	0.1589	0.2356
192.5	0.4094	0.3185	0.3393
195	0.4174	0.3251	
197.5	0.3825	0.3205	
200	0.3984		
202.5	0.3914	0.2947	
205	0.3916	0.2890	
207.5	0.3310	0.2791	
210	0.3246		
212.5	0.3268		
215	0.2846	0.2521	
217.5	0.2261		
220	0.2547		0.2224
222.5	0.2560		
225	0.2550		
227.5	0.2239	0.2099	
230	0.2130	0.1868	0.2009
232.5	0.2030	0.1944	0.2004
235	0.1936	0.1919	0.1913
237.5	0.1852	0.1796	0.1853
240	0.1780	0.1782	0.1807
242.5	0.1722	0.1677	0.1780
245	0.1678	0.1623	0.1702
247.5	0.1661	0.1577	0.1673
250	0.1653	0.1541	0.1632
252.5	0.1662	0.1505	0.1594
255	0.1676	0.1488	0.1585
257.5	0.1773	0.1493	0.1557
260	0.1765	0.1492	0.1548
262.5	0.1897	0.1475	0.1541
265	0.2027	0.1526	0.1553
267.5	0.2008	0.1532	0.1574
270	0.2126	0.1583	0.1576

(b) 풋옵션

표 4.7.1. 내재변동성

행사가격과 잔여기간에 대한 콜옵션가격곡면(call option price surface with strike price and time to maturity, τ)을 살펴보자. 그림 4.7.3는 콜옵션가격들을 3차B-스플라인을 사용해서 무재정조건을 만족하는 곡면에 적합시킨 그래프이다. 이 적합곡면 상의 흑색 점은 실제 콜옵션가격을 나타낸다. 이때 λ값을 100으로 설정하였다.

[3] 행사가격에 대한 내재변동성곡선

행사가격에 대한 내재변동성곡선(volatility curve with strike price under fixed tenor)을 살펴보자. 그림 4.7.4는 잔여기간이 0.0630인 내재변동성(implied volatility)들을 3차B-스플라인을 사용해서 무재정조건을 만족하는 곡선에 적합시킨 그래프이다. 이때 λ값은 10으로

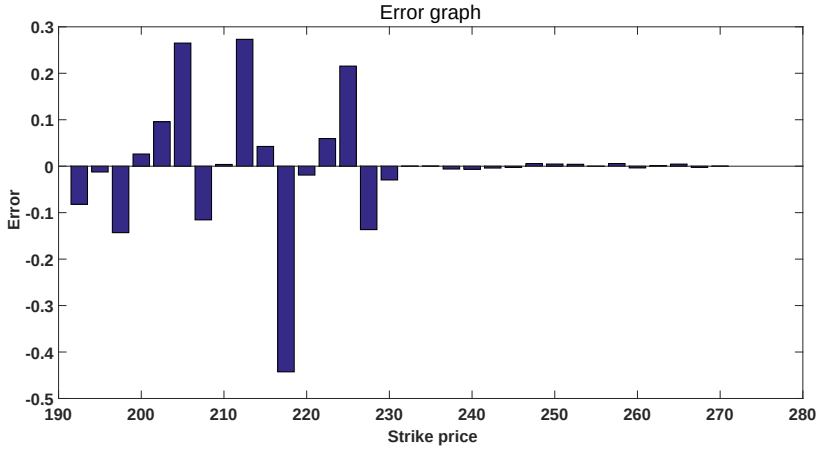


그림 4.7.2. 콜옵션가격곡선의 오차

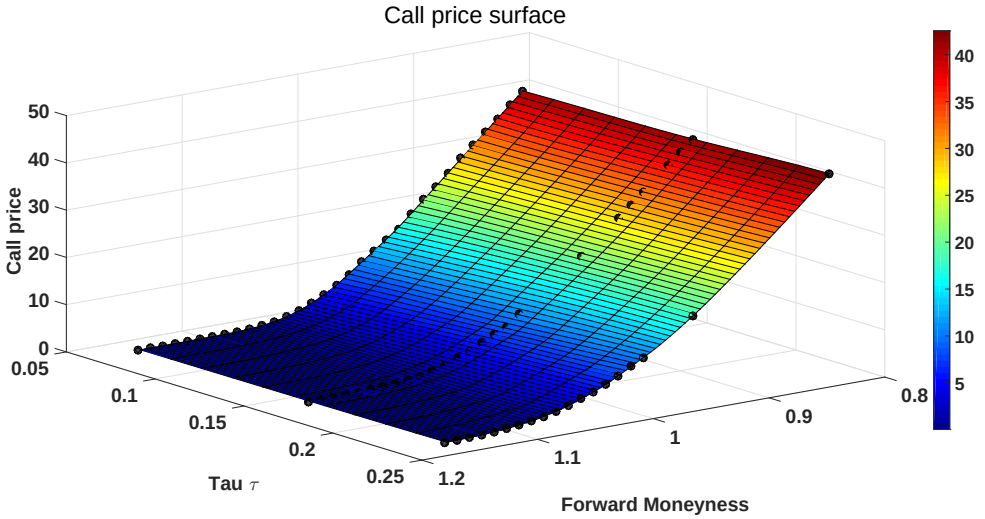


그림 4.7.3. 콜옵션가격곡면

설정하였다. 청색 점은 내재변동성이고 적색 선은 3차B-스플라인으로 적합시킨 그래프이다. 적색 선에서 보이는 울퉁불퉁한 부분이 B-스플라인을 사용해서 평활하게 적합되었음을 알 수 있다.

그림 4.7.5는 내재변동성과 그림 4.7.4의 적합된 내재변동성곡선의 차이인 오차를 그린 것이다. 행사가격의 중간 부근에서는 오차가 작고 행사가격이 크거나 작은 경우에는 오차가 크다는 것을 알 수 있다.

그림 4.7.4는 모수 λ의 값으로 10을 선택하였다. 모수 λ의 변화에 따라 행사가격에 대한 내재변동성곡선의 어떻게 변하는지 관찰하기 위해 λ값이 100인 변동성곡선을 그림 4.7.6에 수록하였다. 그림 4.7.4과 그림 4.7.6을 비교하면, 모수 λ가 10인 그래프보다 모수 λ가 100인 그래프가 더 평활하다. 예를 들어, λ값이 10인 경우 선행머니니스(forward moneyness)가

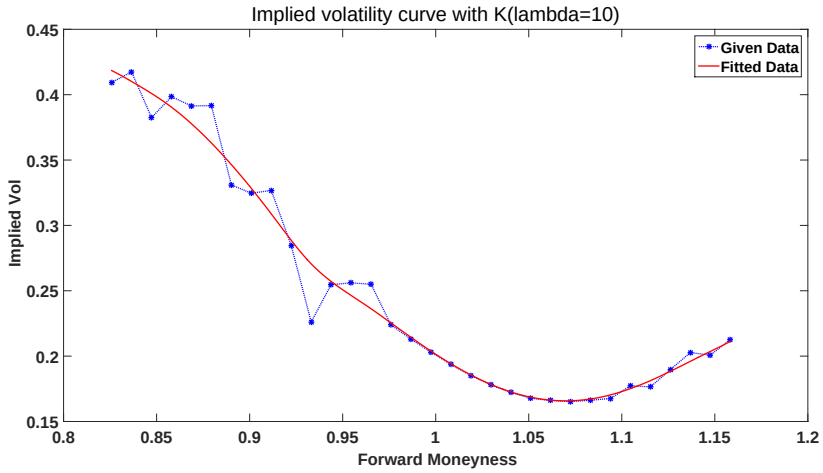


그림 4.7.4. 행사가격에 대한 콜옵션가격곡선 ($\lambda = 10$)

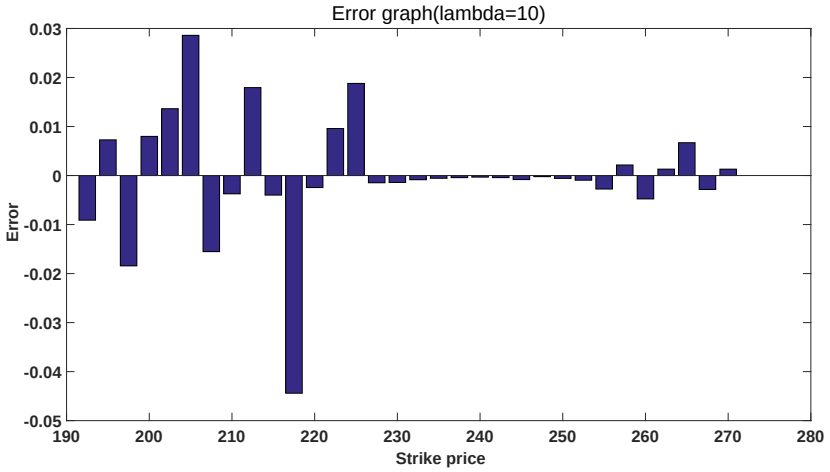


그림 4.7.5. 내재변동성곡선의 오차 ($\lambda = 10$)

0.85 부근에서 그래프가 오목이나 (concave) λ값이 100인 경우에는 그래프에서는 그런 현상이 나타나지 않는다.

그림 4.7.7는 내재변동성과 그림 4.7.6의 적합된 내재변동성곡선의 차이인 오차를 그린 것이다. 모수 λ가 커지면, 실제 데이터에 의한 오차 부분보다 벌칙함수에 더 큰 가중값을 가한다. 따라서 그래프가 더 평활해지고, 실제 오차는 더 커진다. 즉, 모수 λ가 10인 그림 4.7.5의 오차보다 모수 λ가 100인 그림 4.7.7의 오차가 더 크다.

[4] 행사가격과 잔여기간에 대한 내재변동성곡면

행사가격과 잔여기간에 대한 내재변동성곡면 (implied volatility surface with strike price and time to maturity, τ)을 살펴보자. 그림 4.7.8는 내재변동성들을 3차B-스플라인을 사용해서 무재정조건을 만족하는 곡면에 적합시킨 그래프이다. 이 내재변동성곡면 상의 흑색 점은 시장에서 관찰된 콜옵션가격으로부터 계산된 내재변동성을 나타낸다. 이때 λ값을 1로

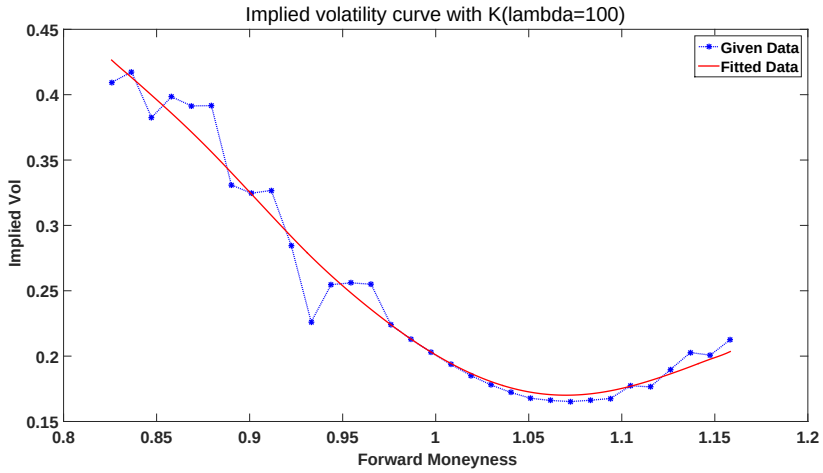


그림 4.7.6. 행사가격에 대한 콜옵션가격곡선 ($\lambda = 100$)

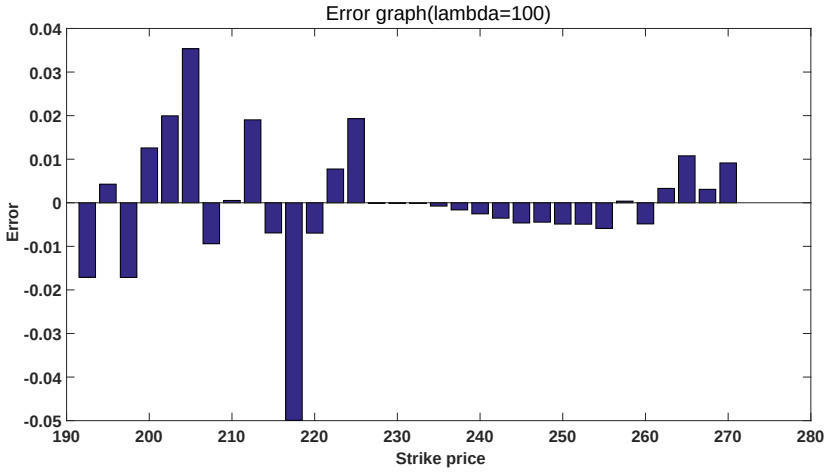


그림 4.7.7. 내재변동성곡선의 오차 ($\lambda = 100$)

설정하였다.

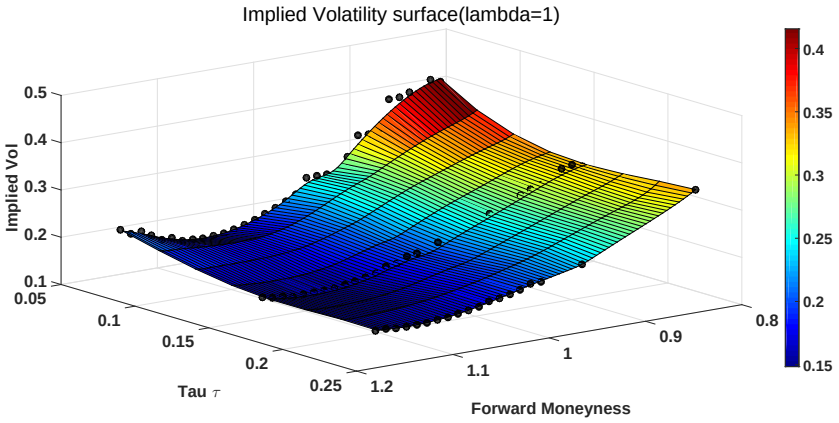


그림 4.7.8. 내재변동성곡면 (λ=1)

그림 4.7.8에 그려진 내재변동성곡면에 Black-Scholes옵션가치평가식을 적용해서 구한 콜옵션가격을 그린 3차원 그래프가 그림 4.7.9에 그려져 있다. 흑색 점은 시장에서 관찰된 콜옵션가격을 나타낸다. 그림 4.7.9의 콜옵션가격곡면은 무재정조건을 만족함을 확인할 수 있다.

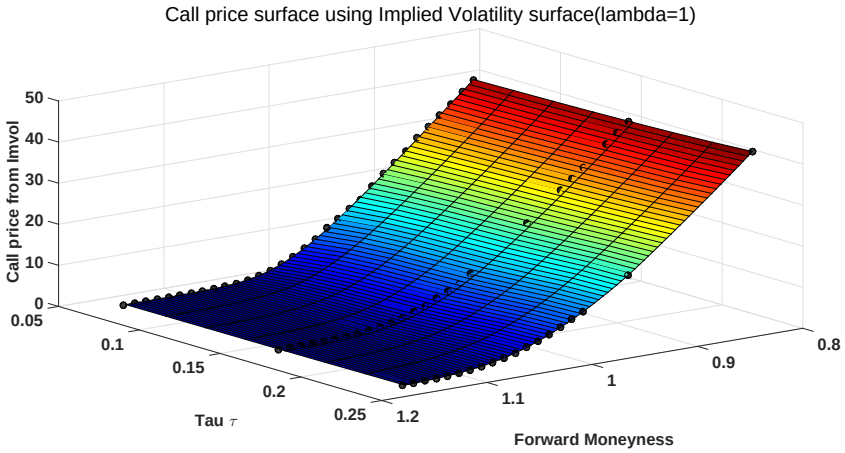


그림 4.7.9. 추정콜옵션가격곡면 (λ=1)

그림 4.7.9의 내재변동성곡면과 시장에서 추정된 콜옵션가격으로부터 구한 내재변동성의 차이를 그린 3차원 그래프가 그림 4.7.10에 그려져 있다. 이 그림에서 Tau τ가 1이면 잔여기간이 0.0630이고, Tau τ가 2이면 잔여기간이 0.1589이고, Tau τ가 3이면 잔여기간이 0.2356이다. 몇 개 점들을 제외하고는 오차가 크지 않음을 알 수 있다.

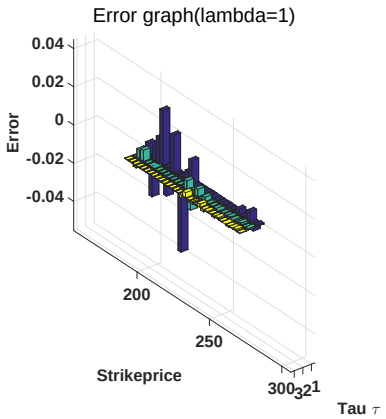


그림 4.7.10. 내재변동성곡면의 오차 (λ=1)

그림 4.7.11은 λ값이 0.01인 경우에 해당하는 내재변동성곡면이고, 그림 4.7.12은 λ값이 100인 경우에 해당하는 내재변동성곡면이다. 이 내재변동성곡면 상의 흑색 점은 시장에서 관찰된 콜옵션가격으로부터 계산된 내재변동성을 나타낸다. 그림 4.7.8, 그림 4.7.11 그리고

그림 4.7.12에서 알 수 있듯이, λ값이 클수록 내재변동성곡면은 더 평활하다.

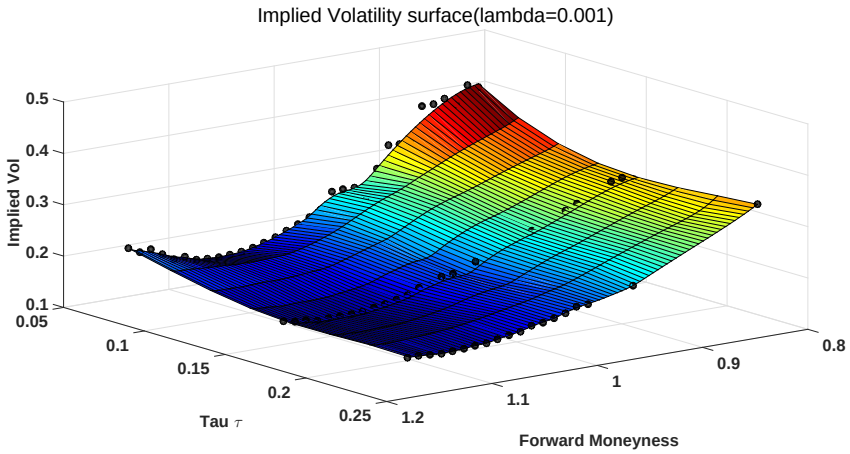


그림 4.7.11. 내재변동성곡면 (λ=0.01)

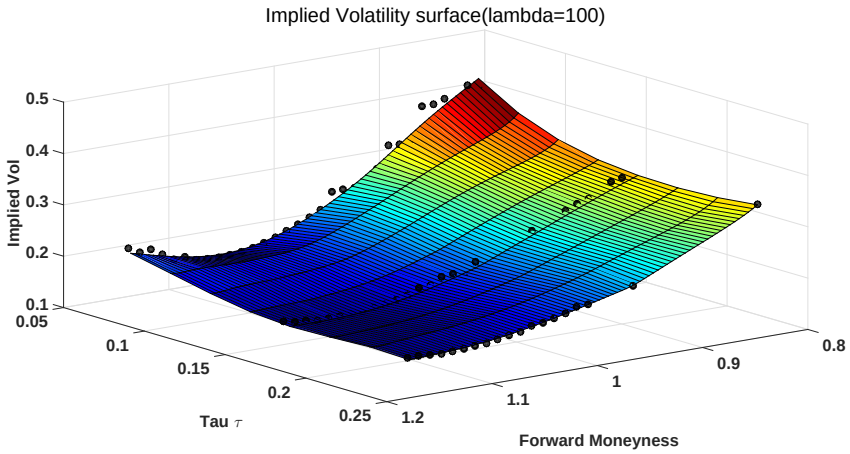


그림 4.7.12. 내재변동성곡면 (λ=100)

4.7.4 Matlab Code

```
% ------------------------------------------------------------------
% Filename: splineofimvol.m
% B-Spline(second derivative penalized) of imvol w/ no arbitrage constraint
% Programmed by MinHyuk Lee(Dep. Industrial Engineering in SNU)
% superou@snu.ac.kr
% with 'calldata.mat','bsplinebase.m','bs_option_price.m'...
%      'IV_usingNewtonRaphson.m','nonlcon3.m','nonlcon4.m'
% ------------------------------------------------------------------
clear all, close all
tic
global strikeprice tau r s
% Inputs
load('calldata.mat')
%strikeprice, tau, call price, implied vol, S, r, Forward price

lengst=length(strikeprice);
```

```
lengtau=length(tau);
imvol=zeros(lengst,lengtau);
blsimvol=zeros(lengst,lengtau);
for i=1:lengtau
    for j=1:lengst
        imvol(j,i)=IV_usingNewtonRaphson(s,strikeprice(j),r,tau(i),callp(j,i)
    ,1);
        blsimvol(j,i)=blsimpv(s,strikeprice(j),r,tau(i),callp(j,i));
    end
end

y=blsimvol(:,1);
yy=blsimvol;
%% Making Bspline base(2D)
for i=1:lengst
    if i==1
        temp=bsplinebase(strikeprice,strikeprice(i));
        B1=temp;
    else
        temp=bsplinebase(strikeprice,strikeprice(i));
        B1=[B1;temp];
    end
end

for i=1:lengtau
    if i==1
        temp=bsplinebase(tau,tau(i));
        B2=temp;
    else
        temp=bsplinebase(tau,tau(i));
        B2=[B2;temp];
    end
end

%% For no arbitrage constraint
global xx constbase
xx=(strikeprice(1):0.5:strikeprice(end))'; %split strikeprice
lengthxx = length(xx);
yhat=zeros(lengthxx,1);

for i=1:lengthxx
    constbase{i}=bsplinebase(strikeprice,xx(i));
end

%% With fixed tau, Making imvol B-spline with strike price(w/ constraint)
lambda=10;
PenaltyOrder=2;
D = diff(eye(length(y)),PenaltyOrder);

options.MaxFunEvals = 200000;
ahat2d = fmincon(@(a) norm(y-(B1*a))^2+lambda*norm(D*B1*a)^2,0.4*ones(39,1)
    ,[],[],[],[],[],[],@nonlcon3,options);

imvolhat2d=B1*ahat2d;

imvolhat2dxx=zeros(lengthxx,1);
for i=1:lengthxx
    imvolhat2dxx(i,1)=bsplinebase(strikeprice,xx(i))*ahat2d;
end

%% Graph of 2-D (Implied volatility with strike price)
```

```
% figure
% plot(strikeprice,y,'b:*',xx,imvolhat2dxx,'-','LineWidth',1.5)
% set(gca,'fontweigh','bold')
% legend('Given Data','Fitted Data','location','NW')
% xlabel('\bf Strike price K');
% ylabel('\bf Implied Vol');

% figure
% plot(strikeprice/F,y,'b:*','LineWidth',2)
% title('Implied volatility curve with K(lambda=10)','fontsize',30)
% hold on
% plot(xx/F,imvolhat2dxx,'r-','Color','r','LineWidth',2)
% set(gca,'fontweigh','bold','fontsize',25)
% legend('Given Data','Fitted Data','location','NE')
% xlabel('\bf Forward Moneyness','fontsize',25);
% ylabel('\bf Implied Vol','fontsize',25);
% hold off
%
%
% figure
% bar(strikeprice,y-imvolhat2d)
% title('Error graph(lambda=10)','fontsize',30)
% set(gca,'fontweigh','bold','fontsize',25)
% xlabel('\bf Strike price','fontsize',25);
% ylabel('\bf Error','fontsize',25);
%% Making Bspline base(3D)
BB1=B1(:,2:end-4);
BB2=B2(:,2:end-4);

termofzero=zeros(lengst,lengtau);
for i=1:lengtau
    count1=0;
    countzero=1;
   for j=1:lengst
      if i==1 && j==1
         temp=kron(BB1(j,:),BB2(i,:));
         BBB=temp;
         yyy(1,1)=yy(1,1);
         count=1;
      else
          if isnan(blsimvol(j,i))~=1
              count=count+1;
              yyy(count,1)=yy(j,i);
              temp=kron(BB1(j,:),BB2(i,:));
              BBB=[BBB;temp];
          end
      end

      if callp(j,i)~=0
          count1=count1+1;
          termofzero(count1,i)=termofzero(count1,i)+countzero;
          countzero=1;
      else
          countzero=countzero+1;
      end

   end
end
numofzero=sum((callp==0));

%% Calculating curvature
```

```
for i=1:lengtau
    D4{i}=zeros(lengst-numofzero(i)-2,lengst-numofzero(i));
end

for i=1:lengtau
   for j=1:lengst-numofzero(i)-2
      D4{i}(j,j)=2/(termofzero(j+1,i)*(termofzero(j+1,i)+termofzero(j+2,i)));
      D4{i}(j,j+1)=-2/(termofzero(j+1,i)*termofzero(j+2,i));
      D4{i}(j,j+2)=2/(termofzero(j+2,i)*(termofzero(j+1,i)+termofzero(j+2,i)))
    ;
   end
end

%% For no arbitrage constraint
global xaxis yaxis
xaxis=(strikeprice(1):1:strikeprice(end))';
yaxis=(tau(1):(tau(end)-tau(1))/7:tau(end))';

lengthx=length(xaxis);
lengthy=length(yaxis);

global temp5
for j=1:lengthy
    for i=1:lengthx
        temp1=bsplinebase(strikeprice,xaxis(i));
        temp2=bsplinebase(tau,yaxis(j));
        temp3=temp1(:,2:end-4);
        temp4=temp2(:,2:end-4);
        temp5{i,j}=kron(temp3,temp4);
    end
end

%% Bspline optimization(3D)
lambda=0.001;
PenaltyOrder=2;

options.MaxFunEvals = 1000000;
options.MaxIter = 200000;
options.TolX = 1.000000e-250;
[ahat3d,fval,exitflag,output] =fmincon(@(a) norm(yyy-(BBB*a))^2+lambda...
    *(norm(D4{1}*BBB(1:lengst-numofzero(1),:)*a)^2+...
    norm(D4{2}*BBB(lengst-numofzero(1)+1:2*lengst-numofzero(1)-numofzero(2),:)
    *a)^2+...
    norm(D4{3}*BBB(2*lengst-numofzero(1)-numofzero(2)+1:3*lengst-sum(numofzero
    ),:)*a)^2)...
    ,0.1*ones(length(BBB(1,:)),1),[],[],[],[],[],[],@nonlcon4,options);

yhat3d=BBB*ahat3d;

xxaxis=zeros(lengthx,lengthy);
yyaxis=zeros(lengthx,lengthy);
zaxis=zeros(lengthx,lengthy);
callhat=zeros(lengthx,lengthy);

for j=1:lengthy
    for i=1:lengthx
        zaxis(i,j)=temp5{i,j}*ahat3d;
        callhat(i,j)=bs_option_price(s,xaxis(i),r,zaxis(i,j),yaxis(j),1);
        xxaxis(i,j)=xaxis(i);
        yyaxis(i,j)=yaxis(j);
    end
```

```
end

test1=diff(callhat')'; %test dx/dt>0
test2=diff(callhat); %test -1<dx/dk<0

%% graph
% figure
% surf(xxaxis, yyaxis, zaxis);
% xlabel('\bf Strike price K');
% ylabel('\bf Tau \tau');
% zlabel('\bf Implied Vol');
% colormap(jet);
%
% forwardmoneyness=xxaxis/F;
%

% xxxx=zeros(1);
% yyyy=zeros(1);
% callpp=zeros(1);
% for i=1: length(callp)
%     for j=1:length(callp(1,:))
%         xxxx((j-1)*length(callp)+i)=strikeprice(i);
%         yyyy((j-1)*length(callp)+i)=tau(j);
%         if callp(i,j)==0
%             callpp((j-1)*length(callp)+i)=NaN;
%         else
%             callpp((j-1)*length(callp)+i)=callp(i,j);
%         end
%     end
% end

% figure
% surf(xxaxis/F, yyaxis, zaxis);
% xlabel('\bf Forward Moneyness');
% ylabel('\bf Tau \tau');
% zlabel('\bf Implied Vol');
% colormap(jet);

% figure
% surf(xxaxis/F, yyaxis, callhat);
% title('Call price surface using Implied Volatility surface(lambda=1)','
    fontsize',30)
% set(gca,'fontweigh','bold','fontsize',25)
% colormap(jet);
% xlabel('\bf Forward Moneyness','fontsize',25);
% ylabel('\bf Tau \tau','fontsize',25);
% zlabel('\bf Call price from Imvol','fontsize',25);
% hold on
% scatter3(xxxx/F,yyyy,callpp,100,'markerfacecolor','k','markeredgecolor','k')
% hold off
%
% xxxx=zeros(1);
% yyyy=zeros(1);
% zzzz=zeros(1);
% for i=1: length(callp)
%     for j=1:length(callp(1,:))
%         xxxx((j-1)*length(callp)+i)=strikeprice(i);
%         yyyy((j-1)*length(callp)+i)=tau(j);
%         if callp(i,j)==0
%             zzzz((j-1)*length(callp)+i)=NaN;
%         else
```

```
%             zzzz((j-1)*length(callp)+i)=blsimvol(i,j);
%         end
%     end
% end
%
% figure
% surf(xxaxis/F, yyaxis, zaxis);
% title('Implied Volatility surface(lambda=100)','fontsize',30)
% xlabel('\bf Forward Moneyness','fontsize',25);
% ylabel('\bf Tau \tau','fontsize',25);
% zlabel('\bf Implied Vol','fontsize',25);
% colormap(jet);
% hold on
% scatter3(xxxx/F,yyyy,zzzz,100,'markerfacecolor','k','markeredgecolor','k')
% set(gca,'fontweigh','bold','fontsize',25)
% hold off

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% figure
% bar3(strikeprice,yy-zzzz)
% set(gca,'fontweigh','bold','fontsize',25)
% ylabel('\bf Strikeprice','fontsize',25);
% xlabel('\bf Tau \tau','fontsize',25);
% zlabel('\bf Error','fontsize',25);
% title('Error graph(lambda=1)','fontsize',30)
```

```
function [B] = bsplinebase(data,x)
    xLeft = floor(data(1)-mean(diff(data)));
    ndx = length(data)+1;
    bdeg = 3;
    xx = [ xLeft; data ];
    dxx = diff(xx);
    dxmean = mean(dxx);
    t = zeros(1,ndx+2*bdeg);
    t(1:bdeg+1) = xLeft + dxmean*(-bdeg:0);
    t(bdeg+2:bdeg+ndx) = data;
    t(bdeg+ndx+1:ndx+2*bdeg) = t(bdeg+ndx)+dxmean*(1:(bdeg));
    B=zeros(length(t),bdeg+1);

    for i=1:length(t)-1
        if t(i)<=x && x<t(i+1)
            B(i,1)=1;
        else
            B(i,1)=0;
        end
    end
    B(end,1)=0;

    for k=2:bdeg+1
        for i=1:length(t)-k
            B(i,k)=(x-t(i))/(t(i+k-1)-t(i))*B(i,k-1)...
                +(t(i+k)-x)/(t(i+k)-t(i+1))*B(i+1,k-1);
        end
        for i=length(t)-k+1:length(t)
            B(i,k)=0;
        end
    end
    B=B(:,bdeg+1)';
end
```

```
function [c, ceq] = nonlcon3(a)
    global tau r s xx constbase

    lengthxx = length(xx);
    yhat=zeros(lengthxx,1);
    callhat=zeros(lengthxx,1);

    for i=1:lengthxx
        yhat(i,1)=constbase{i}*a;
        callhat(i,1)=bs_option_price(s,xx(i),r,yhat(i,1),tau(1),1);
    end

    y=callhat;

    D1 = diff(eye(lengthxx),1);
    D2 = diff(eye(lengthxx),2);

    c=D1*y;
    c=[c;-(D1*y)/(xx(2)-xx(1))-1];
    c=[c;-D2*y];

    ceq=[];
end
```

```
function [c, ceq] = nonlcon4(a)
    global xaxis yaxis temp5 s r

    lengthx=length(xaxis);
    lengthy=length(yaxis);

    zaxis=zeros(lengthx,lengthy);
    callaxis=zeros(lengthx,lengthy);

    for j=1:lengthy
        for i=1:lengthx
            zaxis(i,j)=temp5{i,j}*a;
            callaxis(i,j)=bs_option_price(s,xaxis(i),r,zaxis(i,j),yaxis(j),1);
        end
    end

    D1 = diff(eye(lengthx),1);
    D2 = diff(eye(lengthx),2);

    for j=1:lengthy
        if j==1
            c=D1*callaxis(:,j);
        else
            c=[c;D1*callaxis(:,j)]; %dcall/dK<0
        end
        c=[c;-(D1*callaxis(:,j))/(xaxis(2)-xaxis(1))-1]; % dcall/dK>-1
        c=[c;-D2*callaxis(:,j)]; % second derivative with K >0
        c=[c;-callaxis(:,j)]; %call value is positive
        c=[c;-zaxis(:,j)]; %imvol value is positive
        c=[c;zaxis(:,j)-1]; %imvol value is less than 1
    end

    D3=diff(eye(lengthy),1);

    for i=1:lengthx
        c=[c;-D3*callaxis(i,:)']; %dcall/dTau>0
```

```
37      end
38
39      ceq=[];
40  end
```

제 5 장

비선형방정식

대수학의 시작은 아마도 x에 관한 방정식 $f(x) = 0$를 푸는 것이 아니었을까? 만약 $f(x)$가 선형함수라면, 방정식 $f(x) = 0$의 해석해를 간단히 구할 수 있다. 그러나, 만약 $f(x)$가 비선형함수라면, 방정식 $f(x) = 0$의 해석해를 쉽게 구할 수 없는 경우가 많다. 금융공학에서도 비선형방정식을 풀어야하는 경우가 많다. 예를 들어, 내재변동성(implied volatility)을 구하거나 붓스트랩을 적용해서 제로커브(zero curve 또는 zero coupon yield curve)를 구하는 경우에는 비선형방정식을 풀어야 한다. 아주 특수한 경우를 제외하고는 비선형방정식의 해석해를 구할 수 없으므로, 수치해석기법을 적용해서 비선형방정식을 푸는 것이 일반적이다. 이 절에서는 비선형방정식을 푸는 수치해석기법을 설명하고자 한다.

제5.1절 직관적 방법

미지수가 하나인 비선형방정식을 수치적으로 풀기 위해서는 진짜 근이 얼마 정도인가를 가늠할 필요가 있다. 이러한 목적을 위해서는 진짜 근을 포함한다고 생각되는 구간에서 그래프를 그려보는 것이 좋다.

예제 5.1.1 다음 함수를 살펴보자.

$$f(x) = \left[1 + \frac{x}{3}\right]^3 - e^x + 0.01 \tag{1}$$

그래프를 이용해서 비선형방정식 $f(x) = 0$의 근사해를 구하기 위해서, 다음 MATLAB프로그램 GraphicalMethod101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: GraphicalMethod101.m
% Graphical method to sovle a nonlinear equation
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
gFun = inline('(1+x/3).^3 - exp(x)+0.01')
x = linspace(-0.5,0.5,201);
% Plotting
plot(x,gFun(x),'k-',x,x-x,'r','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','ylim',[-1/20 1/20])
grid on
grid minor
xlabel('\bf \it x','fontsize',12')
ylabel('\bf \it f(x)','fontsize',12,'rotation',0)
plt.axis([ -0.5, 0.5, -0.05, 0.05 ]);
saveas(gcf','GraphicalMethod101.jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB 프로그램 GraphicalMethod101.m을 실행하면, 구간 $[-0.50.5]$에서 함수 $f(x)$를 그린 그림 5.1.1을 출력한다. 그림 5.1.1에서 알 수 있듯이, 한 근사해는 $x_1 \approx 0.225$이고 다른 근사해는 $x_2 \approx -0.275$이다. ■

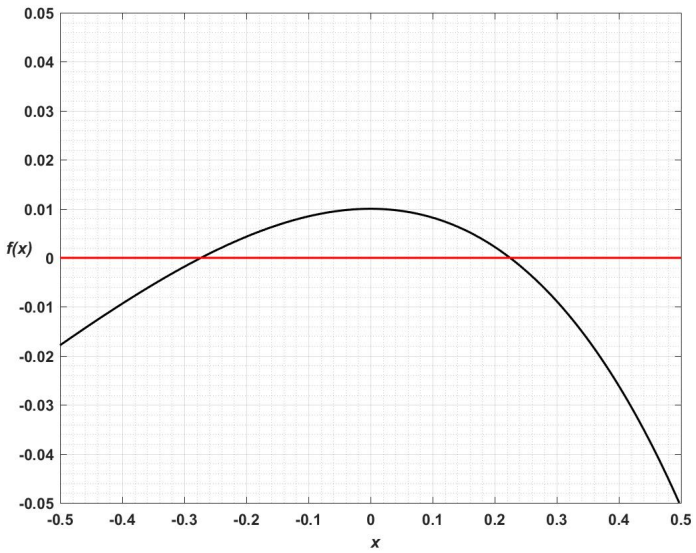


그림 5.1.1. 근사해를 찾는 그래프

예제 5.1.2 Python을 사용해서 예제 5.1.1을 다시 다루기 위해서, 다음 Python 프로그램 GraphicalMethod101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
```

```
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

gFun = lambda x: (1+x/3)**3 - np.exp(x)+0.01;
x = np.linspace(-0.5,0.5,201);

# Plotting
fig = plt.figure()
ax = fig.gca()
ax.set_xticks(np.arange(-0.5,0.5,0.1))
ax.set_yticks(np.arange(-0.05,0.05,0.01))
plt.plot(x,gFun(x),'k-',lw=2)
plt.plot(x,x-x,'r',lw=2)
plt.xlabel('x'); plt.ylabel('f(x)')
plt.grid()
plt.axis([ -0.5, 0.5, -0.05, 0.05 ]);
plt.show()
fig.savefig('GraphicalMethod101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.1.1의 결과와 같다. ■

미지수가 하나인 비선형방정식을 수치적으로 풀기 위해서 일양난수(uniform random number)를 이용해서 무작위찾기(random search)를 할 수 있다. 다음 예제를 살펴보자.

예제 5.1.3 다시 다음 함수를 생각해보자.

$$f(x) = \left[1 + \frac{x}{3}\right]^3 - e^x + 0.01 \tag{1}$$

무작위찾기를 적용해서 비선형방정식 $f(x) = 0$의 근사해를 구하기 위해서, 다음 MATLAB 프로그램 RandomSearch101.m을 실행해 보자.

```
% --------------------------------------------------------------------------
% Filename: RandomSearch101.m
% Random search to sovle a nonlinear equation
% Programmed by CBS
% --------------------------------------------------------------------------
clear all, close all, format long
gFun = inline('(1+x/3).^3 - exp(x)+0.01')
xMin = -0.5; xMax = 0.5;
rng('default')
for kk=1:5
    x = xMin + (xMax-xMin)*rand(10^6,1);
    FunVal = gFun(x);
    [ sol(kk),ii(kk) ] = min(abs(FunVal))
    xsol(kk) = x(ii(kk));
    f(kk) = gFun(xsol(kk));
```

```
end
[xsol' f']
% End of program
% --------------------------------------------------------------------
```

이 MATLAB프로그램 RandomSearch101.m을 실행하면, 다음과 같은 결과물을 출력한다.

```
   -0.272948354391520   0.000000007849110
    0.224732527871978   0.000000003594570
   -0.272948675112003  -0.000000013055524
   -0.272948680062897  -0.000000013378224
   -0.272948566093243  -0.000000005949655
```

이 결과에서 알 수 있듯이 근사해들은 $x_1 \approx 0.22473$ 과 $x_2 \approx -0.27295$ 이다. ■

예제 5.1.4 Python을 사용해서 예제 5.1.3을 다시 다루기 위해서, 다음 Python프로그램 RandomSearch101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import random

gFun = lambda x: (1+x/3)**3 - np.exp(x)+0.01;
xMin = -0.5; xMax = 0.5;
random.seed(9001)

ii = [0]*5
sol = [0.0]*5
xsol = [0.0]*5
f = [0.0]*5
for kk in range(0,5):
    x = xMin + (xMax-xMin)*np.random.rand(10**6,1);
    FunVal = gFun(x);
    ii[kk] = np.argmin(np.abs(FunVal));
    sol[kk] = np.min(np.abs(FunVal));
    xsol[kk] = np.asscalar(x[ii[kk]]);  # convert array to scalar
    f[kk] = gFun(xsol[kk]);
print(xsol)
print(f)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 5.1.3의 결과와 같다. ■

비선형방정식의 근이 존재하는 구간을 좁혀가는 블래키팅 (bracketing) 을 사용해서 근을 구할 수도 있다.

예제 5.1.5 예제 5.1.3에서 다룬 다음 함수를 다시 생각해보자.

$$f(x) = \left[1 + \frac{x}{3}\right]^3 - e^x + 0.01 \tag{1}$$

블래키팅법을 적용해서 비선형방정식 $f(x) = 0$의 근사해를 구하기 위해서, 다음 MATLAB 프로그램 BracketingSearch101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: BracketingSearch101.m
% Bracketing search to sovle a nonlinear equation
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
tic
gFun = inline('(1+x/3).^3 - exp(x)+0.01')
xMin = -0.5; xMax = 0.5;
nn = 10^5;
Delta = (xMax-xMin)/nn;
a = xMin; Sol = [ ];
kk = 0;
fa = feval(gFun,a);
for idum = 1:nn
    b = a + Delta;
    fb = feval(gFun,b);
    if sign(fa)*sign(fb) < 0
        kk = kk+1;
        Sol(kk,:) = [ a b ];
    end
    a = b; fa = fb;
end
Sol
toc
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 BracketingSearch101.m을 실행하면, 다음과 같은 결과물을 출력한다.

```
gFun =
     인라인 함수:
     gFun(x) = (1+x/3).^3 - exp(x)+0.01


Sol =
  -0.272949999999773  -0.272939999999773
   0.224730000000443   0.224740000000443

경과 시간은 29.617645초입니다.
```

이 결과에서 알 수 있듯이 한 근은 구간 $[0.22473, 0.22474]$에 존재하고 다른 한 근은 구간 $[-0.27295, -0.27294]$에 존재한다. 블래키팅법은 많은 시간을 필요로 한다. 이러한 구간을 얻는데 거의 30초가 걸렸다. 따라서, 블래키팅법은 초기근을 구하는 단계에서 사용하고, 좀 더 정밀한 답을 구하기 위해서는 Newton-Raphson법이나 시컨트법과 같은 다른 방법을 사용하는 것이 좋을 것이다. ■

예제 5.1.6 Python을 사용해서 예제 5.1.5을 다시 다루기 위해서, 다음 Python프로그램 BracketingSearch101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import timeit

tic=timeit.default_timer()
gFun = lambda x: (1+x/3)**3 - np.exp(x)+0.01;
xMin = -0.5; xMax = 0.5;
nn = 100000;
Delta = (xMax-xMin)/nn;
a = xMin;
SolDum = [];
kk = 0;
fa = gFun(a);
for idum in range(0,nn):
    b = a + Delta;
    fb = gFun(b);
    if np.sign(fa)*np.sign(fb) < 0:
        kk = kk+1;
        SolDum.append([ a, b ]);
    a = b; fa = fb;
Sol = np.array(SolDum)
print(Sol)
toc=timeit.default_timer()
timee = toc - tic   #elapsed time in seconds
print(timee)
# End of program
```

이 Python프로그램을 수행한 결과는 예제 5.1.5의 결과와 같다. 그러나 MATLAB을 사용한 예제 5.1.5에서는 29.6초 걸린 반면 Python을 이용한 이 예제에서는 0.7초가 걸렸다. 정말 빨라졌다. 그 이유는 무엇일까? ■

제5.2절 이분할법

방정식 $f(x)=0$의 근을 구하는 방법들 중에서 가장 직접적이고 쉬운 것은 이분할법(bisection method)일 것이다. 이분할법은 중간값정리(intermediate value theorem: IVT)를 바탕으로 한다.

명제 5.2.1: 중간값정리

만약 함수 $y=f(x)$가 폐구간 $[a,b]$에서 연속이고 또한 식 $f(a)f(b)<0$이 성립하면, 방정식 $f(x)=0$는 개구간 (a,b)에서 적어도 한 실근을 갖는다.

구간 $[a,b]$의 양 끝점에서 식 $f(a)>0$과 식 $f(b)<0$이 성립하면, 개구간 (a,b)에 속하는 적당한 한 점 c를 취해 $f(c)$를 계산한다. 만약 식 $f(c)>0$가 성립하면, 구간 (c,b)에 근이 있다. 반대로, 만약 $f(c)<0$이면, 근은 구간 (a,c)에 속한다. 또한, 식 $f(a)<0$와 식 $f(b)>0$가 성립하는 경우에도, 같은 논리를 적용할 수 있다. 따라서, 다음과 같은 알고리즘을 도출할 수 있다.

알고리즘 5.2.1: 이분할법

만약 함수 $y=f(x)$가 폐구간 $[a,b]$에서 연속이고 또한 $f(a)f(b)<0$이면, 다음과 같은 단계들을 거쳐서 방정식 $f(x)=0$의 근 x^*를 구한다.

(1단계) 초기값들로 $i=1$, $x_1=a$ 그리고 $x_2=b$라 놓는다.

(2단계) 점 $x_{i+2}=[x_{i+1}+x_i]/2$에 대해서, $f(x_{i+2})$를 계산한다. 만약 $f(x_i)f(x_{i+2})=0$이면, $x^*=x_{i+2}$라 놓고 알고리즘을 끝낸다. 만약 $f(x_i)f(x_{i+2})<0$이면, $x_{i+3}=x_i$라 놓고 제3단계로 넘어간다. 만약 $f(x_{i+1})f(x_{i+2})<0$이면, $x_{i+3}=x_{i+1}$라 놓고 제3단계로 넘어간다.

(3단계) 주어진 $\epsilon(>0)$에 대해서 식 $|x_{i+1}-x_i|<\epsilon$이 성립하면, $x^*=x_{i+2}$ 라 놓고 알고리즘을 끝낸다. 만약 식 $|x_{i+1}-x_i|\geq\epsilon$이 성립하면, i를 $i+1$으로 증가시킨 다음 제2단계로 돌아간다.

다음 예제는 MATLAB의 Symbolic Math Toolbox의 기능을 사용해서 알고리즘 5.2.1을 구현하는 것이다.

예제 5.2.1 알고리즘 5.2.1의 이분할법을 사용해서 방정식의 근을 구하기 위해서, 다음 MATLAB프로그램 BisectionMethod101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: BisectionMethod101.m
% Bisection Method 1
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
syms x xi xi1 xi2
f = x^2 - 2
epsil = 1.0e-6
xi = sym('1');  xi1= sym('2');
for ii = 1:1:40
    xi2 = (xi + xi1)/2;
    fxi = subs(f,xi); fxi1 = subs(f,xi1); fxi2 = subs(f,xi2);
    if double(fxi*fxi2) < 0, xi1 = xi2;
    else
        xi = xi2;
    end
    if abs(double(xi1-xi)) < epsil,
        break
    end
    IterNo(ii) = ii;
    xstar(ii) = double(xi);
end
xstar
% ii, xstar = double(xi)
plot(IterNo,xstar,'k*',IterNo,xstar,'b:','LineWidth',1.5)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf i','FontSize',12),
ylabel('{\bf x_{i}}','FontSize',12,'rotation',0)
saveas(gcf,'BisectionMethod101','jpg')
save('BisectionMethod101.txt','xstar','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 BisectionMethod101.m은 이분할법을 사용해서 방정식 $x^2 - 2 = 0$의 양근을 구하기 위한 것으로, 초기값들로 $x_1 = 1$과 $x_2 = 2$를 사용하였다.

이 MATLAB프로그램을 실행하면, $i = 19$, 즉 제19번째 반복을 중단하고 근 $x^* = 1.414213180541992$를 출력한다. 또한, 그림 5.2.1을 출력한다. 그림 5.2.1에서 알 수 있듯이, 이분할법은 수렴속도가 빠른 방법은 아니다. ■

예제 5.2.2 Python을 사용해서 예제 5.2.1을 다시 다루기 위해서, 다음 Python프로그램 BisectionMethod101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
```

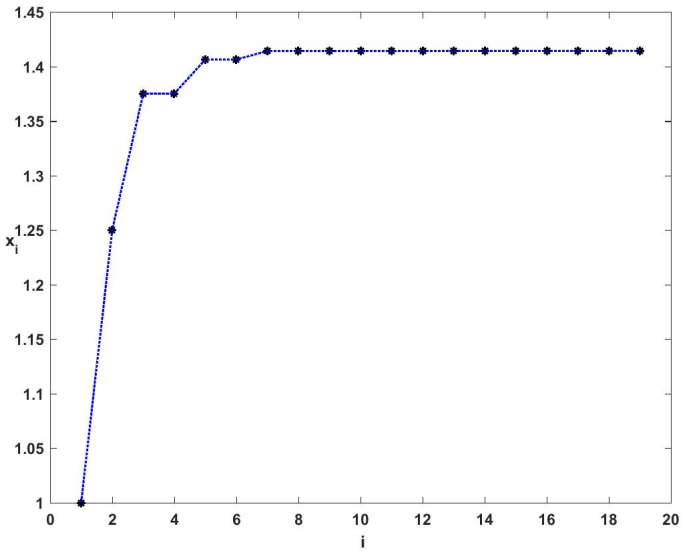


그림 5.2.1. 이분할법 I

```
5
6  import numpy as np
7  import matplotlib.pyplot as plt
8
9  f = lambda x: x**2 - 2
10 epsil = 1.0e-6
11 # xi = symbols('1');  xi1= symbols('2');
12 IterNo = [];
13 xstar = [];
14 xi = 1; xi1 =2;
15 for ii in range(0,40):
16     xi2 = (xi + xi1)/2;
17     fxi = f(xi); fxi1 = f(xi1); fxi2 = f(xi2);
18     if fxi*fxi2 < 0:
19         xi1 = xi2;
20     else:
21         xi = xi2;
22     if abs(xi1-xi) < epsil:
23         break
24     IterNo.append(ii+1)
25     xstar.append(xi)
26
27 # Plotting
28 print(xstar)
29 fig = plt.figure()
30 plt.plot(IterNo,xstar,'k*', lw=2)
31 plt.plot(IterNo,xstar,'b:', lw=2)
32 plt.xlabel('i'); plt.ylabel('x_{i}')
33 plt.axis([0, 20, 1, 1.45 ])
34 plt.show()
35 fig.savefig('BisectionMethod101Py.jpg')
36
37 # End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.2.1의 결과와 같다. ■

다음 예제는 MATLAB의 Symbolic Math Toolbox의 기능을 사용하지 않고 알고리즘 5.2.1을 구현하는 것이다.

예제 5.2.3 알고리즘 5.2.1의 이분할법을 사용해서 방정식의 근을 구하기 위해서, 다음 MATLAB프로그램 BisectionMethod102.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: BisectionMethod102.m
% Bisection Method 2
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
square2 = inline('z^2-2','z')
epsil = 1.0e-6;
x(1) = 1; ff(1) = square2(x(1));
x(2) = 2; ff(2) = square2(x(2));;
for ii = 2:1:40
    x(ii+1) = (x(ii) + x(ii-1))/2;
    ff(ii+1) = square2(x(ii+1));
    if ff(ii+1) == 0,
       xstar = x(ii+1);
       break
    end
    if sign(ff(ii)) == sign(ff(ii+1)),
        x(ii) = x(ii-1);
        end
    if abs(x(ii+1)-x(ii)) < epsil,
        xstar = x(ii+1);
        break
    end
end
ii, xstar
% Plotting
iidum = (1:1:ii);
plot(iidum,x(1:ii),'g-',iidum,x(1:ii),'k.','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\bf Solution \it x_{n}','fontsize',12)
axis( [ 0.5 ii+0.5 0.9 1.6 ])
saveas(gcf,'BisectionMethod102','jpg')
save('BisectionMethod102.txt','xstar','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 BisectionMethod2.m도 이분할법을 사용해서 방정식 $x^2 - 2 = 0$의 양근을 구하기 위한 것으로, 초기값들로 $x_1 = 1$과 $x_2 = 2$를 사용하였다.

이 MATLAB프로그램을 실행하면, $i = 21$, 즉 제21번째 반복을 중단하고 근 $x^* = 1.414214134216309$를 출력한다. 또한, 그림 5.2.2을 출력한다. 그림 5.2.2에서 확인할 수 있듯이, 이분할법은 수렴속도가 빠른 방법은 아니다. ■

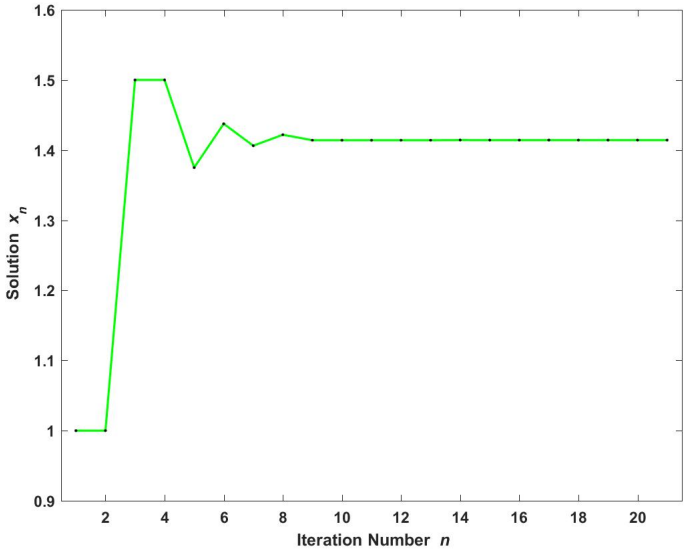


그림 5.2.2. 이분할법 II

예제 5.2.4 Python을 사용해서 예제 5.2.3을 다시 다루기 위해서, 다음 Python프로그램 BisectionMethod102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

square2 = lambda z: z**2 -2
epsil = 1.0e-6;
x = []; ff = [];
x.append(1); ff.append(square2(x[0]));
x.append(2); ff.append(square2(x[1]));

for ii in range(1,40):
    x.append((x[ii] + x[ii-1])/2)
    ff.append(square2(x[ii+1]));
    if ff[ii+1] == 0:
       xstar = x[ii+1];
       break
    if sign(ff[ii]) == sign(ff[ii+1]):
        x[ii] = x[ii-1];
    if abs(x[ii+1]-x[ii]) < epsil:
        xstar = x[ii+1];
        break

# Plotting
print(ii,xstar)
iidum = np.arange(0,ii+2)
fig = plt.figure()
plt.plot(iidum,x,'g-', lw=2)
plt.plot(iidum,x,'k.', lw=2)
plt.xlabel('i'); plt.ylabel('x_{i}')
```

```
plt.axis([ 0.5, ii+0.5, 0.9, 1.6 ])
plt.show()
fig.savefig('BisectionMethod102Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.2.3의 결과와 같다. ■

IEEE 754에서 배정도수의 가수(mantissa)는 52비트로 표현된다. 따라서, 함수 $f(x)$가 어떠한 형태이든 이분할법알고리즘은 제52번째 단계에서는 수렴해야만 한다. 이 경우에 1비트의 정밀도로 답을 얻는다.

제5.3절 고정점법

단순반복법(simple iteration method), 고정반복법(fixed point iteration method), 단순범함수반복법(simple functional iteration method)이라고도 불리우는 고정점법(fixed point method)은 비선형방정식을 푸는 가장 일반적인 기법으로서 오래된 역사를 가지고 있다. 이에 대해서는 **IM&F10** [5]의 제3.13절을 참조하라. 고정점법은 점화식(recursive formula)과 밀접한 관계가 있다. 본저자는 고등학교 때 점화식에 관한 원고를 썼고, 그 원고를 대학교 1학년 때 출간하였다. 세월이 흘러서 이제는 그 책의 원본을 가지고 있지 않지만 지인에게 원본을 빌려서 그 내용을 **SAS4TSA4** [1]의 부록에 수록하였으니, 관심있는 독자는 참조하기 바란다.

고정점법을 적용해서 $\sqrt{2}$를 계산하는 문제를 생각해보자. 다음 방정식의 양근이 $\sqrt{2}$임이 명백하다.

$$x^2 - 2 = 0 \tag{5.3.1}$$

어떤 실수 a에 대해서, 방정식 (5.3.1)을 다음과 같이 쓸 수 있다.

$$x[x + a] = 2 + ax \tag{5.3.2}$$

방정식 (5.3.2)를 다음과 같이 쓸 수 있다.

$$x = \frac{2 + ax}{x + a} \tag{5.3.3}$$

방정식 (5.3.3)에 해당하는 고정점법은 다음 점화식을 이용하는 것이다.

$$x_{n+1} = \frac{2 + ax_n}{x_n + 1}, \quad (n = 1, 2, \cdots) \tag{5.3.4}$$

만약 점화식 (5.3.4)에 의해서 생성된 수열 $\{x_n\}$이 수렴하면, 이 극한값이 방정식 (5.3.1)의 근이다.

예제 5.3.1 다양한 a에 대해서 고정점법을 적용해 방정식 $x^2 - 2 = 0$를 풀기 위해서, 다음 MATLAB프로그램 FixedPointMethod101.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: FixedPointMethod101.m
%  Recursive method to solve a nonlinear equation
%  Programmed by CBS
% ------------------------------------------------------------------------
close all, clear all
for dumk = 1:1:5
    a(dumk) = dumk-3;
    x(1,dumk) = 1.6;
    for n = 1:19
        x(n+1,dumk) = a(dumk) + (2-a(dumk)*a(dumk))/(x(n,dumk)+a(dumk));
    end
end
% Plotting
ii = (1:1:20);
plot(ii,x(:,1),'b',ii,x(:,2),'r-.',ii,x(:,3),'g--',ii,x(:,4),'k', ...
     ii,x(:,5),'y-.','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
legend('a=-2','a=-1','a=0','a=1','a=2','location','SE')
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\bf Solution \it x_{n}','fontsize',12)
axis( [ 1 20 -5 4 ])
saveas(gcf,'FixedPointMethod101','jpg')
save('FixedPointMethod101.txt','x','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 FixedPointMethod101.m을 실행하면, 다양한 a에 대해서 고정점법이 적용된 결과인 그림 5.3.1이 그려진다. 그림 5.3.1에서 알 수 있듯이, 동일한 초기값 $x_1 = 1.6$을 사용하여도 상수 a에 따라 극한형태가 달라진다. 만약 $a = 2$ 또는 $a = 1$이면, $\{x_n\}$은 방정식 $x^2 - 2 = 0$의 근 $\sqrt{2}$에 수렴한다. 만약 $a = -2$ 또는 $a = -1$이면, $\{x_n\}$은 방정식 $x^2 - 2 = 0$의 다른 근 $-\sqrt{2}$에 수렴한다. 반면에, 만약 $a = 0$이면, $\{x_n\}$은 진동한다. 이 예제에서도 알 수 있듯이, 고정점법을 사용해서 비선형방정식을 풀기위해서는 해당하는 점화식을 잘 구축해야 한다. ■

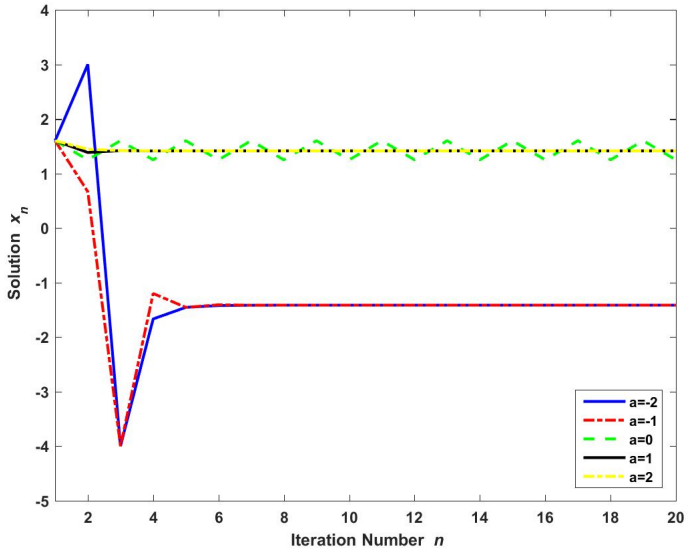


그림 5.3.1. 다양한 모수에 따른 고정점법

예제 5.3.2 Python을 사용해서 예제 5.3.1을 다시 다루기 위해서, 다음 Python프로그램 FixedPointMethod101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.zeros((20,5))
a = [0]*5
for dumk in range(0,5):
    a[dumk] = dumk-2;
    x[0,dumk] = 1.6;
    for n in range(0,19):
        x[n+1,dumk] = a[dumk] + (2-a[dumk]*a[dumk])/(x[n,dumk]+a[dumk]);

# Plotting
ii = np.arange(0,20)
fig = plt.figure()
plt.plot(ii,x[:,0],'b',lw=2, label='a=-2')
plt.plot(ii,x[:,1],'r-.',lw=2, label='a=-1')
plt.plot(ii,x[:,2],'g--',lw=2, label='a=0')
plt.plot(ii,x[:,4],'k',lw=2, label='a=1')
plt.plot(ii,x[:,0],'y-.',lw=2, label='a=2')
plt.xlabel('\bf Iteration Number \it n');
plt.ylabel('\bf Iteration Number \it n')
plt.axis([ 0, 19, -5, 4 ])
plt.legend(loc='lower right', numpoints=1)
plt.show()
fig.savefig('FixedPointMethod101Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.3.1의 결과와 같다. ■

방정식 $f(x)=0$의 근 α를 고정점법으로 풀기 위해서, 이 방정식을 $x=g(x)$로 표현한다고 하자. 다음 식이 성립한다.

$$\alpha = g(\alpha) \tag{5.3.5}$$

이에 해당하는 점화식은 다음과 같다.

$$x_{n+1} = g(x_n), \quad (n=1,2,\cdots) \tag{5.3.6}$$

즉, 방정식 $f(x)=0$를 푸는 문제를 이와 동등한 방정식 $x=g(x)$의 고정점(fixed point)을 찾아내는 문제로 바꾸는 것이다. 함수 $g(x)$는 다양한 형태를 취할 수 있다. 예를 들어, 가장 간단한 방법으로는 함수 $g(x)=x+f(x)$를 고려할 수 있고, 적당한 상수 γ에 대한 함수 $g(x)=x+\gamma f(x)$를 사용할 수도 있다. 그러나, 함수 $f(x)$의 특성을 나타나는 특정한 값 x^*에 주목해서, 함수 $f(x)$를 변형해서 방정식 $x=g(x)$를 만든다. 물론 어떤 x^*를 선택하느냐에 따라 $g(x)$가 달라진다.

동일한 점화식을 사용해도 초기값에 따라 극한값과 수렴속도가 달라진다. 예를 들어, 다음 방정식을 고정점법으로 풀어보자.

$$x^3 - 3x^2 - x + 3 = 0 \tag{5.3.7}$$

이 방정식의 근들이 $-1, 1, 3$임을 쉽게 알 수 있다. 방정식 (5.3.7)을 다음과 같이 쓸 수 있다.

$$x = \frac{-3}{x^2 - 3x - 1} \tag{5.3.8}$$

따라서, 다음 점화식을 사용해서 방정식 (5.3.7)의 근을 구할 수 있다.

$$x_{n+1} = \frac{-3}{x_n^2 - 3x_n - 1}, \quad (n=1,2,\cdots) \tag{5.3.9}$$

초기값 x_1을 어떻게 선택하느냐에 따라 점화식 (5.3.9)를 만족하는 수열 $\{x_n\}$의 극한값이 달라진다. 이러한 현상을 설명하기 위해서, 우선 다음 예제를 살펴보자.

예제 5.3.3 다양한 초기값 x_1에 대한 고정점법을 적용해서 방정식 $x^3 - 3x^2 - x + 3 = 0$을 풀기 위해, 우선 다음 MATLAB프로그램 FixedPointMethod102pix.m를 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: FixedPointMethod102pix.m
%  Figure for Fixed Point Mehtod
%  Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
ftn = @(x) -3./(x.^2 - 3*x -1)
xx = -2:0.05:5;
hold on
plot(xx,ftn(xx),'b','LineWidth',1.5);
plot(xx,xx,'r--','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf y = g(x)','\bf y = x','location','NW')
plot([-2 5], [0 0], 'k-','LineWidth',1.2);
alpha1 = (3-sqrt(13))/2; alpha2 = (3+sqrt(13))/2;
xlabel('\bf \it x','fontsize',12)
ylabel('\bf \it y','fontsize',12,'rotation',0)
text(alpha1-0.4,-0.12,'\bf \alpha_{1}')
text(alpha2+0.05,-0.12,'\bf \alpha_{2}')
axis([ -2 5 -2 5 ])
hold off
saveas(gcf,'FixedPointMethod102pix','jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 FixedPointMethod102pix.m를 실행하면, 함수 $g(x) = \frac{-3}{x^2-3x-1}$ 과 함수 $y = x$를 그린 그림 5.3.2이 그려진다. 여기서 $\alpha_1 = \frac{3-\sqrt{13}}{2}$ 이고, $\alpha_2 = \frac{3+\sqrt{13}}{2}$ 이다. ■

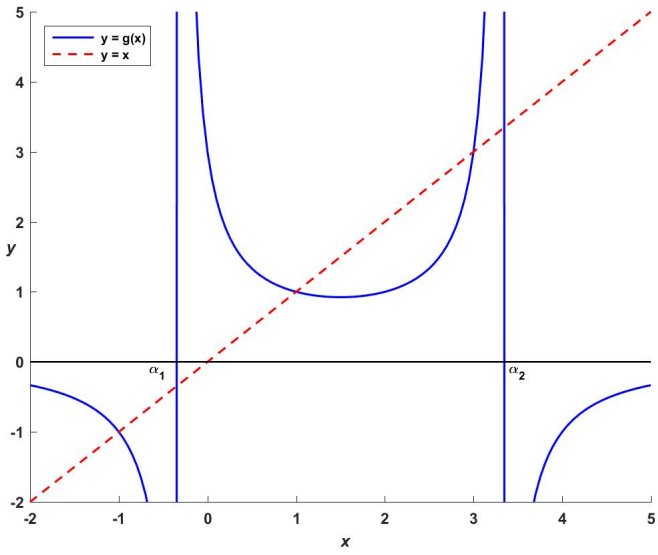


그림 5.3.2. 함수 $y = g(x)$

예제 5.3.4 다양한 초기값 x_1에 대한 고정점법을 적용해서 방정식 $x^3 - 3x^2 - x + 3 = 0$을 풀기 위해, 다음 MATLAB프로그램 FixedPointMethod102.m를 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: FixedPointMethod102.m
%  Initial values for Recursive method to solve a nonlinear equation
%  Programmed by CBS
% ------------------------------------------------------------------------
close all, clear all, format long
maxN = 40;
x(1,:) = [ -2 -0.5 0.5 2 4 ];
for dumk = 1:1:5
    for n = 1:maxN-1
        x(n+1,dumk) = -3/( -1+x(n,dumk)*(-3+x(n,dumk)));
    end
end
% Plotting
ii = (1:1:maxN);
plot(ii,x(:,1),'b',ii,x(:,2),'r-.',ii,x(:,3),'g--',ii,x(:,4),'y', ...
      ii,x(:,5),'k-.','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
legend('x_{1} =-2','x_{1} =-0.5','x_{1} =0.5','x_{1} =2', ...
       'x_{1} =4','location','SE')
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\bf Solution \it x_{n}','fontsize',12)
axis( [ 0, maxN, -35 30 ] )
saveas(gcf,'FixedPointMethod102','jpg')
save('FixedPointMethod102.txt','x','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 FixedPointMethod102.m를 실행하면, 다양한 초기값 x_1에 대해서 고정점법이 적용된 결과인 그림 5.3.3이 그려진다. 그림 5.3.3에서 알 수 있듯이, 동일한 점화식이라 하더라도 초기값 x_1에 따라 극한형태가 달라진다. 식 (5.3.8)의 우변의 분모에 해당하는 방정식 $x^2-3x-1=0$의 작은 근이 $\alpha_1=[3-\sqrt{13}]/2$이고 큰 근이 $\alpha_2=[3+\sqrt{13}]/2$임을 상기하라. 만약 초기값이 $x_1=-2$이면, $\{x_n\}$은 진동발산한다. 따라서 초기값이 가장 작은 근 -1보다 작으면, $\{x_n\}$은 발산함을 알 수 있다. 초기값이 $x_1=-0.5$이면, $\{x_n\}$은 진동발산한다. 따라서 초기값이 α_1보다 작으면, $\{x_n\}$은 발산함을 알 수 있다. 초기값이 $x_1=0.5$이면, $\{x_n\}$은 가운데 근 1에 수렴한다. 따라서, 초기값이 구간 $(\alpha_1,1]$에 속하면, $\{x_n\}$ 은 가운데 근 1에 수렴함을 알 수 있다. 초기값이 $x_1=2$이면, $\{x_n\}$은 가운데 근 1에 수렴한다. 따라서, 초기값이 구간 $(1,3)$에 속하면, $\{x_n\}$은 가운데 근 1에 수렴함을 알 수 있다. 초기값이 $x_1=3$이면, $\{x_n\}$은 큰 근 3에 수렴한다. 초기값이 $x_1=3.1$이면, $\{x_n\}$은 진동한다. 따라서, 초기값이 구간 $(3,\alpha_2)$에 속하면, $\{x_n\}$은 발산함을 알 수 있다. 초기값이 $x_1=4$이면, $\{x_n\}$은 작은 근 -1에 수렴한다. 따라서, 초기값이 구간 (α_2,∞)에 속하면, $\{x_n\}$은 작은 근 -1에 수렴함을 알 수 있다. 그림 5.3.2에서 점화식 (5.3.9)를 적용해서 이 결과를 확인할 수 있다. ■

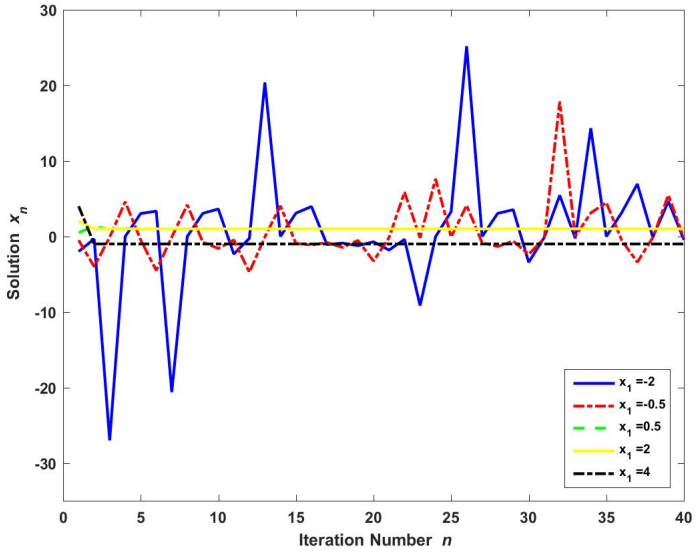


그림 5.3.3. 다양한 초기값에 따른 고정점법

예제 5.3.5 Python을 사용해서 예제 5.3.4을 다시 다루기 위해서, 다음 Python프로그램 FixedPointMethod102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

maxN = 40;
x = np.zeros((maxN,5))
x[0,:] = [ -2, -0.5, 0.5, 2, 4 ];
for dumk in range(0,5):
    for n in range(0,maxN-1):
        x[n+1,dumk] = -3/( -1+x[n,dumk]*(-3+x[n,dumk]));

# Plotting
ii = np.arange(0,maxN)
fig = plt.figure()
plt.plot(ii,x[:,0],'b',lw=2, label='x_{0}=-2')
plt.plot(ii,x[:,1],'r-.',lw=2, label='x_{0}=-0.5')
plt.plot(ii,x[:,2],'g--',lw=2, label='x_{0}=0.5')
plt.plot(ii,x[:,4],'y',lw=2, label='x_{0}=2')
plt.plot(ii,x[:,0],'k-.',lw=2, label='x_{0}=4')
plt.xlabel('\bf Iteration Number \it n');
plt.ylabel('\bf Iteration Number \it n')
plt.axis([ 0, maxN-1, -35, 30 ])
plt.legend(loc='lower right', numpoints=1)
plt.show()
fig.savefig('FixedPointMethod102Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.3.4의 결과와 같다. ■

고정점법을 적용해서 근을 구하기 위해서는 점화식 (5.3.6)에서 발생된 수열 $\{x_n\}$이 수렴해야만 한다. 고정점법의 수렴에 관한 다음과 같은 명제가 알려져 있다. 이에 대한 자세한 내용은 Lange [36]를 참조하라.

명제 5.3.1: Lipschitz조건

폐구간 I에서 정의되는 함수 $g(x)$가 다음 조건들을 만족한다고 가정하자.

(a) 각 $x(\in I)$에 대해 식 $g(x) \in I$이 성립한다.

(b) 임의의 점들 $x(\in I)$와 $y(\in I)$에 대해 다음 Lipschitz조건이 성립한다.

$$|g(y) - g(x)| \leq \lambda |y - x|$$

만약 함수 $g(x)$에 대한 Lipschitz상수 λ가 구간 $[0, 1)$에 속하면, 임의의 점 $x_0(\in I)$을 초기값으로 하는 점화식 $x_{n+1} = g(x_n)$에 의해서 생성된 수열 $\{x_n\}$은 식 $\alpha = g(\alpha)$를 만족시키는 $\alpha(\in I)$에 일의적으로 수렴한다.

명제 5.3.1에서 조건 (a)는 필요조건이고 조건 (b)는 충분조건의 하나이다. 만약 함수 $g(x)$가 구간 I에서 미분가능이라면, 다음과 같이 명제 5.3.1를 쉽게 증명할 수 있다. 식 (5.3.5)와 식 (5.3.6)에서 알 수 있듯이, 다음 점화식이 성립한다.

$$x_{n+1} - \alpha = g(x_n) - g(\alpha), \quad (n = 0, 1, \cdots) \tag{5.3.10}$$

식 (5.3.10)에서 알 수 있듯이, 다음 근사점화식이 성립한다.

$$x_{n+1} - \alpha \approx g'(\alpha)[x_n - \alpha], \quad (n = 0, 1, \cdots) \tag{5.3.11}$$

즉, 다음 근사식들이 성립한다.

$$|x_n - \alpha| \approx |g'(\alpha)|^n |x_0 - \alpha|, \quad (n = 0, 1, \cdots) \tag{5.3.12}$$

따라서, 식 $\lim_{n\to\infty} x_n = \alpha$ 이 성립하기 위한 충분조건은 다음과 같다.

$$|g'(\alpha)| < 1 \tag{5.3.13}$$

식 (5.3.12) 에서 알 수 있듯이, 오차 $e_n \doteq |x_n - \alpha|$ 는 다음 식을 만족한다.

$$e_{n+1} \approx \left|g'(x_n)\right| e_n \tag{5.3.14}$$

조건 (5.3.13) 이 만족되면, e_{n+1} 은 e_n 보다는 작아지지만 e_n 과 같은 차원이다. 즉, 고정점법의 수렴은 1차이다. 특히 $|g'(\alpha)|$ 가 0에 가까울수록 수렴은 빨라지므로, 근의 근방에서 미분계수가 0에 가까운 함수 $g(x)$ 를 선택하는 것이 좋다.

예제 5.3.6 예제 5.3.4를 다시 살펴보자. 다음 식이 성립한다.

$$g(x) = \frac{-3}{x^2 - 3x - 1} \tag{1}$$

다음 식이 성립한다.

$$g'(x) = \frac{3[2x - 3]}{x^2 - 3x - 1} \tag{2}$$

따라서 다음 식들이 성립한다.

$$g'(1) = -\frac{1}{3}, \;\; g'(-1) = -\frac{5}{3}, \;\; g'(3) = 0 \tag{3}$$

근 $\alpha = 1$ 는 조건 $|g'(\alpha)| < 1$ 을 만족한다. 그러나, 근 $\alpha = -1$ 은 이 조건을 만족하지 못한다. 그럼에도 불구하고, 초기값이 $x_1 = 4$ 이면 $\{x_n\}$ 은 근 -1 에 수렴한다. 그 이유는 다음과 같다.

$$x_2 = g(4) = -1, \;\; x_n = g(-1) = -1, \;\; (n = 3, 4, \cdots) \tag{4}$$

따라서, 조건 $|g'(\alpha)| < 1$ 은 $\{x_n\}$ 이 근으로 수렴하기 위한 충분조건이나 필요조건은 아니라는 것을 확인할 수 있다. ■

식 $\lim_{n\to\infty} x_n = \alpha$ 이 성립하기 위한 충분조건을 미분을 사용하지 않고 나타낸 것이 다음 명제이다.

명제 5.3.2

함수 $y = g(x)$가 x의 구간 J에서 다음 조건들을 만족한다고 가정하자.

(a) 함수 $y = g(x)$와 함수 $y = x$는 구간 J에서 유일한 공유점 $(\alpha, g(\alpha))$를 갖는다. 즉, α는 유한값으로서 방정식 $\alpha = g(\alpha)$를 만족한다.

(b) 각 $x \in J$에 대해, 식 $g(x) \in J$가 성립한다.

(c) 각 $x \in J$에 대해, 다음 부등식이 성립한다.

$$|g(x) - \alpha| < |x - \alpha|$$

이러한 조건 하에서 임의의 점 $x_0 (\in J)$를 초기값으로 하고 점화식 $x_{n+1} = g(x_n)$에 의해서 생성된 수열 $\{x_n\}$은 $\alpha (\in J)$에 수렴한다.

증명. 각 $n (= 0, 1, \cdots)$에 대해서 다음 식들이 성립한다.

$$|x_{n+1} - \alpha| = |g(x_n) - \alpha| < |x_n - \alpha| \tag{1}$$

식 (1)에서 알 수 있듯이, 수열 $\{\, |x_n - \alpha| \,\}$은 어떤 값으로 수렴한다. 만약 이 수열이 수렴하는 값이 $\beta (> 0)$라면, 수열 $\{x_n\}$은 $\alpha + \beta$ 또는 $\alpha - \beta$에 수렴한다. 즉, 다음 식들이 성립한다.

$$\alpha - \beta = g(\alpha - \beta), \quad \alpha + \beta = g(\alpha + \beta) \tag{2}$$

이는 가정에 모순된다. 즉, β는 0이어야 한다. 따라서, 구간 J에 속하는 임의의 점 x_0에서 출발하는 수열 $\{x_n\}$은 α에 수렴한다. ■

만약 함수 $y = g(x)$의 도함수가 존재한다면, 명제 5.3.2는 식 $|g'(\alpha)| < 1$이 성립함을 의미한다. 즉, 함수 $y = g(x)$가 직선 $y = x$를 '위에서 아래로' 지나가는 것을 의미한다. 만약 함수 $y = g(x)$가 직선 $y = x$를 '아래에서 위로' 지나가면, 수열 $\{x_n\}$은 그 점 (α, α)로 수렴하지 않는다. 명제 5.3.1에서는 식 $\lambda < 1$이 성립한다고 가정하고 있지만, 명제 5.3.2에서는 조건 $\lambda < 1$이 성립한다고 가정하지는 않는다. 또한, 명제 5.3.1에서 구간 I는 폐구간이지만, 명제 5.3.2에서 구간 J는 반드시 폐구간일 필요는 없다. 반면에, 명제 5.3.1에서는 근 α가 존재한다는 가정을 하지 않지만, 명제 5.3.2에서는 근 α가 일의적으로 존재한다는 가정을 하고

있다. **SAS4TSA4** [1] 의 부록에서는 점화식 (5.3.6) 에 의해서 발생한 수열 $\{x_n\}$ 의 수렴성을 좀 더 직관적으로 다루고 있다.

예제 5.3.7 다음 함수를 살펴보자.

$$f(x) = x^2 - 3x + 2 \tag{1}$$

방정식 $f(x) = 0$ 의 두 근은 $\alpha_1 = 1$ 과 $\alpha_2 = 2$ 이다. 지금부터는 고정점법을 적용해서 근을 구하기로 하자.

첫째, 다음과 같은 점화식을 살펴보자.

$$x_{n+1} = g_1(x_n) \doteq x_n + f(x_n) = x_n^2 - 2x_n + 2, \quad x_0 = 1.8 \tag{2}$$

함수 $g_1(x)$ 의 도함수는 $g_1'(x) = 2[x-1]$ 이므로, 점 $x = \alpha_1$ 에서 식 (5.3.13) 을 만족한다. 즉, 명제 5.3.1의 조건이 만족된다. 또한, 초기값 $x_0 = 1.8$ 에서 도함수값은 $g_1'(1.8) = 1.6$ 이므로 명제 5.3.1의 조건 $\lambda < 1$ 은 만족되지 않지만, 점 $x = \alpha_1$ 에서 주변에서 명제 5.3.2의 조건은 만족된다. 따라서, 점화식 (2) 에 의해서 발생한 수열 $\{x_n\}$ 은 근 α_1 으로 수렴한다. 마차가지로, 개구간 $J = (0, 2)$ 에서 명제 5.3.2의 조건은 만족되므로, 초기값을 $x_0 (\in (0, 2))$ 로 하면서 점화식 (2) 에 의해서 발생한 수열 $\{x_n\}$ 은 근 α_1 으로 수렴한다. 또한, $g_1'(1) = 0$ 이므로 수렴속도가 빠르다.

둘째, 다음과 같은 점화식을 살펴보자.

$$x_{n+1} = g_2(x_n) \doteq x_n - f(x_n) = -x_n^2 + 4x_n - 2, \quad x_0 = 1.8 \tag{3}$$

함수 $g_2(x)$ 의 도함수는 $g_2'(x) = -2[x-2]$ 이므로, 점 $x = \alpha_2$ 에서 명제 5.3.1의 조건이 만족된다. 또한, 초기값 $x_0 = 1.8$ 에서 도함수값은 $g_2'(1.8) = 0.4$ 이므로, 명제 5.3.1의 충분조건 $\lambda < 1$ 이 만족되며 또한 명제 5.3.2의 조건도 만족된다. 따라서, 점화식 (3) 에 의해서 발생한 수열 $\{x_n\}$ 은 근 α_2 로 수렴한다. 개구간 $J = (1, 3)$ 에서 명제 5.3.2의 조건이 만족되므로, 초기값을 $x_0 (\in (1, 3))$ 로 하는 점화식 (3) 에 의해서 발생한 수열 $\{x_n\}$ 은 근 α_2 로 수렴한다. 또한, 식 $g_2'(2) = 0$ 이 성립하므로, 이 점화식의 수렴속도는 빠르다.

셋째, 다음과 같은 점화식을 살펴보자.

$$x_{n+1} = g_3(x_n) \doteq \frac{x_n^2 + 2}{3}, \quad x_0 = 1.8 \tag{4}$$

함수 $g_3(x)$의 도함수는 $g_3'(x) = \frac{2}{3}x$이므로, 점 $x = \alpha_1$에서 명제 5.3.1의 조건이 만족된다. 또한, 초기값 $x_0 = 1.8$에서 도함수값은 $g_3'(1.8) = 1.2$이므로 명제 5.3.1의 조건 $\lambda < 1$은 만족되지 않으나, 개구간 $J_1 = (-4, 2)$에서 명제 5.3.2의 조건 (b)가 만족된다. 그러나, 개구간 $J = (-2, 2)$에서 명제 5.3.2의 조건이 모두 만족된다. 따라서, 초기값을 $x_0(\in J)$로 하는 점화식 (4)에 의해서 발생한 수열 $\{x_n\}$은 근 α_1으로 수렴한다. 그러나, $g_3'(1) = 0.667$이므로 수렴속도가 늦다.

넷째, 다음과 같은 점화식을 살펴보자.

$$x_{n+1} = g_4(x_n) \doteq \sqrt{3x_n - 2}, \quad x_0 = 1.8 \tag{5}$$

함수 $g_4(x)$의 도함수는 $g_4'(x) = \frac{3}{2\sqrt{3x-2}}$이므로, 점 $x = \alpha_2$에서 $g_4'(\alpha_2) = 0.75$이다. 즉, 명제 5.3.1의 조건 $\lambda < 1$은 만족된다. 따라서, 점화식 (5)에 의해서 발생된 수열 $\{x_n\}$은 근 α_2로 수렴한다. 마찬가지로 초기값을 $x_0(> 1)$로 하는 점화식 (5)에 의해서 발생된 수열 $\{x_n\}$은 근 α_2으로 수렴한다. 그러나, 수렴속도가 늦다.

지금까지 내용을 확인하기 위해서, 다음 MATLAB프로그램 FixedPointMethod104.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: FixedPointMethod104.m
%  Convergence of fixed point methods
%  Programmed by CBS
% ------------------------------------------------------------------------
close all, clear all, format long
maxN = 10;
x1(1) = 1.8;
for nn=1:maxN
    x1(nn+1) = x1(nn)^2 - 2*x1(nn) +2;
end
x2(1) = 1.8;
for nn=1:maxN
    x2(nn+1) = -x2(nn)^2 + 4*x2(nn) - 2;
end
x3(1) = 1.8;
for nn=1:maxN
    x3(nn+1) = (x3(nn)^2 + 2)/3;
end
x4(1) = 1.8;
for nn=1:maxN
    x4(nn+1) = sqrt(3*x4(nn) - 2);
end
x5(1) = 1.8;
for nn=1:maxN
    x5(nn+1) = 1/101*(100*x5(nn)+3-2/x5(nn));
end
% Plotting
xd = linspace(0,3,301);
```

```
nx = 1:2*maxN
ix = floor(nx/2)+1
ny = 2:2*maxN+1
iy = floor(ny/2)+1
subplot(2,2,1)
plot(xd,xd,'g-',xd,xd.^2-2*xd+2,'g-',x1(ix),x1(iy),'k-', ...
     x1(iy),x1(iy),'r.','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[ 0.9,2.1 ])
xlabel('\bf \it x_{n-1}','fontsize',12)
ylabel('\bf \it x_{n}','fontsize',12,'rotation',0)
subplot(2,2,2)
plot(xd,xd,'g-',xd,-xd.^2+4*xd-2,'g-',x2(ix),x2(iy),'k-', ...
     x2(iy),x2(iy),'r.','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[ 1.75, 2.05 ])
xlabel('\bf \it x_{n-1}','fontsize',12)
ylabel('\bf \it x_{n}','fontsize',12,'rotation',0)
subplot(2,2,3)
plot(xd,xd,'g-',xd,(xd.^2+2)/3,'g-',x3(ix),x3(iy),'k-', ...
     x3(iy),x3(iy),'r.','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[ 0.9 1.9 ])
xlabel('\bf \it x_{n-1}','fontsize',12)
ylabel('\bf \it x_{n}','fontsize',12,'rotation',0)
subplot(2,2,4)
plot(xd,xd,'g-',xd,sqrt(3*xd-2),'g-',x4(ix),x4(iy),'k-', ...
     x4(iy),x4(iy),'r.','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[ 1.79 2.01 ])
xlabel('\bf \it x_{n-1}','fontsize',12)
ylabel('\bf \it x_{n}','fontsize',12,'rotation',0)
saveas(gcf,'FixedPointMethod104','jpg')
save('FixedPointMethod102.txt','x1','x2','x3','x4','x5','-ascii')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 FixedPointMethod104.m을 실행하면, 이 점화식들의 그래프들을 그린 그림 5.3.4이 그려진다. 그림 5.3.4의 좌측상단 그래프는 점화식 (2)에 의한 수열을, 우측상단 그래프는 점화식 (3)에 의한 수열을, 좌측하단 그래프는 점화식 (4)에 의한 수열을, 그리고 우측하단 그래프는 점화식 (5)에 의한 수열을 그린 것이다. 이 그래프들에서 알 수 있듯이, 점화식 (2), 점화식 (3), 점화식 (5), 그리고 점화식 (4) 순으로 속도가 느리다. ■

예제 5.3.8 Python을 사용해서 예제 5.3.7을 다시 다루기 위해서, 다음 Python프로그램 FixedPointMethod104.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

maxN = 10;
x1 = [0.0]*(maxN+1)
```

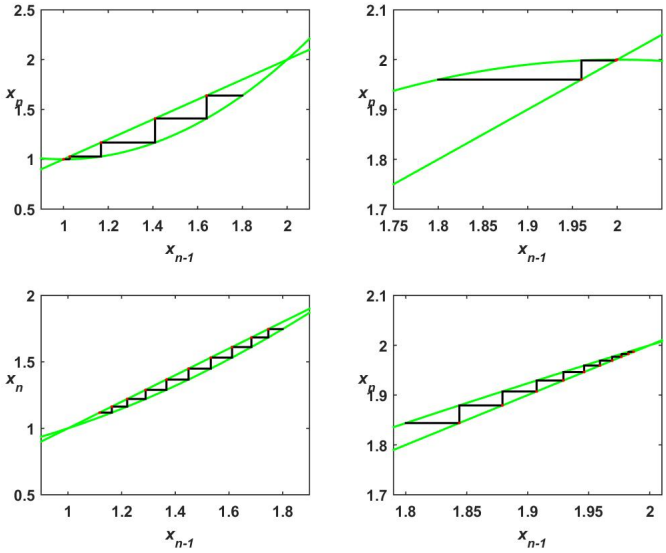


그림 5.3.4. 고정점법의 수렴성

```
x2 = [0.0]*(maxN+1)
x3 = [0.0]*(maxN+1)
x4 = [0.0]*(maxN+1)
x5 = [0.0]*(maxN+1)
x1[0] = 1.8;
for nn in range(0,maxN):
    x1[nn+1] = x1[nn]**2 - 2*x1[nn] +2;
x1 = np.asarray(x1)   # Convert list to nd-array
x2[0] = 1.8;
for nn in range(0,maxN):
    x2[nn+1] = -x2[nn]**2 + 4*x2[nn] - 2;
x2 = np.asarray(x2)   # Convert list to nd-array
x3[0] = 1.8;
for nn in range(0,maxN):
    x3[nn+1] = (x3[nn]**2 + 2)/3;
x3 = np.asarray(x3)   # Convert list to nd-array
x4[0] = 1.8;
for nn in range(0,maxN):
    x4[nn+1] = np.sqrt(3*x4[nn] - 2);
x4 = np.asarray(x4)   # Convert list to nd-array
x5[0] = 1.8;
for nn in range(0,maxN):
    x5[nn+1] = 1/101*(100*x5[nn]+3-2/x5[nn]);
x5 = np.asarray(x5)   # Convert list to ndarray

# Plotting
xd = np.linspace(0,3,301);
nx = np.arange(0,2*maxN); print(nx)
ixR = np.around((nx+1-10**(-3))/2);
ix = ixR.astype(np.int64); print(ix)      # real to integer
ny = np.arange(1,2*maxN+1); print(ny)
iyR = np.around((ny+1-10**(-3))/2);
iy = iyR.astype(np.int64); print(iy)      # real to integer
fig = plt.figure()
# row and column sharing
# f, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, sharex='col', sharey='row
    ')
```

```
ax1 = plt.subplot(221)
ax1.plot(xd,xd,'g-',xd,xd**2-2*xd+2,'g-',lw=2)
ax1.plot(x1[ix],x1[iy],'k-',x1[iy],x1[iy],'r.',lw=2)
ax1.set_xlim([0.9, 2.1]);  ax1.set_ylim([0, 3])
ax1.set_xlabel('x_{n-1}'); ax1.set_ylabel('x_{n}')
ax2 = plt.subplot(222)
ax2.plot(xd,xd,'g-',xd,-xd**2+4*xd-2,'g-',lw=2)
ax2.plot(x2[ix],x2[iy],'k-',x2[iy],x2[iy],'r.',lw=2)
ax2.set_xlim([1.75, 2.05]); ax2.set_ylim([1.6, 2.2])
ax2.set_xlabel('x_{n-1}');  ax2.set_ylabel('x_{n}')
ax3 = plt.subplot(223)
ax3.plot(xd,xd,'g-',xd,(xd**2+2)/3,'g-',lw=2)
ax3.plot(x3[ix],x3[iy],'k-',x3[iy],x3[iy],'r.',lw=2)
ax3.set_xlim([0.9, 1.9]); ax3.set_ylim([0.5, 2.0])
ax3.set_xlabel('x_{n-1}'); ax3.set_ylabel('x_{n}')
ax4 = plt.subplot(224)
ax4.plot(xd,xd,'g-',xd,np.sqrt(3*xd-2),'g-',lw=2)
ax4.plot(x4[ix],x4[iy],'k-',x4[iy],x4[iy],'r.',lw=2)
ax4.set_xlim([1.79, 2.01]); ax4.set_ylim([1.6, 2.2])
ax4.set_xlabel('x_{n-1}'); ax4.set_ylabel('x_{n}')
plt.show()
fig.savefig('FixedPointMethod104Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.3.7의 결과와 같다. ■

예제 5.3.9 다음 함수를 살펴보자.

$$g(x) = x^2 + \frac{3}{16} \tag{1}$$

방정식 $g(x) = 0$를 고정점법으로 풀기 위해서, 다음 점화식을 적용하기로 하자.

$$x_{n+1} = x_n^2 + \frac{3}{16} \tag{2}$$

점화식 (2)의 고정점은 $x = g(x)$의 근들인 0.25와 0.75이다. 다음 식들이 성립한다.

$$g'\left(\frac{1}{4}\right) = \frac{1}{2}, \quad g'\left(\frac{3}{4}\right) = \frac{3}{2} \tag{3}$$

따라서, 고정점 0.25 주변에서 점화식 (2)는 수렴할 것이고, 고정점 0.75 주변에서 점화식 (2)는 발산할 것이다. 이를 확인하기 위해서, 다음 MATLAB프로그램 FixedPointMethod901DY.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: FalsePositionMethod901DY.m
```

```
% Fixed Point Method
% Programmed by CDY
% ------------------------------------------------------------------------
clear all, close all, format short g
ftn = @(x) x.^2 + 3/16
Nmax = 30
idum = (1:1:Nmax)';
X = zeros(4,Nmax);
X(:,1) = [ 0 0.5 0.7499 0.7501]';
for ii = 1:1:Nmax-1;
    X(:,ii+1) = ftn(X(:,ii));
end
% Plotting
plot(idum,X(1,idum),'g-',idum,X(2,idum),'b-.', ...
     idum,X(3,idum),'k--',idum,X(4,idum),'r:','LineWidth',2);
legend('\bf x(0) = 0','\bf x(0) = 0.5','\bf x(0) = 0.7499',...
       '\bf x(0) = 0.7501','location','NW')
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\bf \it x_{n}','fontsize',12,'rotation',0)
axis([1 30 0 3 ])
% title('\bf False Posiiton Method')
saveas(gcf,'FixedPointMethod901DY.jpg')
save('FixedPointMethod901DY.txt','X','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 FixedPointMethod901DY.m을 실행하면, 이 점화식들의 그래프들을 그린 그림 5.3.5이 그려진다. 그림 5.3.5에서 알 수 있듯이, 초기값 x_0가 고정점 0.25보다 작으면, 점화식 (2)에 의해서 발생된 수열 $\{x_n\}$은 고정점 0.25로 수렴한다. 명제 5.3.1에서 알 수 있듯이, 그 이유는 $|g'(0.25)| < 1$이기 때문이다. 초기값 x_0가 고정점 0.25이면, 수열 $\{x_n\}$은 각 n에 대해서 식 $x_n = 0.25$을 만족한다. 초기값 x_0가 구간 $(0.25, 0.75)$에 속하면, 수열 $\{x_n\}$은 고정점 0.25로 수렴한다. 그 이유는 $|g'(0.25)| < 1$이기 때문이다. 예를 들어, 초기값 $x_0 = 0.7499$에 해당하는 수열 $\{x_n\}$도 고정점 0.25로 수렴한다. 이 경우에 다음 부등식이 성립한다.

$$|x_n - 0.25| \le \alpha^n |x_0 - 0.25|, \quad (n = 1, 2, \cdots) \tag{4}$$

여기서 $\alpha = g'(0.25) = 0.5$이다. 따라서, 다음 부등식이 성립하면 제n단계 절대오차는 10^{-M}보다 작다.

$$n > \frac{1}{\ln 4}\left[M \ln 10 + \ln |x_0 - 0.25|\right] \tag{5}$$

초기값 x_0가 0.75보다 크면, 해당하는 수열 $\{x_n\}$은 발산한다. 그 이유는 $|g'(0.75)| > 1$이기 때문이다. 예를 들어, 초기값 $x_0 = 0.7501$에 해당하는 수열 $\{x_n\}$도 ∞로 발산한다.

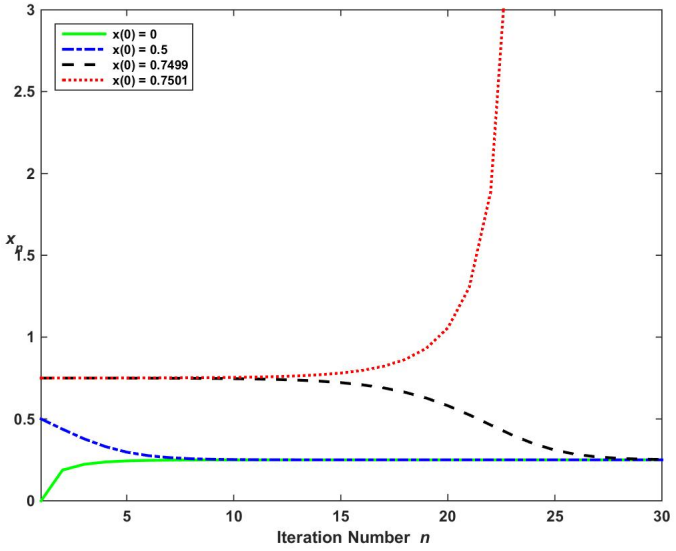


그림 5.3.5. 초기값과 수렴성

예제 5.3.10 Python을 사용해서 예제 5.3.9을 다시 다루기 위해서, 다음 Python프로그램 FixedPointMethod901DY.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

ftn = lambda x: x**2 + 3/16
Nmax = 30
idum = np.arange(0,Nmax)
X = np.zeros((4,Nmax));
X[0,0] =0;  X[1,0] = 0.5;  X[2,0] = 0.7499;  X[3,0] = 0.7501;
for ii in range(0,Nmax-1):
    X[:,ii+1] = ftn(X[:,ii]);

# Plotting
ii = np.arange(0,20)
fig = plt.figure()
plt.plot(idum,X[0,idum],'g-',lw=2,label='x(0) = 0')
plt.plot(idum,X[1,idum],'b-',lw=2,label='x(0) = 0.5')
plt.plot(idum,X[2,idum],'k--',lw=2,label='x(0) = 0.7499')
plt.plot(idum,X[3,idum],'r:',lw=2,label='x(0) = 0.7501')
plt.xlabel('Iteration Number n');
plt.ylabel('x_{n}')
plt.axis([ 0, 29, 0, 3 ])
plt.legend(loc='upper left', numpoints=1)
plt.show()
fig.savefig('FixedPointMethod901DYPy.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.3.9의 결과와 같다. ■

예제 5.3.11 이태리의 수학자 Fibonacci(또한, Leonardo Pisano Bigollo, Leonardo of Pisa, Leonardo Pisano, Leonardo Bonacci, Leonardo Fibonacci로 알려짐)는 1225년 다음과 같은 3차방정식 $f(x) = 0$의 근을 구하였다.

$$f(x) = x^3 + 2x^2 + 10x - 20 \tag{1}$$

Fibonacci가 방정식 (1)의 근 $\alpha = 1.358,808,107$을 구했지만, 당시에는 Fibonacci를 제외한 누구도 Fibonacci가 어떤 방법을 써서 이 근을 구했는지는 알지 못한다. 당시로서는 이 근의 정확도는 놀라운 것이었다. Fibonacci는 이 근을 다음과 같이 표기했다.

$$\alpha = 1 + \frac{22}{60^1} + \frac{7}{60^2} + \frac{42}{60^3} + \frac{33}{60^4} + \frac{4}{60^5} + \frac{40}{60^6}. \tag{2}$$

Fibonacci는 왜 60진법을 사용해서 해를 표기했을까? 그 이유는 메소포타미아에서 사용하던 기법을 적용해서 구했기 때문이 아닐까?

고정점법을 적용해서 3차방정식 (1)을 풀기 위해, 다음 MATLAB프로그램 FixedPoint-Method106.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: FixedPointMethod106.m
%  Solving Leonardo Fibonacci problem through Recursive Mehtods
%  Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
maxN = 100;
x = zeros(4,maxN);
x(:,1) = 1;
NN = maxN*ones(4,1);
alpha = NaN*ones(4,1);
Niter = 40;

% g(x) = 20/(x^2 + 2x + 10)
for nn=1:1:maxN
    x(1,nn+1) = 20/(x(1,nn)^2+2*x(1,nn)+10);
    if abs(x(1,nn+1)-x(1,nn)) < 10^-15,
        alpha(1) = x(1,nn+1);
        NN(1) = nn+1;
        break
    end
end
Niter1 = 1:NN(1);
X1 = x(1,Niter1);

```

```
% g(x) = (20 - 2x^2 - x^3)/10
for nn=1:1:maxN
    x(2,nn+1) = (20-2*x(2,nn)^2-2*x(2,nn)^3)/10;
    if abs(x(2,nn+1)-x(2,nn)) < 10^-15,
        alpha(2) = x(2,nn+1);
        NN(2) = nn+1;
        break
    end
end
Niter2 = 1:NN(2);
X2 = x(2,Niter2);

% g(x) = (20 - 10x - x^3)/(2x)
for nn=1:1:maxN
    x(3,nn+1) = (20-10*x(3,nn)-x(3,nn)^3)/(2*x(3,nn));
    if abs(x(3,nn+1)-x(3,nn)) < 10^-15,
        alpha(3) = x(3,nn+1);
        NN(3) = nn+1;
        break
    end
end
Niter3 = 1:NN(3);
X3 = x(3,Niter3);

% Newton-Raphson method
for nn=1:1:maxN
    dum41 = -20+x(4,nn)*(10+x(4,nn)*(2+x(4,nn)));
    dum42 = 10+x(4,nn)*(4+3*x(4,nn));
    x(4,nn+1) = x(4,nn) - dum41/dum42;
    if abs(x(4,nn+1)-x(4,nn)) < 10^-15,
        alpha(4) = x(4,nn+1);
        NN(4) = nn+1;
        break
    end
end
Niter4 = 1:NN(4);
X4 = x(4,Niter4);

% Output
alpha, NN
% Plotting
subplot(2,2,1)
plot(Niter1,X1,'r','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0,NN(1)])
xlabel('{\bf \it n}'), ylabel('\bf \it x_{n}','rotation',0)
subplot(2,2,2)
plot(Niter2,X2,'b','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0,NN(2)], ...
    'ylim',[-1,2.5])
xlabel('{\bf \it n}'), ylabel('\bf \it x_{n}','rotation',0)
subplot(2,2,3)
plot(Niter3,X3,'k','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0,NN(3)])
xlabel('{\bf \it n}'), ylabel('\bf \it x_{n}','rotation',0)
subplot(2,2,4)
plot(Niter4,X4,'g','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0,NN(4)])
xlabel('{\bf \it n}'), ylabel('\bf \it x_{n}','rotation',0)
saveas(gcf,'FixedPointMethod106','jpg')

% Sumer's Sexagesimal
```

```
alpha = X4(end)
s(1) = floor(60*(alpha-1))
r(1) = 60*(alpha-1) - s(1)
for jj=2:9
    dum = 60*r(jj-1);
    s(jj) = floor(dum);
    r(jj) = dum- s(jj);
end
s
save('FixedPointMethod106.txt','s','-ascii')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 FixedPointMethod106.m에서 사용한 첫 번째 점화식은 다음과 같다.

$$x_{n+1} = g_1(x_n) \doteq \frac{20}{x_n^2 + 2x_n + 10}, \quad x_0 = 1 \tag{3}$$

다음 식이 성립할 때, 이 점화식의 계산을 멈춘다.

$$|x_{44} - x_{43}| < 10^{-15} \tag{4}$$

이 경우에 근사해는 $x_{44} = 1.368,808,107,821,373$이다. 또한, 다음 식들이 성립한다.

$$g_1'(x_0) = -0.4734, \quad g_1'(\alpha) = -0.4438 \tag{5}$$

식 (5)에서 알 수 있듯이, 명제 5.3.1의 충분조건 $\lambda < 1$이 만족된다. 따라서, 점화식 (3)에 의해서 발생한 수열 $\{x_n\}$은 근 α로 수렴한다.

두 번째 점화식은 다음과 같다.

$$x_{n+1} = g_2(x_n) \doteq \frac{20 - 2x_n^2 - x_n^3}{10}, \quad x_0 = 1 \tag{6}$$

다음 식이 성립할 때, 이 점화식의 계산을 멈춘다.

$$|x_{47} - x_{46}| < 10^{-15} \tag{7}$$

이 경우에 수렴값은 $\beta = 2.000,000,000,000,000$이다. 이 값은 방정식 (1)의 근이 아니다. 다음 식들이 성립한다.

$$g_2'(x_0) = -1, \quad g_2'(\alpha) = -0.6717 \tag{8}$$

식 (8)에서 알 수 있듯이, 명제 5.3.1의 조건이 만족되지 않는다. 이는 점화식 (6)에 의해서 발생한 수열 $\{x_n\}$이 방정식 (1)의 근으로 수렴하지 않음을 설명하고 있다.

세 번째 점화식은 다음과 같다.

$$x_{n+1} = g_3(x_n) \doteq \frac{20 - 10x_n - x_n^3}{2x_n}, \quad x_0 = 1 \tag{9}$$

점화식 (9)는 n이 100이 되어도 멈추지 않는다. 즉, 점화식 (9)에 의해서 발생한 수열 $\{x_n\}$은 발산한다. 다음 식들이 성립한다.

$$g_3'(x_0) = -11, \quad g_3'(\alpha) = -6.7060 \tag{10}$$

식 (10)에서 알 수 있듯이, 명제 5.3.1의 조건이 만족되지 않는다. 이는 점화식 (9)에 의해서 발생한 수열 $\{x_n\}$은 방정식 (1)의 근으로 수렴하지 않음을 설명하고 있다.

네 번째 점화식은 다음과 같다.

$$x_{n+1} = g_4(x_n) \doteq x_n - \frac{x_n^3 + 2x_n^2 + 10x_n - 20}{3x_n^2 + 4x_n + 10}, \quad x_0 = 1 \tag{11}$$

다음 식이 성립할 때, 이 점화식의 계산을 멈춘다.

$$|x_7 - x_6| < 10^{-15} \tag{12}$$

이 경우에 근사해는 $x_6 = 1.368,808,107,821,373$이다. 점화식 (11)에 의해서 발생한 수열 $\{x_n\}$은 아주 빨리 수렴함을 알 수 있다. 다음 식들이 성립한다.

$$g_4'(x_0) = -0.2422, \quad g_4'(\alpha) = 0 \tag{13}$$

식 (13)에서 알 수 있듯이, 명제 5.3.1의 충분조건이 만족된다. 따라서, 점화식 (11)에 의해서 발생한 수열 $\{x_n\}$은 근 α로 수렴한다.

이 MATLAB프로그램 FixedPointMethod106.m을 실행하면, 이 점화식들의 그래프들을 그린 그림 5.3.6이 그려진다. 그림 5.3.6의 좌측상단 그래프는 점화식 (3)에 의한 수열을, 우측상단 그래프는 점화식 (6)에 의한 수열을, 좌측하단 그래프는 점화식 (9)에 의한 수열을, 그리고 우측하단 그래프는 점화식 (11)에 의한 수열을 그린 것이다. 세번째 점화식에 의하면 x_{11}은 $-\infty$이고, x_{12}는 NaN, 즉 숫자가 아니다. 따라서, 수열의 앞 부분만이 그려졌다.

이 $\alpha = 1.368,808,107,821,373$는 정확도가 10^{-15}이고, 또한 식 $60^{-7} = 3.572245084590763\times 10^{-13}$이다. 따라서, α를 60진법으로 소수 7자리까지 정확히 나타낼 수 있다. 이 MATLAB 프로그램 FixedPointMethod106.m을 실행하면, 근 α의 60진법 표기가 다음과 같음을 알 수 있다.

$$\alpha = 1 + \frac{22}{60^1} + \frac{7}{60^2} + \frac{42}{60^3} + \frac{33}{60^4} + \frac{4}{60^5} + \frac{38}{60^6} + \frac{30}{60^7} \tag{14}$$

■

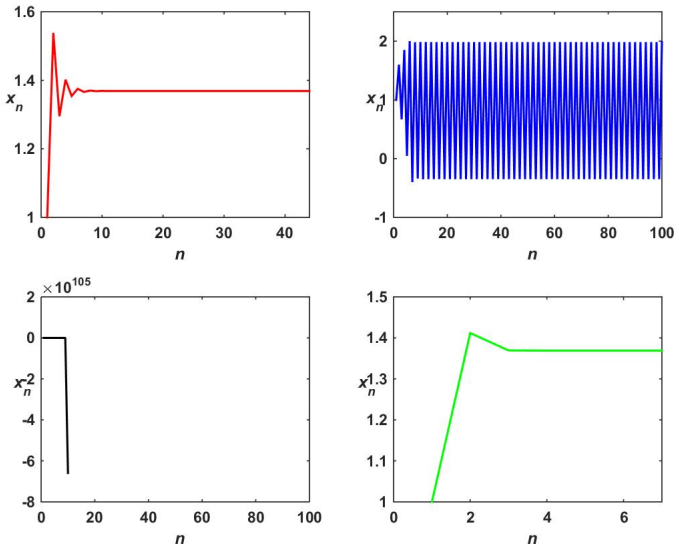


그림 5.3.6. Fibonacci문제

예제 5.3.12 Python을 사용해서 예제 5.3.11을 다시 다루기 위해서, 다음 Python프로그램 FixedPointMethod106.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

maxN = 100;
x = np.zeros((4,maxN));
x[:,0] = 1.0;
NN = maxN*np.ones((4,1));
alpha = np.nan*np.ones((4,1));
Niter = 40;
# g(x) = 20/(x**2 + 2x + 10)
for nn in range(0,maxN):
```

```
    x[0,nn+1] = 20/(x[0,nn]**2+2*x[0,nn]+10);
    if abs(x[0,nn+1]-x[0,nn]) < 10**(-15):
        alpha[0] = x[0,nn+1]
        NN[0] = nn+1;
        break

Niter1 = np.arange(0,min(NN[0]+1,maxN-1))
Niter1 = Niter1.astype(int)
X1 = x[0,Niter1]

# g(x) = (20 - 2x**2 - x**3)/10
for nn in range(0,maxN-1):
    x[1,nn+1] = (20-2*x[1,nn]**2-2*x[1,nn]**3)/10;
    if abs(x[1,nn+1]-x[1,nn]) < 10**(-15):
        alpha[1] = x[1,nn+1]
        NN[1] = nn+1;
        break
Niter2 = np.arange(0,min(NN[1]+1,maxN-1))
Niter2 = Niter2.astype(int)
X2 = x[1,Niter2]

# g(x) = (20 - 10x - x**3)/(2x)
for nn in range(0,maxN-1):
    x[2,nn+1] = (20-10*x[2,nn]-x[2,nn]**3)/(2*x[2,nn]);
    if abs(x[2,nn+1]-x[2,nn]) < 10**(-15):
        alpha[2] = x[2,nn+1]
        NN[2] = nn+1;
        break
Niter3 = np.arange(0,min(NN[2]+1,maxN-1))
Niter3 = Niter3.astype(int)
X3 = x[2,Niter3]

# Newton-Raphson method
for nn in range(0,maxN-1):
    dum41 = -20+x[3,nn]*(10+x[3,nn]*(2+x[3,nn]));
    dum42 = 10+x[3,nn]*(4+3*x[3,nn]);
    x[3,nn+1] = x[3,nn] - dum41/dum42;
    if abs(x[3,nn+1]-x[3,nn]) < 10**(-15):
        alpha[3] = x[3,nn+1]
        NN[3] = nn+1;
        break
Niter4 = np.arange(0,min(NN[3]+1,maxN-1))
Niter4 = Niter4.astype(int)
X4 = x[3,Niter4]

# Plotting
print(alpha); print(NN)

fig = plt.figure()
ax1 = plt.subplot(221)
ax1.plot(Niter1,X1,'r',lw=2)
ax1.set_xlim([0,NN[0]]); ax1.set_ylim([1.0, 1.6]);
ax1.set_xlabel('n'); ax1.set_ylabel('x_{n}')
ax2 = plt.subplot(222)
ax2.plot(Niter2,X2,'b',lw=2)
ax2.set_xlim([0,NN[1]]); ax2.set_ylim([-1, 2.5]);
ax2.set_xlabel('n'); ax2.set_ylabel('x_{n}')
ax3 = plt.subplot(223)
ax3.plot(Niter3,X3,'k',lw=2)
ax3.set_xlim([0,NN[2]]); ax3.set_ylim([-10e105, 5e105]);
ax3.set_xlabel('n'); ax3.set_ylabel('x_{n}')
```

```
ax4 = plt.subplot(224)
ax4.plot(Niter4,X4,'g',lw=2)
ax4.set_xlim([0,NN[3]]);  ax4.set_ylim([1.0, 1.6]);
ax4.set_xlabel('n'); ax4.set_ylabel('x_{n}')
plt.show()
fig.savefig('FixedPointMethod106Py.png')

# Sumer's Sexagesimal
s = np.arange(0,9);
r = [0.0]*9;
alpha = X4[-1]
s[0] = np.floor(60*(alpha-1))
r[0] = 60*(alpha-1) - s[0]
for jj in range(1,9):
    dum = 60*r[jj-1];
    s[jj] = np.floor(dum);
    r[jj] = dum- s[jj];
sInt = s.astype(int)
print(sInt)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 5.3.11의 결과와 같다. ■

예제 5.3.13 수열 $\{x_n\}$이 극한값 a에 1차적으로 수렴한다고(linearly converge) 가정하면, 다음 식들을 만족하는 상수 $A(\in (-1,1))$가 존재한다.

$$a - x_{n+1} \doteq e_{n+1} \approx Ae_n \doteq A[a - x_n], \quad (n = 1, 2, \cdots) \tag{1}$$

Aitken의 델타제곱과정을 사용해서 Steffensen알고리즘을 유도하기로 하자. 다음 식들이 성립한다.

$$\frac{a - x_{n+1}}{a - x_n} \approx A \approx \frac{a - x_{n+2}}{a - x_{n+1}} \tag{2}$$

근사식 (2)를 등식으로 간주하면, 다음 식을 얻는다.

$$[a - x_{n+1}]^2 = [a - x_n]\,[a - x_{n+2}] \tag{3}$$

식 (3)을 정리하면, 다음 식을 얻는다.

$$a = \frac{x_{n+2}x_n - x_{n+1}^2}{x_{n+2} - 2x_{n+1} + x_n} = x_n - \frac{[x_{n+1} - x_n]^2}{x_{n+2} - 2x_{n+1} + x_n} \tag{4}$$

식 (4)의 우변을 x_{n+3}으로 놓으면, 다음 점화식이 성립한다.

$$x_{n+3} = x_n - \frac{[x_{n+1} - x_n]^2}{x_{n+2} - 2x_{n+1} + x_n}, \quad (n = 1, 2, \cdots) \tag{5}$$

점화식 (5)를 사용해서 방정식을 푸는 방법을 Steffensen법이라 한다.

점화식 (5)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\begin{aligned} a - x_{n+3} &= a - x_n + \frac{[x_{n+1} - x_n]^2}{x_{n+2} - 2x_{n+1} + x_n} \\ &= a - x_n + \frac{\{-[a - x_{n+1}] + a - x_n]\}^2}{-[a - x_{n+2}] + 2[a - x_{n+1}] - [a - x_n]} \end{aligned} \tag{6}$$

식 (1)을 식 (6)에 대입하면, 다음 식을 얻는다.

$$\begin{aligned} a - x_{n+3} &\approx a - x_n + \frac{[-A + 1]^2[a - x_n]^2}{[-A^2 + 2A - 1][a - x_n]} \\ &= a - x_n - [a - x_n] = 0 \end{aligned} \tag{7}$$

식 (7)에서 알 수 있듯이, 수열 $\{x_n\}$이 1차적으로 수렴할 때 Steffensen법을 적용하면 수열 $\{x_n\}$의 수렴속도는 향상된다.

Steffensen법이 수렴속도를 증가하는 예를 살펴보기 위해서, 다음 MATLAB프로그램 SteffensenMethod101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: SteffensenMethod101.m
% Steffensen Method for solving x = fcn(x)
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long, format compact
fcn = inline('-(z^2 - 2)/1.5 + z','z')
tol = 1.0e-8;
x(1) = 1; y(1) = x(1);
for ii= 1:3:1000
    y(ii+1) = fcn(y(ii));
    y(ii+2) = fcn(y(ii+1));
    y(ii+3) = fcn(y(ii+2));
    x(ii+1) = fcn(x(ii));
    x(ii+2) = fcn(x(ii+1));
    x(ii+3) = x(ii) - (x(ii+1)-x(ii))^2 ...
                /(x(ii+2) - 2*x(ii+1) + x(ii));
    if abs(x(ii+3)-x(ii)) < tol;
       break
    end
end
ii, y, x, y-x
% Plotting
nn = (1:1:ii+3);
plot(nn,y,'k--',nn,x,'r-','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
legend('Simple Iteration','Steffensen','location','SE')
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\bf Solution \it x_n','fontsize',12)
axis( [ 1, ii+3, 0.9 1.7 ])
```

```
saveas(gcf,'SteffensenMethod101','jpg')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 SteffensenMethod.m은 방정식 $x^2-2=0$을 풀기 위한 것이다. 다음과 같은 고정점법에 의한 점화식을 살펴보자.

$$y_{n+1}=-\frac{2}{3}\left[y_n^2-2\right]+y_n,\quad y_0=1 \tag{8}$$

다음 식들이 성립한다.

$$y_{n+1}-\sqrt{2}=\left\{1-\frac{2}{3}\left[y_n+\sqrt{2}\right]\right\}\left[y_n-\sqrt{2}\right]\approx\left[1-\frac{4\sqrt{2}}{3}\right]\left[y_n-\sqrt{2}\right] \tag{9}$$

이 고정법에 의해 발생된 수열 $\{y_n\}$은 극한값 $\sqrt{2}$에 1차적으로 수렴한다. 이 고정법에 Steffensen법을 적용하면 다음 점화식을 얻는다.

$$x_{m+3}=\begin{cases}x_m-\frac{[x_{m+1}-x_m]^2}{x_{m+2}-2x_{m+1}+x_m}, & (m=1,4,7,\cdots)\\ -\frac{2}{3}\left[x_{m+2}^2-2\right]+x_{m+2}, & (m=0,2,3,5,\cdots)\end{cases} \tag{10}$$

이 점화식의 초기값들로 $x_1=1$과 $x_2=5/3$를 사용한다. 이 MATLAB프로그램 Steffensen-Method101.m이 실행한 결과, 다음 식이 성립할 때 점화식 (10)의 계산을 멈춘다.

$$|x_{16}-x_{15}|<10^{-10} \tag{11}$$

이 경우에 근사해는 $x_{16}=1.414,213,562$이다. 또한, 이 MATLAB프로그램이 실행되면, Steffensen식 (10)에 의해서 발생한 수열 $\{x_n\}$과 점화식 (8)에 의해서 발생한 수열 $\{y_n\}$을 그린 그림 5.3.7가 그려진다. 그림 5.3.7에서 알 수 있듯이, Steffensen알고리즘에 의한 수열 $\{x_n\}$의 수렴속도가 단순반복알고리즘에 의한 수열 $\{y_n\}$의 수렴속도보다 훨씬 빠르다. ■

예제 5.3.14 Python을 사용해서 예제 5.3.13을 다시 다루기 위해서, 다음 Python프로그램 SteffensenMethod101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
```

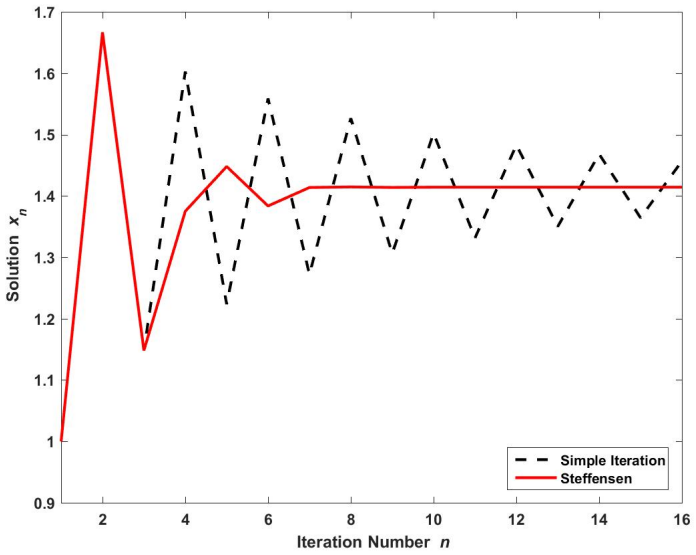


그림 5.3.7. Steffensen법

```
"""

import numpy as np
import matplotlib.pyplot as plt

fcn = lambda z: -(z**2 - 2)/1.5 + z;
tol = 1.0e-8;
x = np.zeros(1003)
y = np.zeros(1003)
x[0] = 1; y[0] = 1;
print(x)
for ii in range(0,1000,3):
    y[ii+1] = fcn(y[ii]);
    y[ii+2] = fcn(y[ii+1]);
    y[ii+3] = fcn(y[ii+2]);
    x[ii+1] = fcn(x[ii]);
    x[ii+2] = fcn(x[ii+1]);
    x[ii+3] = x[ii] - (x[ii+1]-x[ii])**2/(x[ii+2] - 2*x[ii+1] + x[ii]);
    if abs(x[ii+3]-x[ii]) < tol:
       break
print(ii); print(x[0:(ii+3)]); print(y[0:(ii+3)]);
yMx = y-x; print(yMx[0:(ii+3)]);

# Plotting
nn = np.arange(0,ii+4)
yy = y[0:ii+4]
xx = x[0:ii+4]
fig = plt.figure()
plt.plot(nn,yy,'k--',lw=2, label='Simple Iteration')
plt.plot(nn,xx,'r-',lw=2, label='Steffensen')
plt.xlabel('Iteration Number n');
plt.ylabel('Solution x_{n}')
plt.axis( [ 0, ii+3, 0.9, 1.7 ])
plt.legend(loc='lower right', numpoints=1)
plt.show()
fig.savefig('SteffensenMethod101Py.jpg')

```

```
# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.3.13의 결과와 같다. ■

제5.4절 Newton-Raphson법

가장 널리 사용되는 고정점법은 Newton-Raphson법이다. Newton-Raphson법은 1차 Taylor 근사를 이용하는 방법이다. 다음 근사식이 성립한다.

$$f(x) \approx f(x_n) + f'(x_n)[x - x_n] \tag{5.4.1}$$

식 (5.4.1)은 함수 $f(x)$를 점 x_n에서 접선으로 근사시킨 것이다. 따라서, 만약 x_n이 근 α에 가깝다면, 방정식 $f(x_n) + f'(x_n)[x - x_n] = 0$의 근은 더 α에 가깝다. 이 근을 x_{n+1}로 표기하면, 다음 식이 성립한다.

$$f(x_n) + f'(x_n)[x_{n+1} - x_n] = 0 \tag{5.4.2}$$

식 (5.4.2)를 다음과 같이 쓸 수 있다.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad (n = 1, 2, \cdots) \tag{5.4.3}$$

점화식 (5.4.3)을 사용해서 방정식 $f(x) = 0$를 구하는 것을 Newton-Raphson법이라 한다. 식 (5.4.3)을 직관적으로 이해하기 위해서 다음 예제를 살펴보자.

예제 5.4.1 Newton-Raphson알고리즘을 직관적으로 이해하기 위해서, 다음 M-파일 NewtonRaphson101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: NewtonRaphson101.m
%  Graphic Demonstration of Newton-Raphson method
%  Programmed by CBS
% ------------------------------------------------------------------------
clear all, clf, format long
ftn = @(x) exp(x) - 5
dftn = @(x) exp(x)
TOL = 10^-10; % toleration number
x = zeros(20,1);
format long;
```

```
% initial iteration
 x(1) = 2.5; % initial guess of the root
 xdiff = 0.1;
% iteration
for iter = 1:1:20;
    fval(iter) = feval(ftn,x(iter));
    dfval(iter) = feval(dftn,x(iter));
    if abs(dfval(iter)) < TOL
      disp('| derivative | < TOL')
      disp('algorithm diverges')
      disp('to continue, hit any key')
      disp('to interrupt, hit Ctrl-C')
    end
    xdiff = fval(iter)/dfval(iter);
    x(iter+1) = x(iter) - xdiff;
    if abs(xdiff) < TOL; break; end
end
iter,x(1:iter+1)
% Ploting
xx = 1.4:0.001:2.2;
yy = ftn(xx);
y2 = ftn(x(2)) + (0-ftn(x(2)))/(x(3)-x(2)).*(xx-x(2));
ydum2 = 0:0.01:ftn(x(2));
set(gca,'fontsize',11,'fontweigh','bold')
hold on
plot(xx,yy,'k',xx,y2,'r--','LineWidth',2.5)
legend('function','linear approx','location','SE')
plot([1 3],[0 0 ],'g-','LineWidth',1.5)
plot([x(2) x(2)],[0,ftn(x(2))],'k-.','LineWidth',1.5)
text(1.90,-0.14,'\bf x_n')
text(1.65,-0.14,'\bf x_{n+1}')
text(log(5)-0.02,0.17,'\bf \alpha')
axis([1.35 2.25 -1.5 4.0 ])
xlabel('\it x','Fontsize',12','Fontweigh','bold');
ylabel('\it y','Fontsize',12','Fontweigh','bold');
hold off
saveas(gcf,'NewtonRaphson101','jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 NewtonRaphson101.m에서는 함수 $f(x) = e^x - 5$에 대한 방정식 $f(x) = 0$를 Newton-Raphson법으로 푸는 과정을 설명하기 위한 것이다. 이 MATLAB프로그램에서는 Newton-Raphson법을 적용하기 위해서는 다음 점화식을 사용한다.

$$x_{n+1} = x_n - \frac{\exp(x_n) - 5}{\exp(x_n)}, \quad x_0 = 2.5 \tag{1}$$

이 MATLAB프로그램을 실행하면, 다음 조건이 만족될 때 이 점화식의 반복을 멈춘다.

$$|f'(x_6)| < 10^{-10} \tag{2}$$

이 경우에 근사해는 $x_6 = 1.6094$이다. 또한, Newton-Raphson식 (5.4.3)을 설명하는 그림 5.4.1을 출력한다. ■

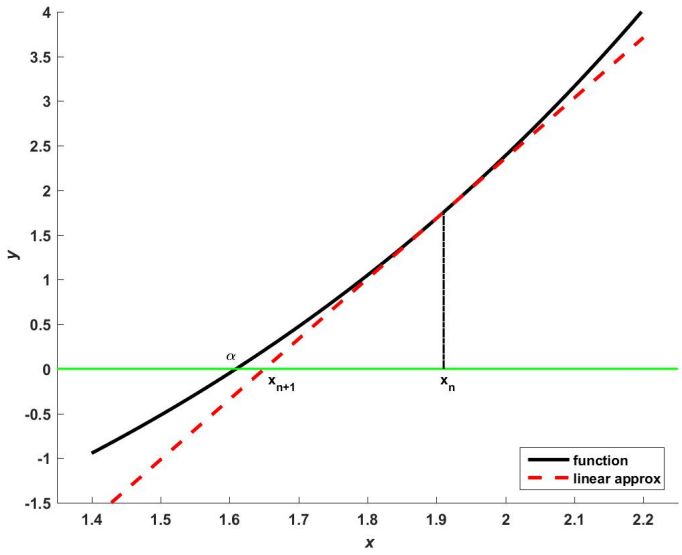


그림 5.4.1. Newton-Raphson법과 접선

예제 5.4.2 Python을 사용해서 예제 5.4.1을 다시 다루기 위해서, 다음 Python프로그램 NewtonRaphson101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

ftn = lambda x: np.exp(x) - 5
dftn = lambda x: np.exp(x)
TOL = 10**(-10);            # toleration number
x = np.zeros(20);
fval = np.zeros(20);
dfval = np.zeros(20);
# initial iteration
x[0] = 2.5;  #  initial guess of the root
xdiff = 0.1;
# iteration
for iter in range(0,20):
    fval[iter] = ftn(x[iter]);
    dfval[iter] = dftn(x[iter]);
    if np.abs(dfval[iter]) < TOL:
      print('| derivative | < TOL')
      print('algorithm diverges')
      print('to continue, hit any key')
      print('to interrupt, hit Ctrl-C')
    xdiff = fval[iter]/dfval[iter];
    x[iter+1] = x[iter] - xdiff;
    if abs(xdiff) < TOL:
       break;
print(iter);
print(x[0:(iter+2)])
```

```
33
34 # Plotting
35 xx = np.arange(1.4,2.2005,0.001);
36 yy = ftn(xx);
37 y2 = ftn(x[1]) + (0-ftn(x[1]))/(x[2]-x[1])*(xx-x[1]);
38 ydum2 = np.arange(0,ftn(x[1])+0.005,0.01);
39 fig = plt.figure()
40 plt.plot(xx,yy,'k',lw=2, label='function')
41 plt.plot(xx,y2,'r--',lw=2, label='linear approx')
42 plt.legend(loc='lower right', numpoints=1)
43 plt.plot([1, 3],[0, 0],'g-',lw=2)
44 plt.plot([x[1], x[1]],[0,ftn(x[1])],'k-.',lw=2)
45 plt.text(1.90,-0.16,r'$x_n$')
46 plt.text(1.65,-0.16,r'$x_{n+1}$')
47 plt.text(np.log(5)-0.02,0.17,r'$\alpha$')
48 plt.axis([1.35, 2.25, -1.5, 4.0 ])
49 plt.xlabel('x'); plt.ylabel('y');
50 plt.show()
51 fig.savefig('NewtonRaphson101Py.jpg')
52
53 # End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.4.1의 결과와 같다. ■

Newton-Raphson식 (5.4.3)을 고정점법의 점화식 (5.3.6) 형태로 나타내면, 함수 $g(x)$는 다음과 같다.

$$g(x) = x - \frac{f(x)}{f'(x)} \tag{5.4.4}$$

식 (5.4.4)의 양변을 x로 미분하면, 다음 식이 성립한다.

$$g'(x) = \frac{f(x)f''(x)}{[f'(x)]^2} \tag{5.4.5}$$

식 (5.3.13)과 식 (5.4.5)에서 알 수 있듯이, Newton-Raphson알고리즘이 수렴하기 위한 충분조건은 다음과 같다.

$$\left| \frac{f(\alpha)f''(\alpha)}{[f'(\alpha)]^2} \right| < 1 \tag{5.4.6}$$

함수 $g(x)$를 $x = \alpha$에서 Taylor전개하면, 다음 식을 만족하는 ζ가 x와 α 사이에 존재한다.

$$g(x) = g(\alpha) + g'(\alpha)[x-\alpha] + \frac{g''(\zeta)}{2}[x-\alpha]^2 \tag{5.4.7}$$

식 $f(\alpha) = 0$이 성립하므로, 식 (5.4.5)에서 알 수 있듯이 다음 식이 성립한다.

$$g'(\alpha) = 0 \tag{5.4.8}$$

식 $\alpha = g(\alpha)$와 식 (5.4.8)을 식 (5.4.7)에 대입하면, 다음 식을 얻는다.

$$g(x) - \alpha = \frac{g''(\zeta)}{2}[x - \alpha]^2 \tag{5.4.9}$$

따라서, 오차 $e_n \doteq x_n - \alpha$는 다음 식을 만족한다.

$$e_{n+1} = \frac{1}{2}g''(\zeta_n)e_n^2, \quad (n = 0, 1, \cdots) \tag{5.4.10}$$

여기서 ζ_n은 x_n과 α 사이의 점이다. 식 (5.4.10)을 다음과 같이 쓸 수 있다.

$$e_{n+1} = O\left(e_n^2\right) \tag{5.4.11}$$

식 (5.4.11)에서 알 수 있듯이, Newton-Raphson법은 2차 수렴속도를 갖는다. 즉, Newton-Raphson법이 수렴하는 경우에는 머신엡스(machine eps) 수준까지 빠르게 수렴하는 고정점법이다. 그러나, Newton-Raphson법에서 초기값을 근 α에 가깝게 선택하지 않으면, 점화식에서 발생된 수열이 수렴하지 않는 경우가 있다. 즉, Newton-Raphson법의 수렴성은 초기값에 크게 의존한다. 따라서, 실제 방정식을 풀 때에는 앞에서 다룬 이분할법 등을 사용해서 근 α에 가까운 초기값을 구한 다음, Newton-Raphson법을 적용하기도 한다.

Newton-Raphson식 (5.4.3)을 기하학적으로 적용하는 과정을 이해하기 위해서, 다음 예제를 살펴보자.

예제 5.4.3 Newton-Raphson알고리즘을 직관적으로 이해하기 위해서, 다음 M-파일 NewtonRaphson102.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: NewtonRaphson102.m
%  Geometrical Interpretation of Newton-Raphson Method
%  Programmed by CBS
%------------------------------------------------------------------
close all; clear all;
x(1) = 1/2;
for n=2:1:20;
    x(n) = 1/2*(x(n-1)+2/x(n-1));
end
nn = 1:1:20;
errorr = sqrt(2)-x;
format long
A = [ nn' x' errorr' ]
yx1 = x;
yx2 = (x +2./x)/2;
xx = 0.4:0.005:3;
y1 = xx;
```

```
y2 = (xx +2./xx)/2;
hold on
plot(xx,y1,'g.',xx,y2,'g.','linewidth',2)
plot(x,yx1,'k.',x,yx2,'k.','linewidth',2)
plot([x(1) x(1) x(2) x(2) x(3) x(3) x(4) x(4)], ...
     [0 x(2) x(2) x(3) x(3) x(4) x(4) x(5)],...
     'k-','linewidth',1.5)
set(gca,'fontsize',10,'fontweigh','bold')
plot([-1 4],[0 0],'r-',[0 0],[-1 4],'r-','linewidth',1.2)
text(0.5,0.05,'\bf A_{0}'), text(0.45,2.32,'\bf A_{1}')
text(2.25,2.2,'\bf A_{2}'), text(2.25,1.6,'\bf A_{3}')
text(1.35,1.6,'\bf A_{4}')
text(2.5,2.5,'\bf y = x')
text(2.7,1.7,'\bf y = g(x)')
axis( [-0.2,3, -0.2, 3 ])
axis equal
hold off
saveas(gcf,'NewtonRaphson102','jpg')
% End of program
%-------------------------------------------------------------------
```

이 MATLAB프로그램 NewtonRaphson102.m에서는 방정식 $x^2-2=0$의 양근을 구하기 위해서, 다음 Newton-Raphson알고리즘을 사용한다.

$$x_{n+1} = g(x_n) = \frac{1}{2}\left[x_n + \frac{2}{x_n}\right], \quad (n = 1, 2, \cdots) \tag{1}$$

이 MATLAB프로그램 NewtonRaphson102.m을 실행하면, 그림 5.4.2이 그려진다. 그림 5.4.2에서, 점 A_0는 $(x_0, 0)$, 점 A_1는 (x_0, x_1), 점 A_2는 (x_1, x_1), 점 A_3는 (x_1, x_2), 점 A_4는 $(x_2, x_2), \cdots$ 이다. 이 점들 $A_0, A_1, A_2, A_3, \cdots$ 가 방정식 $x = g(x)$의 근으로 수렴함을 알 수 있다. ■

예제 5.4.4 Python을 사용해서 예제 5.4.3을 다시 다루기 위해서, 다음 Python프로그램 NewtonRaphson102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.zeros(20);
x[0] = 1/2;
for n in range(1,20):
    x[n] = 1/2*(x[n-1]+2/x[n-1]);
errorr = np.sqrt(2)-x;
nn = np.arange(0,20)
```

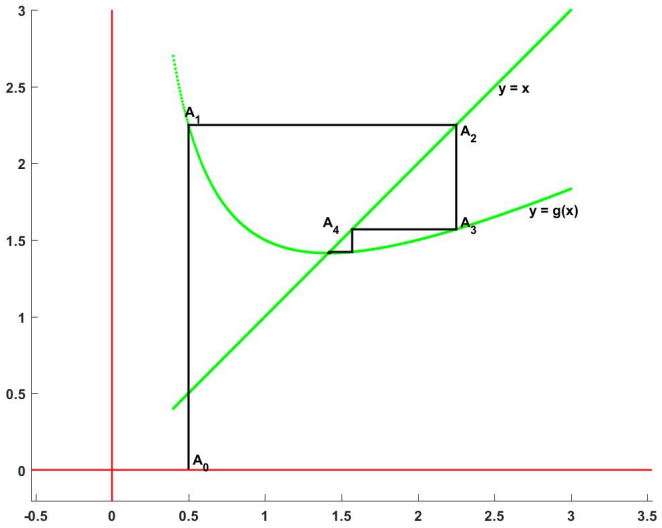


그림 5.4.2. Newton-Raphson법의 기하학

```
A = np.vstack((nn,x,errorr))
yx1 = x;
yx2 = (x +2/x)/2;
xx = np.arange(0.4,3.001,0.005);
y1 = xx;
y2 = (xx +2./xx)/2;

# Plotting
fig = plt.figure()
plt.plot(xx,y1,'g.',xx,y2,'g.',lw=2)
plt.plot(x,yx1,'k.',x,yx2,'k.',lw=2)
xdum1 = [ x[0], x[0], x[1], x[1], x[2], x[2], x[3], x[3] ]
ydum1 = [ 0, x[1], x[1], x[2], x[2], x[3], x[3], x[4] ]
plt.plot(xdum1,ydum1,'k-',lw=2)
plt.plot([-1, 4],[0, 0],'r-', [0, 0],[-1, 4],'r-',lw=1.5)
plt.text(0.5,0.05,r'$A_{0}$')
plt.text(0.45,2.32,r'$A_{1}$')
plt.text(2.25,2.2,r'$A_{2}$')
plt.text(2.25,1.6,r'$A_{3}$')
plt.text(1.35,1.6,r'$A_{4}$')
plt.text(2.75,2.5,'y = x')
plt.text(3.0,1.7,'y = g(x)')
plt.axis( [-0.2,3, -0.2, 3 ])
plt.axis('equal')
plt.text(4.05,0.1,r'$x_n$');
plt.text(-0.5,3.5,r'$x_{n+1}$');
plt.show()
fig.savefig('NewtonRaphson102Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.4.3의 결과와 같다. ■

예제 5.4.5 방정식 $x^2 - 2 = 0$의 양근을 구하기 위해서, 다음 MATLAB프로그램 NewtonRaphson103.m을 실행하라.

```
% ------------------------------------------------------------------
%  Filename: NewtonRaphson103.m
%  Newton-Raphson method to solve x^2 = 2
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, clf, format long
nmax = 9;
for dumk = 1:1:3
    a(dumk) = dumk-1;
    x(1,dumk) = 1.6;
    for n = 1:nmax
        x(n+1,dumk) = a(dumk) ...
                  + (2-a(dumk)*a(dumk))/(x(n,dumk)+a(dumk));
    end
end
a
x
% Newton-Raphson method
xn(1) = 1.6;
for n = 1:nmax
        xn(n+1) = ( xn(n) + 2/xn(n) )/2;
end
% Plotting
ii = (1:1:nmax+1);
plot(ii,x(:,1),'g--',ii,x(:,2),'r:',ii,x(:,3),'b-.', ...
      ii,xn(:),'k','LineWidth',2.5);
set(gca,'fontsize',11,'fontweigh','bold')
legend('a=0','a=1','a=2','Newton-Raphson','location','SE')
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\bf Solution \it x_{n}','fontsize',12)
axis( [ 1, 10, 1.15 1.65 ])
saveas(gcf,'NewtonRaphson103','jpg')
save('NewtonRaphsonGraph103.txt','xn','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 NewtonRaphson103.m에서는 방정식 $x^2 - 2 = 0$의 양근을 구하기 위해서, 다음 Newton-Raphson알고리즘을 사용한다.

$$x_{n+1} = \frac{1}{2}\left[x_n + \frac{2}{x_n}\right], \quad (n = 1, 2, \cdots) \tag{1}$$

식 (1)에서 알 수 있듯이, 다음 점화식들이 성립한다.

$$x_{n+1} - \sqrt{2} = \frac{1}{2x_n}[x_n - \sqrt{2}]^2, \quad (n = 1, 2, \cdots) \tag{2}$$

$$x_{n+1} + \sqrt{2} = \frac{1}{2x_n}[x_n + \sqrt{2}]^2, \quad (n = 1, 2, \cdots) \tag{3}$$

식 (2)의 양변을 식 (3)의 양변으로 나누면, 다음 점화식이 성립한다.

$$\frac{x_{n+1}-\sqrt{2}}{x_{n+1}+\sqrt{2}} = \left[\frac{x_n-\sqrt{2}}{x_n+\sqrt{2}}\right]^2, \quad (n=1,2,\cdots) \tag{4}$$

식 (4)를 다음과 같이 쓸 수 있다.

$$\ln\left|\frac{x_{n+1}-\sqrt{2}}{x_{n+1}+\sqrt{2}}\right| = 2\ln\left|\frac{x_n-\sqrt{2}}{x_n+\sqrt{2}}\right|, \quad (n=1,2,\cdots) \tag{5}$$

식 (5)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\ln\left|\frac{x_n-\sqrt{2}}{x_n+\sqrt{2}}\right| = 2^{n-1}\ln\left|\frac{x_1-\sqrt{2}}{x_1+\sqrt{2}}\right|, \quad (n=1,2,\cdots) \tag{6}$$

식 (6)을 다음과 같이 쓸 수 있다.

$$\left|\frac{x_n-\sqrt{2}}{x_n+\sqrt{2}}\right| = \left|\frac{x_1-\sqrt{2}}{x_1+\sqrt{2}}\right|^{2^{n-1}}, \quad (n=1,2,\cdots) \tag{7}$$

만약 부등식 $x_1 > 0$가 성립하면, 다음 식이 성립한다.

$$\left|\frac{x_1-\sqrt{2}}{x_1+\sqrt{2}}\right| < 1 \tag{8}$$

식 (7)과 식 (8)에서 알 수 있듯이, 부등식 $x_1 > 0$가 성립하면 다음 식이 성립한다.

$$\lim_{n\to\infty} x_n = \sqrt{2} \tag{9}$$

식 (2)에서 알 수 있듯이, 오차항 $e_n \doteq x_n - \sqrt{2}$은 다음 근사식을 만족한다.

$$e_{n+1} \approx \frac{1}{2\sqrt{2}} e_n^2 \quad \text{for} \quad \text{large } n \tag{10}$$

식 (10)에서 알 수 있듯이, $\{x_n\}$은 수렴속도는 2차이다. 즉, 식 (10)은 식 (5.4.10)을 예증한 것이다.

이 MATLAB프로그램 NewtonRaphson103.m을 실행하면, 예제 5.3.1에서 다룬 고정점 법을 $a = 0, 1, 2$에 대해서 적용한 결과와 Newton-Raphson알고리즘을 적용한 결과가 그림 5.4.3이 그려진다. 그림 5.4.3에서 알 수 있듯이, 이 중에서 Newton-Raphson알고리즘이 가장 빨리 수렴한다.

식 (10)을 다음과 같이 유도할 수도 있다. 산술평균과 기하평균의 부등식에서 다음 식들이 성립함을 알 수 있다.

$$x_{n+1} = \frac{1}{2}\left[x_n + \frac{2}{x_n}\right] \geq \sqrt{x_n \frac{2}{x_n}} = \sqrt{2}, \quad (n = 1, 2, \cdots) \tag{11}$$

식 (1)과 식 (11)에서 알 수 있듯이, 다음 식들이 성립한다.

$$x_{n+1} - \sqrt{2} = \frac{1}{2x_n}[x_n - \sqrt{2}]^2 \leq \frac{1}{2\sqrt{2}}[x_n - \sqrt{2}]^2, \quad (n = 2, 3, \cdots) \tag{12}$$

식 (12)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\left|x_{n+1} - \sqrt{2}\right| \leq \frac{1}{2\sqrt{2}}[x_n - \sqrt{2}]^2, \quad (n = 2, 3, \cdots) \tag{13}$$

즉, 다음 식이 성립한다.

$$\epsilon_{n+1} \leq \frac{1}{2\sqrt{2}}\epsilon_n^2, \quad (n = 2, 3, \cdots) \tag{14}$$

■

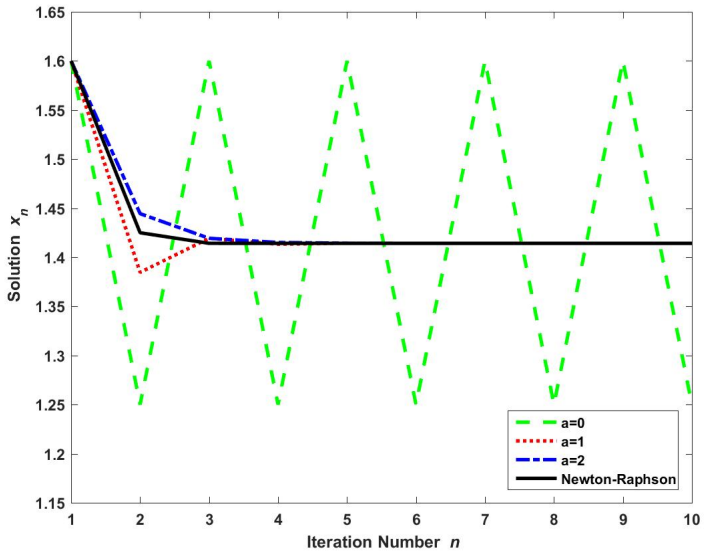


그림 5.4.3. Newton-Raphson법의 수렴속도

예제 5.4.6 Python을 사용해서 예제 5.4.5을 다시 다루기 위해서, 다음 Python프로그램 NewtonRaphson103.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

nmax = 9;
x = np.zeros((nmax+1,3));
a1 = np.arange(0,3)
for dumk in range(0,3):
    a1[dumk] = dumk;
    x[0,dumk] = 1.6;
    for n in range(0,nmax):
        x[n+1,dumk] = a1[dumk] + (2-a1[dumk]*a1[dumk])/(x[n,dumk]+a1[dumk]);

# Newton-Raphson method
xn = [0.0]*(nmax+1)
xn[0] = 1.6;
for n in range(0,nmax):
    xn[n+1] = ( xn[n] + 2/xn[n] )/2;

# Plotting
fig = plt.figure()
ii = np.arange(0,nmax+1)
plt.plot(ii,x[:,0],'g--',lw=2,label='a=0')
plt.plot(ii,x[:,1],'r:',lw=2,label='a=1')
plt.plot(ii,x[:,2],'b-.',lw=2,label='a=2')
plt.plot(ii,xn[:],'k',lw=2,label='Newton-Raphson')
plt.legend(loc='lower right', numpoints=1)
plt.ylabel('Solution x_{n}')
plt.xlabel('Iteration Number n')
plt.axis( [ 0, 9, 1.15, 1.65 ])
plt.show()
fig.savefig('NewtonRaphson103Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.4.5의 결과와 같다. ■

다음 예제는 MATLAB의 Symbolic Math Toolbox를 이용해서 Newton-Raphson법을 적용하는 예이다.

예제 5.4.7 방정식 $x^2 - 2 = 0$의 양근을 구하기 위해서, 다음 MATLAB프로그램 NewtonRaphson104.m을 실행하라.

```
% ------------------------------------------------------------------
%  Filename: NewtonRaphson104.m
%  Newton-Raphson method to solve a nonlinear equation
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, format rat
syms x
```

```
f = x^2 - 2
df = diff(f)
x = sym('1')
for ii=1:1:10
    ff = subs(f,x);
    dff = subs(df,x);
    xNew = x - ff/dff
    if abs(double(x-xNew)) < 10^-8,
        break
    end
    x = xNew;
end
format long
xnum = double(x),ii
save('NewtonRaphson104.txt','xnum','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 NewtonRaphson104.m에서는 Symbolic Math Toolbox를 사용해서 방정식 $x^2 - 2 = 0$의 양근을 구한다.

이 MATLAB프로그램을 실행하면, 다음과 같은 수열 $\{x_n\}$이 출력된다.

$$x_0 = 1, \quad x_1 = \frac{3}{2}, \quad x_2 = \frac{17}{12}, \quad x_3 = \frac{577}{408}, \quad x_4 = \frac{665857}{470832}, \quad x_5 = \frac{886731088897}{627013566048} \tag{1}$$

즉, 다음 식들이 성립한다.

$$x_5 = 1.414,213,562,374,690 \tag{2}$$

$$|x_5 - \sqrt{2}| < 1.6 \times 10^{-12} \tag{3}$$

■

다음 예제에서는 Newton-Raphson알고리즘에서 발생한 수열이 수렴하지 않고 진동하는 현상을 보여준다.

예제 5.4.8 다음과 같은 점화식을 살펴보자.

$$x_{n+1} - a = -[x_n - a], \quad (n = 1, 2, \cdots) \tag{1}$$

만약 초기값이 $x_1 = a$가 아니라면, 이 점화식에서 발생하는 수열 $\{x_n\}$은 진동한다. 이 수열 $\{x_n\}$이 다음 Newton-Raphson알고리즘을 만족한다고 가정하자.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad (n = 1, 2, \cdots) \tag{2}$$

식 (1)과 식 (2)에서 알 수 있듯이, 다음 점화식이 성립한다.

$$x_n - \frac{f(x_n)}{f'(x_n)} - a = -[x_n - a], \quad (n = 1, 2, \cdots) \tag{3}$$

따라서 함수 $f(x)$는 다음 미분방정식을 만족한다.

$$\frac{f'(x)}{f(x)} = \frac{1}{2[x-a]} \tag{4}$$

이 미분방정식의 해는 다음과 같다.

$$f(x) = \text{sign}(x-a)\sqrt{|x-a|} \tag{5}$$

즉, 식 (5)의 함수 $f(x)$에 대해서 점화식 (1)이 성립한다.

방정식 $\text{sign}(x-1)\sqrt{|x-1|} = 0$의 양근을 구하기 위해서, 다음 MATLAB프로그램 NewtonRaphsonPerverse101.m을 실행하라.

```
% ------------------------------------------------------------------
%  Filename: NewtonRaphsonPerverse101.m
%  A Perverse Example of Newton-Raphson method
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, clf
ftn = @(x) sign(x-1).*sqrt(abs(x-1))
dftn = @(x) sign(x-1).*sqrt(abs(x-1))/2/(x-1)
TOL = 10^-10; % toleration number
x = zeros(20,1);
format long;
% initial iteration
 x(1) = 0.6; % initial guess of the root
 xdiff = 0.1;
% iteration
for iter = 1:1:20;
    fval(iter) = feval(ftn,x(iter));
    dfval(iter) = feval(dftn,x(iter));
    xdiff = fval(iter)/dfval(iter);
    x(iter+1) = x(iter) - xdiff;
    if abs(xdiff) < TOL; break; end
end
x
% Ploting
hold on
xx = 0:0.01:2; yy = xx-2*(xx-1);
plot(xx,xx,'b-.',xx,yy,'k','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
for nn = 1:1:iter-1
    plot([x(nn),x(nn+1),x(nn+1),x(nn),x(nn)], ...
         [x(nn),x(nn),x(nn+1),x(nn+1),x(nn)],'r:','LineWidth',2)
end
axis square
```

```
text(0.3,0.62,'\bf (x_{n},x_{n})')
text(0.225,1.38,'\bf (x_{n},x_{n+1})')
text(1.4,0.62,'\bf (x_{n+1},x_{n})')
text(1.4,1.38,'\bf (x_{n+1},x_{n+1})')
text(0.22,0.22,'\bf y = x')
text(0.22,1.8,'\bf y = x - f''(x)/f(x)')
hold off
saveas(gcf,'NewtonRaphsonPerverse101','jpg')
% End of program
% --------------------------------------------------------------------
```

이 MATLAB프로그램 NewtonRaphsonPerverse101.m은 Newton-Raphson알고리즘을 적용해서 방정식 $\text{sign}(x-1)\sqrt{|x-1|}=0$의 근을 구하는 것이다.

이 MATLAB프로그램 NewtonRaphsonPerverse101.m을 실행하면, 그림 5.4.4가 그려진다. 그림 5.4.4에서 알 수 있듯이, 이 Newton-Raphson알고리즘은 진동한다. ■

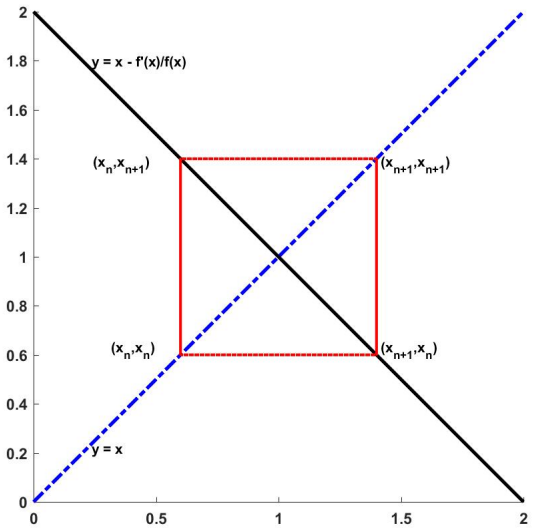


그림 5.4.4. NewtonRaphson수열의 진동

예제 5.4.9 Python을 사용해서 예제 5.4.8을 다시 다루기 위해서, 다음 Python프로그램 NewtonRaphsonPerverse101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

ftn = lambda x: np.sign(x-1)*np.sqrt(abs(x-1))
```

```
dftn = lambda x: np.sign(x-1)*np.sqrt(abs(x-1))/2/(x-1)
TOL = 10**(-10);                    # toleration number

x = [0.0]*21
# initial iteration
x[0] = 0.6;          # initial guess of the root
xdiff = 0.1;
fval = [0.0]*21
dfval = [0.0]*21
# iteration
for iter in range(0,20):
    fval[iter] = ftn(x[iter])
    dfval[iter] = dftn(x[iter])
    xdiff = fval[iter]/dfval[iter]
    x[iter+1] = x[iter] - xdiff
    if abs(xdiff) < TOL:
       break
print(x)

# Plotting
fig = plt.figure()
xx = np.arange(0,2.005,0.01)
yy = xx-2*(xx-1)
plt.plot(xx,xx,'b-.',xx,yy,'k',lw=2)
for nn in range(0,iter):
    xdum1 = [ x[nn], x[nn+1], x[nn+1], x[nn], x[nn] ]
    ydum1 = [ x[nn], x[nn], x[nn+1], x[nn+1], x[nn] ]
    plt.plot(xdum1, ydum1, 'r:', lw=2)
plt.axis( [ 0.0, 2.0, 0.0, 2.0 ])
plt.axis('equal')
plt.text(0.24,0.62, r'$(x_{n},x_{n})$')
plt.text(0.18,1.38, r'$(x_{n},x_{n+1})$')
plt.text(1.45,0.62, r'$(x_{n+1},x_{n})$')
plt.text(1.45,1.38, r'$(x_{n+1},x_{n+1})$')
plt.text(0.28,0.22, r'$y = x$')
plt.text(0.24,1.8, r'y = x - f''(x)/f(x)$')
plt.show()
fig.savefig('NewtonRaphsonPerverse101Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.4.8의 결과와 같다. ■

제5.5절 시컨트법

Newton-Raphson법을 적용하려면, 함수 $f(x)$의 도함수 $f'(x)$를 알아야 한다. 이것이 Newton-Raphson법의 가장 큰 약점이다. 만약 $f'(x)$가 존재하지 않거나 너무 복잡하다면, 이 도함수 대신 평균변화율을 적용하는 다음 점화식을 사용할 수 있다.

$$x_{n+2} = x_{n+1} - \frac{f(x_{n+1})}{f(x_{n+1}) - f(x_n)}[x_{n+1} - x_n], \quad (n = 1, 2, \cdots) \tag{5.5.1}$$

이 경우에, 함수 $f(x)$는 근 $x = \alpha$ 주변에서 연속이어야 한다. 점화식 (5.5.1)을 적용하는 방법을 시컨트법 (secant method) 또는 가상위치법 (regula falsi method)이라 부른다.

예제 5.5.1 방정식 $x^2 - 2 = 0$의 양근을 구하기 위해서, 다음 MATLAB프로그램 Secant-Method101.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: SecantMethod101.m
%  Secant method to solve a nonlinear equation 1
%  Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
nmax = 9;
% Newton-Raphson method
xn(1) = 1.6;
for n = 1:nmax
        xn(n+1) = ( xn(n) + 2/xn(n) )/2;
end
% Secant method
xs(1) = 1.6; xs(2) = 1.5;
for n = 1:nmax-1
        xs(n+2) = xs(n+1) - (xs(n+1)^2 -2)/(xs(n+1)+ xs(n) );
end
% Plotting
ii = (1:1:nmax+1);
plot(ii,xs,'k',ii,xn,'r--','LineWidth',2.0);
set(gca,'fontsize',11,'fontweigh','bold')
legend('Secant','Newton-Raphson','location','SE')
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\bf Solution \it x_{n}','fontsize',12)
axis( [ 1, 10, 1.3 1.65 ])
saveas(gcf,'SecantMethod101','jpg')
save('SecantMethod101.txt','xn','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 SecantMethod1.m을 실행하면, Newton-Raphson법과 시컨트법을 적용한 결과인 그림 5.5.1이 그려진다. 여기서 시컨트법에 의한 점화식은 다음과 같다.

$$x_{n+2} = x_{n+1} - \frac{x_{n+1}^2 - 2}{x_{n+1} + x_n}, \quad (n = 1, 2, \cdots) \tag{1}$$

이 MATLAB프로그램에서는 초기값들 $x_1 = 1.6$과 $x_2 = 1.5$을 사용하였다. 그림 5.5.1에서 알 수 있듯이, 이 경우에는 시컨트법에서 출력된 값은 급속히 Newton-Raphson알고리즘의 값과 비슷해진다. ■

예제 5.5.2 Python을 사용해서 예제 5.5.1을 다시 다루기 위해서, 다음 Python프로그램 SecantMethod101.Py를 실행해 보자.

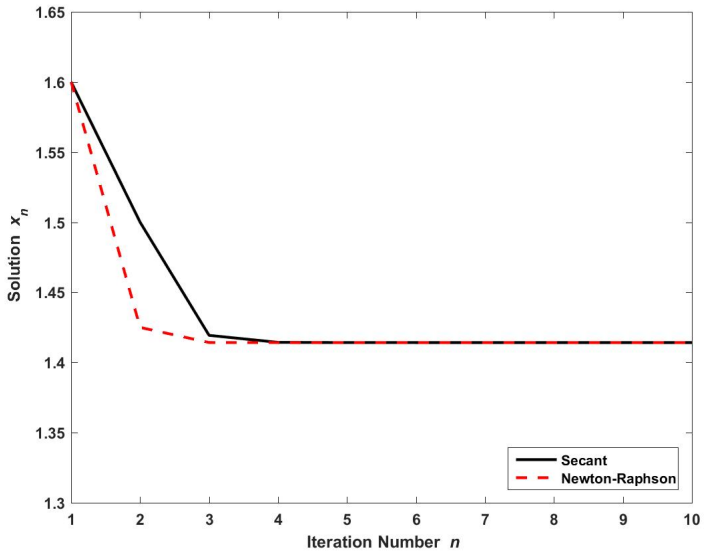


그림 5.5.1. Newton-Raphson과 할선법

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

nmax = 9;
# Newton-Raphson method
xn = [0.0]*(nmax+1)
xn[0] = 1.6;
for n in range(0,nmax):
    xn[n+1] = ( xn[n] + 2.0/xn[n] )/2.0;

# Secant method
xs = [0.0]*(nmax+1)
xs[0] = 1.6; xs[1] = 1.5;
for n in range(0,nmax-1):
    xs[n+2] = xs[n+1] - (xs[n+1]**2 -2)/(xs[n+1] + xs[n] );

# Plotting
ii = np.arange(0,nmax+1);
fig = plt.figure()
plt.plot(ii,xs,'k', lw=2, label='Secant')
plt.plot(ii,xn,'r--', lw=2, label='Newton-Raphson')
plt.xlabel('Iteration Number n'); plt.ylabel('Solution x_{n}')
plt.legend(loc='lower right', numpoints=1)
plt.axis([ 0, 9, 1.3, 1.65 ])
plt.show()
fig.savefig('SecantMethod101Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.5.1의 결과와 같다. ■

만약 적절한 초기값을 선택해서 시컨트법이 성공적으로 수행된다면, 오차항들 $\{e_n\}$은 다음 식을 만족한다.

$$e_{n+1} = O\left(e_n^{\theta}\right) \tag{5.5.2}$$

여기서 $\theta = [\sqrt{5}+1]/2$이다. 그러나, Newton-Raphson법과 마찬가지로 시컨트법에서도 초기값을 근 α에 가깝게 선택하지 않으면, 점화식에서 발생된 수열이 수렴하지 않는 경우가 있다.

제5.6절 Müller법

지금까지 다룬 고정점법은 각 단계에서 함수 $f(x)$를 1차식으로 근사시키는 방법을 사용하였다. 우리는 2차방정식을 푸는 근의 공식을 알고 있으므로, 함수 $f(x)$를 2차식으로 근사시키는 방법을 사용할 수도 있을 것이다. 이러한 방법을 Müller법 또는 2차내삽법(quadratic interpolation method)이라 부른다.

Müller법을 설명하기 위해서, 우선 내림차순 또는 올림차순으로 배열된 x_0, x_1, x_2가 지나는 점들 $(x_0, f(x_0)), (x_1, f(x_1)), (x_2, f(x_2))$를 생각해보자. 이 세 점들을 지나는 2차함수 $P(x)$를 다음과 같이 표기하자.

$$P(x) = a[x - x_2]^2 + b[x - x_2] + c \tag{5.6.1}$$

다음 식들이 성립한다.

$$f(x_0) = P(x_0) = a[x_0 - x_2]^2 + b[x_0 - x_2] + c \tag{5.6.2}$$

$$f(x_1) = P(x_1) = a[x_1 - x_2]^2 + b[x_1 - x_2] + c \tag{5.6.3}$$

$$f(x_2) = P(x_2) = a[x_2 - x_2]^2 + b[x_2 - x_2] + c \tag{5.6.4}$$

연립방정식 (5.6.2)~(5.6.4)를 풀면, 다음 식들이 성립함을 알 수 있다.

$$c = f(x_2) \tag{5.6.5}$$

$$b = \frac{[x_0 - x_2]^2[f(x_1) - f(x_2)] - [x_1 - x_2]^2[f(x_0) - f(x_2)]}{[x_0 - x_2][x_1 - x_2][x_0 - x_1]} \tag{5.6.6}$$

$$a = \frac{[x_1 - x_2]^2[f(x_0) - f(x_2)] - [x_0 - x_2]^2[f(x_1) - f(x_2)]}{[x_0 - x_2][x_1 - x_2][x_0 - x_1]} \tag{5.6.7}$$

방정식 $P(x_3) = 0$의 근이 2개 존재한다. 그 중에서 x_2에 가까운 근이 우리가 원하는 것이다. 비슷한 숫자를 뺄 때 발생하는 반올림오차를 피하기 위해서, 다음과 같은 근의 공식을 적용하자.

$$x_3 = x_2 + \frac{-2c}{b \pm \sqrt{b^2 - 4ac}} \tag{5.6.8}$$

이 근들 중에서 x_2에 가까운 근 x_3는 다음과 같다.

$$x_3 = x_2 + \frac{-2c}{b + \text{sign}(b)\sqrt{b^2 - 4ac}} \tag{5.6.9}$$

예제 5.6.1 Müller법을 직관적으로 이해하기 위해서, 다음 MATLAB프로그램 MullerMethodGraph101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: MullerMethodGraph101
% Graph explaining Muller method
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
ftn = @(x) x.^3 - x.^2 - x + 0.7
roots(1) = fzero(ftn,1.5);
roots(2) = fzero(ftn,0.5)
ftn(roots)
x = [ 0.5 0.8 1.2 ]';
fx = ftn(x);
Van = vander( x - x(2) )
coef = Van\ftn(x)
xx = linspace(0.2,1.5,201);
ff = ftn(xx);
P = coef(3)*1 + coef(2)*(xx-x(2)) + coef(1)*(xx-x(2)).^2;
plot(xx,ff,'b--','LineWidth',1.5);
hold on
plot(xx,P,'r-','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0.2,1.5])
plot(x,fx,'ko','LineWidth',2);
plot(1.42,0,'bdiamond','LineWidth',2);
plot(roots,roots-roots,'r*','LineWidth',1.5);
legend('\bf True','\bf Quadratic','\bf Given Points', ...
       '\bf New Point','\bf True roots','location','NE')
plot(xx,xx-xx,'g:','LineWidth',1.5);
xlabel('\bf \it x','fontsize',12)
ylabel('\bf \it f(x)','fontsize',12,'rotation',0)
hold off
saveas(gcf,'MullerMethodGraph101','jpg')
save('MullerMethodGraph101.txt','coef','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 MullerMethodGraph101.m을 실행하면, 그림 5.6.1이 그려진다. 이 예제의 경우에는 Müller법에 의한 근이 진짜 근에서 떨어져 있다. ■

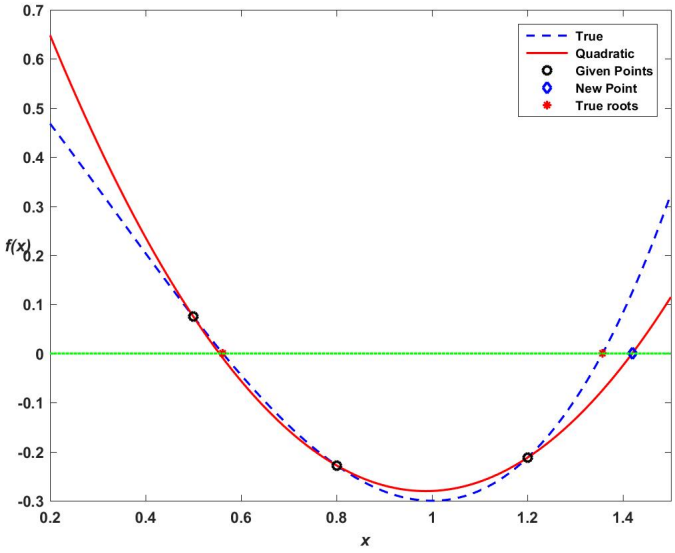


그림 5.6.1. 근사 2차함수

예제 5.6.2 Python을 사용해서 예제 5.6.1을 다시 다루기 위해서, 다음 Python프로그램 MullerMethodGraph101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

ftn = lambda x: x**3 - x**2 - x + 0.7
root1 = fsolve(ftn, 1.5);
print(root1, ftn(root1))
root2 = fsolve(ftn, 0.5);
print(root2, ftn(root2))
x = np.array([ 0.5, 0.8, 1.2 ]);
print(x)
fx = ftn(x);
print(fx)
Van = np.vander( x - x[1] );
coef = np.matmul(np.linalg.inv(Van),ftn(x));
xx = np.linspace(0.2,1.5,201);
ff = ftn(xx);

# Plotting
P = coef[2]*1 + coef[1]*(xx-x[1]) + coef[0]*(xx-x[1])**2;
fig = plt.figure()
plt.plot(xx,ff,'b--',lw=2,label='True');
plt.plot(xx,P,'r-',lw=2,label='Quadratic')
plt.plot(x,fx,'ko',lw=2,label='Given Points')
plt.plot([1.42],[0],'bd',lw=2,label='New Point')
roots = np.array([root1, root2])
plt.plot(roots,roots-roots,'r*',lw=2,label='True roots')
```

```
plt.legend(loc='upper right', numpoints=1)
plt.plot(xx,xx-xx,'g:',lw=2)
plt.xlabel('x'); plt.ylabel('f(x)');
plt.axis([ 0.2, 1.5, -0.4, 0.8 ])
plt.show()
fig.savefig('MullerMethodGraph101Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.6.1의 결과와 같다. ■

예제 5.6.3 Müller법을 적용해서 방정식 $x^3 - x^2 - x + 1 = 0$를 풀기 위해서, 다음 MATLAB프로그램 MullerMethod101.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: MullerMethod101.m
% Muller Method
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
fun = inline('z.^3 -z.^2 -z +1','z')
% Initialization w/ Newton-Raphson method
tol = 1.0e-10;
xInitial = 9;
x(1) = xInitial;
for ii=1:1:2
    dff = ( fun(x(ii)+0.0001)-fun(x(ii)-0.0001) )/0.0002;
    fxp(ii) = fun(x(ii));
    x(ii+1) = x(ii) - fxp(ii)/dff;
end

% Iteration
xp = x;
fxp(3) = fun(x(3));
dxp43(1) = abs(x(2)-x(1));
for ii = 2:1:40
    [xp,Ixp] = sort(xp);
    dxp12 = xp(1)-xp(2);
    dxp32 = xp(3)-xp(2);
    fxp = fxp(Ixp);
    c = fxp(2);
    f12 = fxp(1)-c;
    f32 = fxp(3)-c;
    DD = max(dxp12*dxp32*(xp(1)-xp(3)),eps);
    a = (dxp32*f12-dxp12*f32)/DD;
    b = (dxp12*dxp12*f32-dxp32*dxp32*f12)/DD;
    dxp43(ii) = -sign(b)*2*c/(abs(b)+sqrt(max(b*b-4*a*c,0)));
    x(ii+1) = xp(2) + dxp43(ii)
    if dxp43(ii) >= tol
        xp(1) = x(ii+1); fxp(1) = fun(x(ii+1));
    elseif dxp43(ii) <= -tol
        xp(3) = x(ii+1); fxp(3) = fun(x(ii+1));
    else
        break
    end
end
```

```
dxp43, ii, x
% Plotting
nn = (1:1:ii+1);
plot([ 0 ii+1],[x(ii+1) x(ii+1)],'b--',nn,x,'r', ...
      nn,x,'ko','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\it x_{n}','fontsize',12,'rotation',0)
axis( [ 1, ii+1, 0.5*x(ii+1) 1.1*x(1) ])
saveas(gcf,'MullerMethod101','jpg')
save('MullerMethod101.txt','x','-ascii')
% End of program
% --------------------------------------------------------------------
```

이 MATLAB프로그램 MullerMethod101.m에서는 초기값들 x_0, x_1, x_2를 지정하지 않고 초기값 $x_0 = 9$만 지정하였다. 그 이유는 주어진 방정식이 점 $(1, 0)$에서 중근을 갖기때문에 초기값들을 잘 선택하지 않으면 이 알고리즘이 잘 수렴하지 않기 때문이다. 따라서, Newton-Raphson법을 사용해서 x_1과 x_2를 선택한 다음, 이 값들을 초기값들로 해서 Müller법을 적용하였다.

이 MATLAB프로그램 MullerMethod101.m을 실행하면, 다음 부등식이 성립할 때 이 점화식의 계산을 멈춘다.

$$|x_{21} - x_{20}| < 10^{-10} \tag{1}$$

이 경우에 해는 $x_{21} = 1.000$이다. 또한, 이 MATLAB프로그램을 실행하면, 그림 5.6.2가 그려진다. 그림 5.6.2에서 알 수 있듯이, 초기값들만 잘 설정된다면 Müller법에서 출력된 수열 $\{x_n\}$은 아주 빠른 속도로 수렴한다. ■

예제 5.6.4 Python을 사용해서 예제 5.6.3을 다시 다루기 위해서, 다음 Python프로그램 MullerMethod101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

fun = lambda z: z**3 - z**2 - z + 1.0
# Initialization w/ Newton-Raphson method
x = []
fxp = []
dxp43 = []
tol = 1.0e-10;
xInitial = 9;
x.append(xInitial)
```

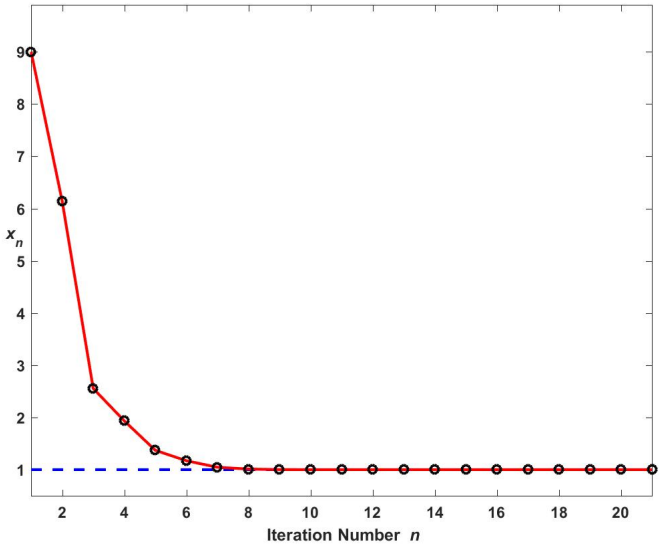


그림 5.6.2. Müller법 I

```
xp = [0.0]*3
for ii in range(0,2):
    dff = ( fun(x[ii]+0.0001)-fun(x[ii]-0.0001) )/0.0002;
    fxp.append(fun(x[ii]))
    dumx = x[ii] - fxp[ii]/dff;
    x.append(dumx)

# Iteration
epss = 7/3 - 4/3 -1;
xp = x
fxp.append(fun(x[2]));
dxp43.append(abs(x[1] - x[0]));

for ii in range(1,40):
    Ixp = np.argsort(xp)
    xp = np.sort(xp)
    dxp12 = xp[0]-xp[1];
    dxp32 = xp[2]-xp[1];
    Ixpp = np.ndarray.tolist(Ixp)
    nIxpp = len(Ixpp)
    fxpDum = [0.0]*nIxpp
    for jjj in range(0,nIxpp):
        fxpDum[jjj] = fxp[Ixpp[jjj]]
    for jj in range(0,nIxpp):
        fxp[jj] = fxpDum[jj];
    c = fxp[1];
    f12 = fxp[0]-c;
    f32 = fxp[2]-c;
    DD = max(dxp12*dxp32*(xp[0]-xp[2]),epss)
    a = (dxp32*f12-dxp12*f32)/DD;
    b = (dxp12*dxp12*f32-dxp32*dxp32*f12)/DD
    dxp43Dum = -np.sign(b)*2*c/(abs(b)+np.sqrt(max(b*b-4*a*c,0)))
    dxp43.append(dxp43Dum)
    if ii==1:
        x[2] = xp[1] + dxp43[1];
    else:
        xDum2 =  xp[1] + dxp43[ii];
```

```
        x.append(xDum2);
    if dxp43[ii] >= tol:
        xp[0] = x[ii+1];
        fxp[0] = fun(x[ii+1]);
    elif dxp43[ii] <= -tol:
        xp[2] = x[ii+1];
        fxp[2] = fun(x[ii+1]);
    else:
        break

print(dxp43);
print(ii);
print(x)

# Plotting
nn = np.arange(0,ii+2)
fig = plt.figure()
plt.plot([0,ii+1],[x[ii+1],x[ii+1]],'b--',lw=2)
plt.plot(nn,x,'r', nn,x,'ko', lw=2)
plt.xlabel('Iteration Number n')
plt.ylabel('x_{n}')
plt.axis( [ 0, ii, 0.5*x[ii], 1.1*x[0] ])
plt.show()
fig.savefig('MullerMethod101Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 5.6.3의 결과와 같다. ■

Vandermode행렬을 사용해서 연립방정식 (5.6.2)~(5.6.4)를 다음과 같이 쓸 수 있다.

$$\begin{bmatrix} c \\ b \\ a \end{bmatrix} = \begin{bmatrix} 1 & [x_0 - x_2] & [x_0 - x_2]^2 \\ 1 & [x_1 - x_2] & [x_1 - x_2]^2 \\ 1 & [x_1 - x_2] & [x_2 - x_2]^2 \end{bmatrix}^{-1} \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix} \tag{5.6.10}$$

식 (5.6.10)에서 알 수 있듯이 사용하는 컴퓨터프로그램이 Vandermonde행렬의 역행렬을 구하는 함수를 포함하고 있으면, 쉽게 Müller법을 적용할 수 있다. 다음 예제에서는 이 방법을 사용한다.

예제 5.6.5 Müller법을 적용해서 방정식 $x^3 - x^2 - x + 1 = 0$를 풀기 위해서, 다음 MATLAB프로그램 MullerMethod102.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: MullerMethod102.m
% Muller Method w/ Vandermonde matrix
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
ftn = @(z) z.^3 - z.^2 - z + 1
```

```
tol_x = 1e-10;
tol_y = 1e-10;
x = [ 0.5 0.8 1.2 ]';   % Initial ( x0, x1, x2 )
fx = ftn(x);
iterNo = 0;
xx = x;
while ( true )
    iterNo = iterNo +1;
    Van = vander( x - x(2) );
    coef = Van\ftn(x);
    D = sqrt( coef(2)^2 - 4*coef(1)*coef(3) );
    % if ( real(c(2))*real(disc) + imag(c(2))*imag(disc) > 0 )
    if abs( coef(2) + D ) > abs( coef(2) - D )
        denom = coef(2) + D;
    else
        denom = coef(2) - D;
    end
    x = [x(2), x(3), x(2) - 2*coef(3)/denom]';
    fx = ftn(x);
    if abs( x(2) - x(3) ) < tol_x && abs( fx(3) ) < tol_y
        break;
    end
     xx(iterNo+3) = x(3);
end
xx
iter = 1:iterNo+2;
plot(iter-1,xx(iter),'k-o','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold','ylim',[0.4 1.3])
xlabel('\bf \it n','fontsize',12)
ylabel('\it x_{n}','fontsize',12,'rotation',0)
saveas(gcf,'MullerMethod102','jpg')
save('MullerMethod102.txt','x','-ascii')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 MullerMethod102.m에서는 초기값들로 $x_0 = 0.5, x_1 = 0.8, x_2 = 1.2$를 지정하였다. 이 초기값들은 여러번 실험을 해서 좋은 결과를 내는 값을 선택한 것이다.

이 MATLAB프로그램 MullerMethod102.m을 실행하면, 다음 부등식이 성립할 때 이 점화식의 계산을 멈춘다.

$$|x_{15} - x_{14}| < 10^{-10} \tag{1}$$

이 경우에 해는 $x_{14} = 1.000$이다. 또한, 이 MATLAB프로그램을 실행하면, 그림 5.6.3이 그려진다. ■

제5.7절 IQI법

시컨트법은 두 점들을 이용해서 다음 점을 구하는 방법이고, 역이차내삽법(inverse quadratic interpolation method: IQI method)은 세 점들을 이용해서 다음 점을 구하는 방법이다. 또한

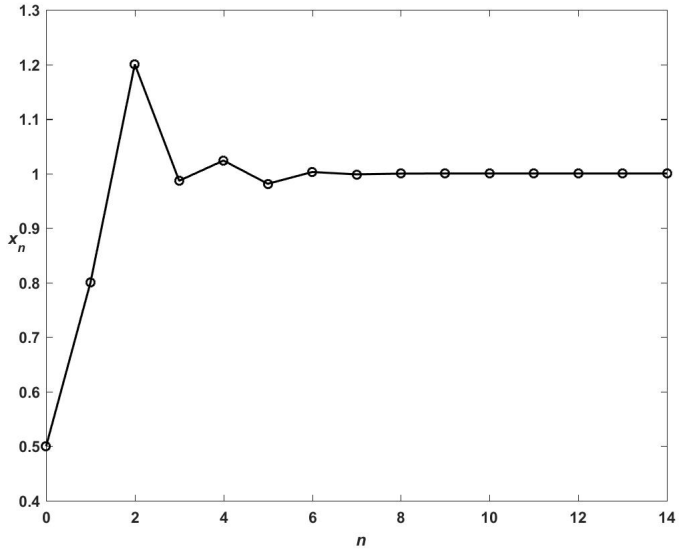


그림 5.6.3. Müller법 II

역이차내삽법은 2차함수의 역함수를 사용해서 방정식 $f(x) = 0$의 근을 구하는 점화식을 만드는 방법이다. 이 점화식으로부터 생성된 수열을 $\{x_n\}$이라 하고, $f_n \doteq f(x_n)$으로 표기하자. 서로 다른 점들 (x_{n-2}, f_{n-2}), (x_{n-1}, f_{n-1}) , (x_n, f_n)을 지나는 역이차내삽식 $g(y)$를 다음과 같이 표기하자.

$$g(y) = a[y - f_n]^2 + b[y - f_n] + c \tag{5.7.1}$$

다음 식들이 성립한다.

$$x_{n-2} = g(f_{n-2}) = a[f_{n-2} - f_n]^2 + b[f_{n-2} - f_n] + c \tag{5.7.2}$$

$$x_{n-1} = g(f_{n-1}) = a[f_{n-1} - f_n]^2 + b[f_{n-1} - f_n] + c \tag{5.7.3}$$

$$x_n = g(f_n) = a[f_n - f_n]^2 + b[f_n - f_n] + c \tag{5.7.4}$$

제5.6절에서는 연립방정식 (5.6.2)∼(5.6.4)를 Gauss소거법으로 풀거나 Vandermonde행렬을 사용해서 풀었다. 여기서는 연립방정식 (5.7.2)∼5.7.3)를 푸는 대신에 다음과 같은 Lagrange 내삽식을 사용하기로 하자.

$$\begin{aligned} h(y) \doteq & \frac{[y - f_{n-1}]\,[y - f_n]}{[f_{n-2} - f_{n-1}]\,[f_{n-2} - f_n]} x_{n-2} + \frac{[y - f_{n-2}]\,[y - f_n]}{[f_{n-1} - f_{n-2}]\,[f_{n-1} - f_n]} x_{n-1} \\ & + \frac{[y - f_{n-2}]\,[y - f_{n-1}]}{[f_n - f_{n-2}]\,[f_n - f_{n-1}]} x_n. \end{aligned} \tag{5.7.5}$$

다음 식들이 성립함은 명백하다.

$$h\left(f_{n-2}\right)=x_{n-2},\quad h\left(f_{n-1}\right)=x_{n-1},\quad h\left(f_{n}\right)=x_{n} \tag{5.7.6}$$

함수 $g(y)$와 함수 $h(y)$는 서로 다른 점들 (x_{n-2},f_{n-2}), (x_{n-1},f_{n-1}) , (x_n,f_n)을 지나는 2차 함수들이므로 $h(y)=g(y)$이다. 따라서, 우리가 구하고자 하는 점 $(x_{n+1},0)$은 함수 $x=g(y)$ 상에 존재한다. 즉, 식 $x_{n+1}=g(0)$을 만족하는 x_{n+1}을 구하고자 한다. 식 (5.7.5)에서 알 수 있듯이, 다음 식이 성립한다.

$$\begin{aligned} x_{n+1}=&\frac{f_{n-1}f_n}{\left[f_{n-2}-f_{n-1}\right]\left[f_{n-2}-f_n\right]}x_{n-2}+\frac{f_{n-2}f_n}{\left[f_{n-1}-f_{n-2}\right]\left[f_{n-1}-f_n\right]}x_{n-1}\\ &+\frac{f_{n-2}f_{n-1}}{\left[f_n-f_{n-2}\right]\left[f_n-f_{n-1}\right]}x_n. \end{aligned} \tag{5.7.7}$$

IQI법을 직관적으로 이해하기 위해서 예제를 살펴보자.

예제 5.7.1 기하학적으로 IQI법을 이해하기 위해서, 다음 MATLAB프로그램 IQImethod101.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename: IQImethod101.m
% Graph explaining IQI method
% Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, format long
ftn = @(x) x.^3 - x.^2 - x + 0.7
roots(1) = fzero(ftn,1.5);
roots(2) = fzero(ftn,0.5)
x = [ 0.5 0.8 1.2 ]';
fx = ftn(x);
ff = linspace(-0.4,0.8,1001);
xx = x(1)/(fx(1)-fx(2))/(fx(1)-fx(3))*(ff-fx(2)).*(ff-fx(3)) ...
    +x(2)/(fx(2)-fx(1))/(fx(2)-fx(3))*(ff-fx(1)).*(ff-fx(3)) ...
    +x(3)/(fx(3)-fx(1))/(fx(3)-fx(2))*(ff-fx(1)).*(ff-fx(2));
hold on
plot(xx,ftn(xx),'b--','LineWidth',1.5);
plot(xx,ff,'r-','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold')
plot(x,fx,'ko','LineWidth',2);
plot(2.12,0,'bdiamond','LineWidth',2);
plot(roots,roots-roots,'r*','LineWidth',1.5);
legend('\bf True','\bf IQI','\bf Given Points', ...
       '\bf New Point','\bf True roots','location','NE')
plot([ -10 10 ],[ 0 0 ],'g:','LineWidth',1.5);
xlabel('\bf \it x','fontsize',12)
ylabel('\bf \it f(x)','fontsize',12,'rotation',0)
axis([ 0 3 -0.4 0.3 ])
hold off
saveas(gcf,'IQImethod101','jpg')
```

```
save('IQImethod101.txt','xx','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 IQImethodGraph101.m을 실행하면, 그림 5.7.1이 그려진다. 이 그림에서는 원리를 강조하기 위해서 역이차함수의 꼭지점이 원래 함수에서 많이 떨어져 있으나, 실제 IQI알고리즘을 실행하면 역이차함수의 꼭지점이 원래 함수에 가까워진다. 그렇지 않으면, 주어진 문제에 IQI법은 적당하지가 않다. ■

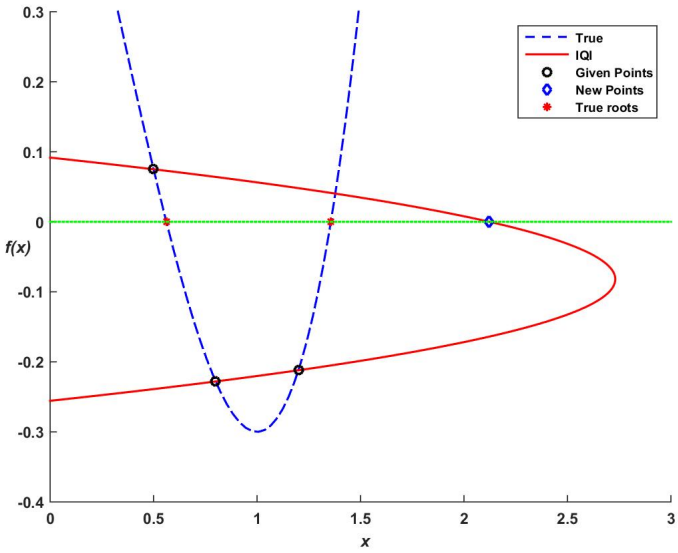


그림 5.7.1. 역이차함수

예제 5.7.2 Python을 사용해서 예제 5.7.1을 다시 다루기 위해서, 다음 Python프로그램 IQIMethod101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

ftn = lambda x: x**3 - x**2 - x + 0.7
root1 = fsolve(ftn, 1.5);
print(root1, ftn(root1))
root2 = fsolve(ftn, 0.5);
print(root2, ftn(root2))
x = np.array([ 0.5, 0.8, 1.2 ]);
fx = ftn(x);
```

```
ff = np.linspace(-0.4,0.8,1001);
xx1 = x[0]/(fx[0]-fx[1])/(fx[0]-fx[2])*(ff-fx[1])*(ff-fx[2])
xx2 = x[1]/(fx[1]-fx[0])/(fx[1]-fx[2])*(ff-fx[0])*(ff-fx[2])
xx3 = x[2]/(fx[2]-fx[0])/(fx[2]-fx[1])*(ff-fx[0])*(ff-fx[1])
xx = xx1 + xx2 + xx3

# Plotting
fig = plt.figure()
plt.plot(xx,ftn(xx),'b--',lw=2,label='True')
plt.plot(xx,ff,'r-',lw=2,label='IQI')
plt.plot(x,fx,'ko',lw=2,label='Given Points')
plt.plot([2.12],[0],'bd',lw=2,label='New Point')
roots = np.array([root1, root2])
plt.plot(roots,roots-roots,'r*',lw=2,label='True roots')
plt.legend(loc='upper right', numpoints=1)
plt.plot([ -10, 10 ],[ 0, 0 ],'g:',lw=2)
plt.xlabel('x'); plt.ylabel('f(x)');
plt.axis([ 0, 3, -0.4, 0.3 ])
plt.show()
fig.savefig('IQImethod101Py.jpg')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 **??**의 결과와 같다. ■

예제 5.7.3 IQI법을 적용해서 방정식 $x^3 - x^2 - x + 1 = 0$를 풀기 위해서, 다음 MATLAB 프로그램 IQImethod102.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename: IQImethod102.m
% Inverse Quadratic Method
% Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, format long
ftn = @(x) x.^3 - x.^2 - x + 0.7
tol = 10^(-10);
roots(1) = fzero(ftn,1.5);
roots(2) = fzero(ftn,0.5)
x = [ 0.5 0.8 1.2 ]';
xx = x;
fx = ftn(x);
for ii=1:100
    xnew = x(1)/(fx(1)-fx(2))/(fx(1)-fx(3))*fx(2)*fx(3) ...
          +x(2)/(fx(2)-fx(1))/(fx(2)-fx(3))*fx(1)*fx(3) ...
          +x(3)/(fx(3)-fx(1))/(fx(3)-fx(2))*fx(1)*fx(2);
    xx(ii+3) = xnew;
    if abs(x(3)-xnew) < tol
        break
    end
    x = [ x(2) x(3) xnew ];
    fx = ftn(x);
end
nn = (1:1:ii+3)';
[ nn xx ]
% Plotting
plot(nn-1,xx,'r',nn-1,xx,'ko','LineWidth',2);
```

```
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\it x_{n}','fontsize',12,'rotation',0)
saveas(gcf,'IQImethod102','jpg')
save('IQImethod102.txt','xx','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 IQImethod102.m에서는 초기값들로 $x_0 = 0.5, x_1 = 0.8, x_2 = 1.2$를 지정하였다.

이 MATLAB프로그램 IQImethod102.m을 실행하면, 다음 부등식이 성립할 때 이 점화식의 계산을 멈춘다.

$$|x_{13} - x_{12}| < 10^{-10} \tag{1}$$

이 경우에 해는 $x_{12} = 1.3568$이다. 또한, 이 MATLAB프로그램을 실행하면, 그림 5.7.2가 그려진다. ■

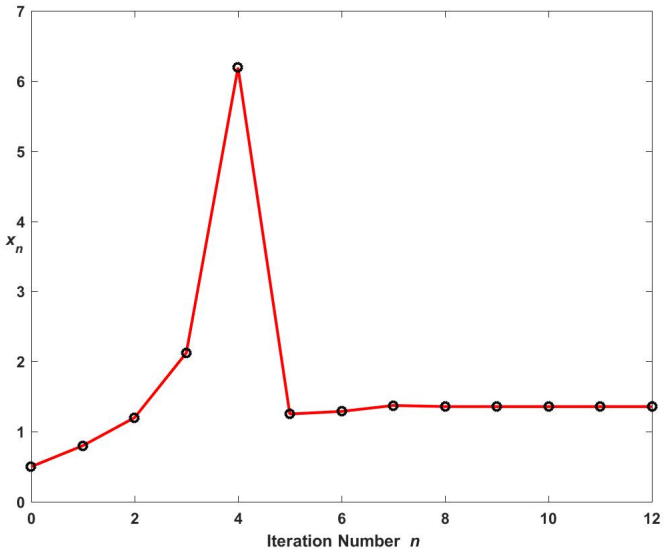


그림 5.7.2. IQI법

예제 5.7.4 Python을 사용해서 예제 5.7.3을 다시 다루기 위해서, 다음 Python프로그램 IQImethod102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

ftn = lambda x: x**3 - x**2 - x + 0.7
tol = 10**(-10)
root1 = fsolve(ftn, 1.5);
print(root1, ftn(root1))
root2 = fsolve(ftn, 0.5);
print(root2, ftn(root2))
x = np.array([ 0.5, 0.8, 1.2 ]);
xx = [];
xx.append(0.5); xx.append(0.8); xx.append(1.2);

fx = ftn(x);
for ii in range(0,100):
    xnew1 = x[0]/(fx[0]-fx[1])/(fx[0]-fx[2])*fx[1]*fx[2]
    xnew2 = x[1]/(fx[1]-fx[0])/(fx[1]-fx[2])*fx[0]*fx[2]
    xnew3 = x[2]/(fx[2]-fx[0])/(fx[2]-fx[1])*fx[0]*fx[1]
    xnew = xnew1 + xnew2 + xnew3
    xx.append(xnew)
    if abs(x[2]-xnew) < tol:
        break
    x[0] = x[1]; x[1] = x[2]; x[2] =xnew;
    fx = ftn(x);

# Plotting
ii
nn = np.arange(0,ii+4)
print(nn); print(xx)
fig = plt.figure()
plt.plot(nn,xx,'r',nn,xx,'ko',lw=2)
plt.xlabel('x'); plt.ylabel(r'$x_{n}$');
plt.show()
fig.savefig('IQImethod102Py.jpg')

# End of program
```

이 Python프로그램을 수행한 결과는 예제 5.7.3의 결과와 같다. ■

Moler (2004)에 의하면 IQI법은 노련하지 못한 경주마(immature race horse)와 같다고 한다. 이러한 말은 결승선에 가까이 가면 매우 빨리 뛰지만, 레이스 전반에 걸쳐서 보면 불규칙한(erratic) 행태를 보인다.

제5.8절 Brent법

Brent법은 Dekker (1969)가 제시한 방법을 발전시킨 것으로서, 이분할법, 시컨트법 그리고 IQI법을 사용해서 비선형방정식의 해를 구하는 방법이다. 즉, Brent법은 이분할법의 강건성(robustness) 그리고 시컨트법과 IQI법의 속도를 이용한 방법으로서, 어떠한 경우에도 해를

제시하는 잘못될 수가 없는(fool-proof) 방법이다. MATLAB 함수 fzero.m에서는 Brent 법을 적용해서 비선형방정식의 해를 구한다. Brent 법을 요약하면, 다음과 같다.

알고리즘 5.8.1

만약 함수 $y = f(x)$가 폐구간 $[a, b]$에서 연속이면, 다음과 같은 단계들을 거쳐서 방정식 $f(x) = 0$의 근 x^*를 구한다.

(1단계) 주어진 구간 $[a, b]$에 대해 시컨트법을 사용해서 $f(c) = 0$인 점 c를 찾는다.

(2단계) 만약 식 $|a - b| < \epsilon$ 또는 식 $f(c) > 0$가 성립하면, $x^* = c$라 하고 알고리즘을 멈춘다. 그렇지 않으면, 제3단계로 넘어간다. 여기서 ϵ은 미리 정한 작은 양수이다.

(3단계) 만약 식 $f(a)f(b) < 0$가 성립하면, 다른 a와 b를 선택한다.

(4단계) 만약 식 $|f(a)| < |f(b)|$가 성립하면, a와 b를 맞바꾼다. 또한, 전단계의 b를 c로 할당한다.

(5단계) 만약 c와 a가 같으면, 시컨트법을 사용해서 새로운 c를 구한다. 그렇지 않으면, IQI법을 적용해서 새로운 c를 구한다.

(6단계) 만약 c가 구간 $[a, b]$에 속하면, 제2단계로 넘어간다. 그렇지 않으면, 이분할법을 사용해서 새로운 c를 구한 다음 제2단계로 넘어간다.

예제 5.8.1 Brent 법을 적용해서 방정식 $x^3 - x^2 - x + 1 = 0$를 풀기 위해서, 다음 MATLAB 프로그램 BrentDekker101.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: BrentDekker101.m
%  Brent-Dekker Method
%  To find a solution x of a nonlinear equation
% ------------------------------------------------------------------------
clear all, close all, format long
f = @(x) x.^3-x.^2-x+1
for ii = 1:1:40
    initial(ii) = 0.25*ii - 3;
    sol(ii) = fzero(f,initial(ii));
end
[ initial; sol]'
% Plotting
plot(initial,sol,'k-',initial,sol,'rd','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf Initial Value','fontsize',12)
ylabel('\bf Solution','fontsize',12)
```

```
axis( [ -2.9, 7.2, -2 2 ])
saveas(gcf,'BrentDekker101','jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 BrentDekker101.m을 실행하면, 초기값이 -2.75에서 0.25씩 증가할 때, Brent법에 의한 근을 출력하고 또한 그림 5.8.1이 그려진다. 그림 5.8.1에서 알 수 있듯이, MATLAB함수 fzero.m에 의한 근은 초기값이 1인 경우에는 근 1을 출력하지만, 나머지 경우에는 근 -1을 출력한다. ■

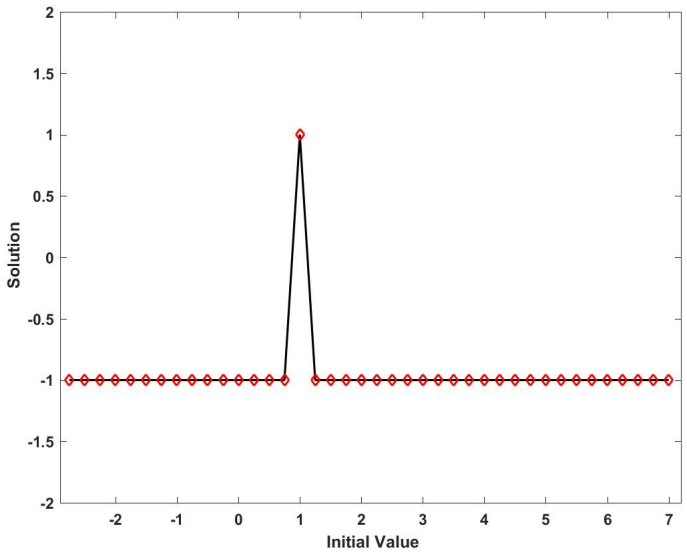


그림 5.8.1. Brent법 (MATLAB)

예제 5.8.2 Python을 사용해서 예제 5.8.1을 다시 다루기 위해서, 다음 Python프로그램 BrentDekker101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

f = lambda x: x**3 - x**2 - x +1;
initial = np.zeros(40);
sol = np.zeros(40);
for ii in range(0,40):
    initial[ii] = 0.25*(ii+1) - 3;
```

```
    sol[ii] = fsolve(f,initial[ii]);
print(initial)
print(sol)

# Plotting
fig = plt.figure()
plt.plot(initial,sol,'k-',initial,sol,'rd',lw=2)
plt.xlabel('Initial Value')
plt.ylabel('Solution')
plt.axis( [ -2.9, 7.2, -2, 2 ])
plt.show()
fig.savefig('BrentDekker101Py.png')

# End of program
```

이 Python 프로그램을 수행한 결과는 예제 5.8.1의 결과와 다르다. 이 Python 프로그램 BrentDekker101.Py를 실행하면, 초기값이 -2.75에서 0.25씩 증가할 때, Brent 법에 의한 근을 출력하고 또한 그림 5.8.2가 그려진다. 그림 5.8.2에서 알 수 있듯이, Python 함수 numpy.fsolve에 의한 근은 초기값이 0보다 큰 경우에는 근 1을 출력하지만, 나머지 경우에는 근 -1을 출력한다. ■

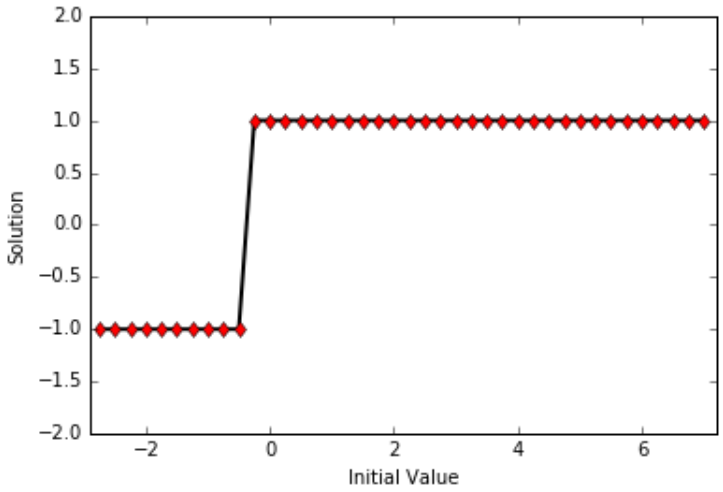


그림 5.8.2. Brent 법 (Python)

제5.9절 비선형방정식과 MATLAB 함수와 Python 함수

이 절에서는 비선형방정식을 푸는데 유용한 MATLAB 함수들을 살펴보자.

5.9.1 MATLAB함수 fzero.m

미지수가 1개인 비선형방정식의 근을 구하기 위해서 MATLAB함수 fzero.m을 사용할 수 있다. 첫째, 방정식 $f(x)=0$의 값 $x=a$ 주변에서 근을 구하기 위해서, 다음 MATLAB 명령문을 사용한다.

```
>> x = fzero(f,a)
```

둘째, 방정식 $f(x)=0$의 구간 $[b,c]$에서 근을 구하기 위해서, 다음 MATLAB명령문을 사용한다.

```
>> x = fzero(f,[b,c])
```

여기서 유의할 점은 MATLAB함수 fzero.m은 식 $f(x+)f(x-)<0$를 만족하는 근만을 구한다는 것이다.

예제 5.9.1 여러 비선형방정식들을 풀기 위해서, 다음 MATLAB프로그램 USEfzero101.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: USEfzero101.m
%  Examples of MATLAB function fzero.m
%  To find a solution x satisfying f(x-)f(x+) < 0
% ------------------------------------------------------------------------
clear all, close all, format long
% (Example 1) Finding the zero of the sine function near 3
x1 = fzero(@sin,3)
% (Example 2) Finding the zero of cosine between 1 and 2
x2 = fzero(@cos,[1 2])
% (Example 3) Finding a zero of an anonymous function
f = @(x)x.^3-2*x-5;
x3 = fzero(f,2)
% (Example 4) Finding a zero of an anonymous function
f4 = @(x) exp(-x) - x;
x4 = fzero(f4,0)
% (Example 5) Finding a zero of an anonymous function
f5 = @(x) x^2 - 2;
x5 = fzero(f5,-1)
% (Example 6) Finding a zero of an anonymous function
f6 = @(x) x^3 - 3*x^2 - x + 3;
x6 = fzero(f6,-2)
% End of Program
% ------------------------------------------------------------------------
```

첫 번째 예제는 방정식 $\sin x=0$의 값 $x=3$ 주변에서 근을 구하는 것으로, 답은 $x_1=3.1416$이다. 두 번째 예제는 방정식 $\cos x=0$의 구간 $[1,2]$에서 근을 구하는 것으로, 답은 $x_2=1.5708$이다. 세 번째 예제는 방정식 $x^3-2x-5=0$의 값 $x=2$ 주변에서 근을 구하는 것으로, 답은 $x_3=2.0946$이다. 네 번째 예제는 방정식 $e^{-x}-x=0$의 값 $x=0$

주변에서 근을 구하는 것으로, 답은 $x_4 = 0.5671$ 이다. 다섯 번째 예제는 방정식 $x^2 - 2 = 0$ 의 값 $x = -1$ 주변에서 근을 구하는 것으로, 답은 $x_5 = -1.4142$ 이다. 여섯 번째 예제는 방정식 $x^3 - 3x^2 - x + 3 = 0$ 의 값 $x = -2$ 주변에서 근을 구하는 것으로, 답은 $x_6 = -1$ 이다. ■

5.9.2 MATLAB함수 roots.m

멱함수방정식 (polynomial equation) 의 근을 구하기 위해서 MATLAB함수 roots.m을 사용할 수 있다. 방정식 $c_1x^n + x_2x^{n-1} + \cdots + c_nx^1 + c_{n+1} = 0$ 의 근을 구하기 위해서, 다음 MATLAB 명령문을 사용한다.

```
>> p = [ c1 c2  ... cp ]
>> x = roots(p)
```

출력변수 x에는 모든 근들이 벡터형태로 출력된다.

예제 5.9.2 멱함수방정식을 푸는 예로서, 다음 MATLAB프로그램 USEroots101.m을 실행해보라.

```
% ------------------------------------------------------------------------
%  Filename: USEroots101.m
%  Examples of MATLAB function roots.m
% To find solutions of polynomial equations
% ------------------------------------------------------------------------
% (Example 1) Finding the zero of x^3 - 6x^2 - 72x -27
p = [1 -6 -72 -27];
r1 = roots(p)
% (Example 2) Finding the zero of x^3 - 3x^2 - x + 3
r2 = roots([1 -3 -1 3 ])
% (Example 3) Finding the zero of x^3 + 1
r3 = roots([1 0 0 1 ])
% End of Program
% ------------------------------------------------------------------------
```

첫 번째 예제는 방정식 $x^3 - 6x^2 - 72x - 27 = 0$ 의 근을 구하는 것으로, 답은 $\{12.1229,\ -5.7345,\ -0.3884\}$ 이다. 두 번째 예제는 방정식 $x^3 - 3x^2 - x + 3 = 0$ 의 근을 구하는 것으로, 답은 $\{-3,\ 1,\ -1\}$ 이다. 세 번째 예제는 방정식 $x^3 + 1 = 0$ 의 근을 구하는 것으로, 답은 $\{-1.0000,\ 0.5000 + 0.8660i,\ 0.5000 - 0.8660i\}$ 이다. 즉, $\left\{-1,\ \frac{1+\sqrt{3}i}{2},\ \frac{1-\sqrt{3}i}{2}\right\}$ 이다. ■

5.9.3 선형연립방정식

MATLAB이나 Python을 사용해서 선형연립방정식을 푸는 것은 매우 간단하다. 우선 다음 예제를 살펴보자.

예제 5.9.3 다음 선형연립방정식을 살펴보자.

$$x_1 + x_2 + x_3 + x_4 = 4 \tag{1}$$

$$2x_1 - x_2 + x_3 = 0 \tag{2}$$

$$3x_1 + 2x_2 + x_3 - x_4 = 6 \tag{3}$$

$$x_1 - 2x_2 - 2x_3 + 2x_4 = -1 \tag{4}$$

이 선형연립방정식을 다음과 같이 쓸 수 있다.

$$A\boldsymbol{x} = \boldsymbol{b} \tag{5}$$

여기서 A, $\boldsymbol{x}$ 그리고 $\boldsymbol{b}$는 각각 다음과 같다.

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & -1 & 1 & 0 \\ 3 & 2 & 1 & -1 \\ 1 & -2 & -2 & 2 \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 4 \\ 0 \\ 6 \\ -1 \end{bmatrix} \tag{6}$$

선형연립방정식 (5)를 풀기 위해서, 다음 MATLAB프로그램 SimultaneousLinearEquation101.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: SimultaneousLinearEquation101.m
%  Solving simultaneous linear equations
% ------------------------------------------------------------------------
A = [ 1 1 1 1 ; 2 -1 1 0 ; 3 2 1 -1 ; 1 -2 -2 2 ]
b = [ 4 0 6 1 ]'
x = A\b
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 SimultaneousLinearEquation101.m을 실행하면, 다음과 같은 해를 얻는다.

$$x_1 = 1.2857, \quad x_2 = 2.0000, \quad x_3 = -0.5714, \quad x_4 = 1.2857 \tag{4}$$

■

예제 5.9.4 Python을 사용해서 예제 5.9.3을 다시 다루기 위해서, 다음 Python프로그램 SimultaneousLinearEquation101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np

A = np.array([[ 1, 1, 1, 1], [2, -1, 1, 0], [ 3, 2, 1, -1], [1, -2, -2, 2]])
b = np.array([ 4, 0, 6, 1 ])
b.shape = (4,1)
x = np.linalg.solve(A,b)
print(x)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 5.9.3의 결과와 같다. ■

5.9.4 MATLAB함수 fsolve.m와 Python함수 fsolve

비선형연립방정식의 근을 구하기 위해서 MATLAB함수 fsolve.m와 Python패키지 scipy.optimize의 Python함수 fsolve를 사용할 수 있다.

MATLAB함수 fsolve.m의 기본적인 사용법은 다음과 같다.

```
>> x = fsolve(fun,x0)
```

여기서 입력변수 fun은 풀고자하는 연립방정식을 나타내는 함수벡터이고, x0는 초기벡터이다. 또한, 출력변수 x는 연립방정식의 해이다. 다른 사용법은 다음과 같다.

```
>> [x,fval] = fsolve(fun,x0)
```

여기서 출력변수 fval는 연립방정식의 해에서 함수값들의 벡터이다.

Python패키지 scipy.optimize의 함수 fsolve의 기본적인 사용법은 다음과 같다.

```
In [101]: from scipy.optimize import fsolve
In [102]: x = fsolve(fun,x0)
In [103]: fval = fun(x)
```

여기서 입력변수 fun은 풀고자하는 연립방정식을 나타내는 함수벡터이고, x0는 초기벡터이고, 출력변수 x는 연립방정식의 해이다. 또한, 출력변수 fval는 연립방정식의 해에서 함수값들의 벡터이다.

예제 5.9.5 다음 비선형연립방정식을 살펴보자.

$$2x_1 - x_2 = \sin x_1 \tag{1}$$

$$-x_1 + 2x_2 = \exp(-x_2) \tag{2}$$

이 선형연립방정식을 풀기 위해서, 다음 MATLAB프로그램 USEfsolve101.m을 실행하라.

```
% ------------------------------------------------------------------
%  Filename: USEfsolve101.m
%  Examples of MATLAB function fsolve.m
%  To find a solution of siultaneous eqautions
%  MATLAB program myNSfun101 should be in the same directory
% ------------------------------------------------------------------
function USEsolve101
clear all, close all, format long
x0 = [ -3; - 2]
x = fsolve(@myNSfun101,x0)
[y fval ]  = fsolve(@myNSfun101,x0)
end
% End of program
%-------------------------------------------------------------------
function F = myNSfun101(x)
F = [ 2*x(1) - x(2) - sin(x(1))  ;
      -x(1) + 2*x(2) - exp( -x(2)) ]
end
%-------------------------------------------------------------------
```

MATLAB명령문 'x = fsolve(@myNSfun1,x0)'이 실행되면, 다음과 같은 해를 얻는다.

$$x_1 = 0.5276, \quad x_2 = 0.5518 \tag{3}$$

MATLAB명령문 '[y fval] = fsolve(@myNSfun1,x0)'이 실행되면, 다음과 같은 해를 얻는다.

$$y_1 = 0.5276, \quad y_2 = 0.5518 \tag{4}$$

또한, 해당하는 함수값들은 각각 다음과 같다.

$$\text{fval}_1 = 2x_1 - x_2 - \sin x_1 = -0.6887 \cdot 10^{-8} \tag{5}$$

$$\text{fval}_2 = -x_1 + 2x_2 - \exp(-x_2) = -0.1177 \cdot 10^{-8} \tag{6}$$

■

예제 5.9.6 Python을 사용해서 예제 5.9.5을 다시 다루기 위해서, 다음 Python프로그램 USEfsolve101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy.optimize import fsolve

def SimEqs(vr):
    x1, x2 = vr
    return ( 2*x1-x2-np.sin(x1), -x1+2*x2-np.exp(-x2) )

x0 = (-3, -2)
x1, x2 = fsolve(SimEqs, x0)
print(x1,x2)
print ( SimEqs((x1,x2)) )

# End of program
```

이 Python프로그램을 수행한 결과는 예제 5.9.5의 결과와 같다. ■

5.9.5 비선형연립방정식

Newton-Raphson법을 다변량으로 확장해서 비선형연립방정식을 풀 수 있다. 그러한 경우에는 Jacobian행렬을 구해야 한다. 이는 때때로 번잡한 일이기도 하다. 따라서, 비선형연립방정식을 최적화문제(optimization problem)로 바꾸어서 풀기도 한다. 다음 연립방정식을 푸는 문제를 살펴보자.

$$f_k(\boldsymbol{x}) = 0, \quad (k = 1, 2, \cdots, K) \tag{5.9.1}$$

여기서 f_k는 편미분 $\frac{\partial f}{\partial x_k}$를 의미하는 것이 아니라 제$k$번째 함수를 의미함에 유의하라. 다음과 같이 목적함수의 최소값이 되는 벡터를 $\boldsymbol{x}^*$라고 하면, 다음 식이 성립한다.

$$\min_{\boldsymbol{x}} \sum_{k=1}^{K} [f_k(\boldsymbol{x})]^2 = \sum_{k=1}^{K} [f_k(\boldsymbol{x}^*)]^2 = 0 \tag{5.9.2}$$

MATLAB함수 fsolve.m은 이러한 방법으로 연립방정식의 해를 구한다.

예제 5.9.7 다음 비선형함수를 살펴보자.

$$f(x) = [x^3 - x] \exp(-x^2) \tag{1}$$

방정식 $f(x) = 0$의 근이 $-1, 0, 1$임은 자명하다. 기본적으로 MATLAB함수 fsolve.m은 이 방정식을 다음과 같은 최적화문제로 바꾸어 푼다.

$$\min_x f^2(x) = \min_x [x^3 - x]^2 \exp(-2x^2) \tag{2}$$

이 비선형방정식을 풀기 위해서, 다음 MATLAB프로그램 USEfsolve102.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: USEfsolve102.m
%  Solving equations using optimization techniques
%  Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
% function f(x)
f = @(x) (x.^3 - x).*exp(-x.^2);
% Plotting f(x)
xx = -6:0.05:6;
subplot(2,1,1)
plot(xx,f(xx),'k', 'LineWidth', 2);
set(gca,'fontsize',11,'fontweigh','bold')
title('\bf Plof of f(x)','fontsize',13)
xlabel('\bf x','fontsize',12)
ylabel('\bf f(x)','fontsize',12,'rotation',0)
grid on
subplot(2,1,2)
plot(xx,f(xx).^2,'k', 'LineWidth', 2);
set(gca,'fontsize',11,'fontweigh','bold')
title('\bf Plot of f^{2} (x)','fontsize',13)
xlabel('\bf x','fontsize',12)
ylabel('\bf f^{2}(x)','fontsize',12,'rotation',0)
grid on
saveas(gcf,'USEfsolve102','jpg')
% solve f(x) = 0 near x0 = 1.2
[x1,f1] = fsolve(f,1.2)
% solve f(x) = 0 near x0 = -1.7
[x2,f2] = fsolve(f,-1.7)
% End of program
% ------------------------------------------------------------------------
```

MATLAB프로그램 USEfsolve102.m을 실행하면, 그림 5.9.1이 그려진다. 그림 5.9.1의 상단 그래프에서 방정식 $f(x) = 0$의 근이 $-1, 0, 1$임은 알 수 있다. 또한, 하단 그래프에서 $|x| > 3$인 경우에 $f^2(x) \approx 0$ 임을 알 수 있다.

MATLAB명령문 'fsolve(f,1.2)'이 실행되면, 초기값이 $x_0 = 1.2$인 경우 방정식 $f(x) = 0$의 근이 $x_1 = 1.0000$임을 알 수 있다. 이 경우에 $f(x_1) = -3.6 \times 10^{-7}$이다. MATLAB명령문 'fsolve(f,-1.7)'이 실행되면, 초기값이 $x_0 = -1.7$인 경우 방정식 $f(x) = 0$의 근이 $x_2 = -3.5056$임을 알 수 있다. 이 경우에 $f(x_2) = -1.8 \times 10^{-4}$이다. 물론 $f(x_2) \neq 0$이므로, $x_2 = -3.5056$은 방정식 $f(x) = 0$의 근이 될 수 없다. 그러나, $|f(x_2)| = 1.8 \times 10^{-4}$ 가

오차허용범위(TolFun) 보다 작으므로, MATLAB함수 fsolve.m은 $x_2 = -3.5056$를 방정식 $f(x) = 0$의 근으로 간주한다. ■

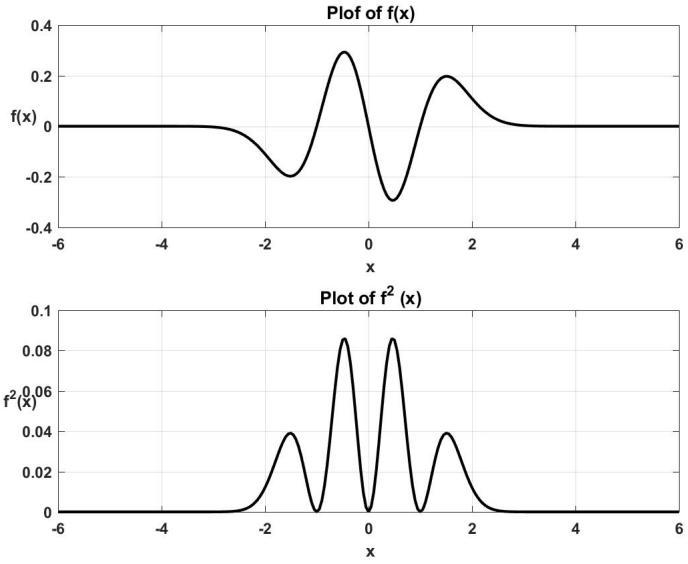


그림 5.9.1. MATLAB함수 fsolve와 최적화

예제 5.9.8 Python을 사용해서 예제 5.9.7을 다시 다루기 위해서, 다음 Python프로그램 USEfsolve102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

f = lambda x: (x**3-x)*np.exp(-x**2)

# Plotting f(x)
xx = np.linspace(-6,6.0,241)
fig = plt.figure()
ax1 = fig.add_subplot(211)
ax1.plot(xx,f(xx),'k',lw=2)
ax1.set_title('Plof of f(x)')
ax1.set_xlabel('x'); ax1.set_ylabel('f(x)')
ax1.grid()
ax2 = fig.add_subplot(212)
ax2.plot(xx,f(xx)**2,'k',lw=2)
ax2.set_title('Plof of f(x)^2')
ax2.set_xlabel('x'); ax2.set_ylabel('f(x)^2')
ax2.grid()
plt.tight_layout()
fig = plt.gcf()
```

```
27
28 # solve f(x) = 0 near x0 = 1.2
29 sol1 = fsolve(f,1.2)
30 print(sol1)
31 print(f(sol1))
32 # solve f(x) = 0 near x0 = -1.7
33 sol2 = fsolve(f,-1.7)
34 print(sol2)
35 print(f(sol2))
36
37 # End of program
```

이 Python프로그램을 수행한 결과 생성된 그래프들은 예제 5.9.7에 의한 그래프들은 같다. 또한, 초기값이 -1.2인 경우 방정식 $f(x)=0$의 근은 1로 동일하다. 그러나, 초기값이 -1.7인 경우에 예제 5.9.7에서 알 수 있듯이 MATLAB의 fsolve에 의한 근은 -3.5056이고 함수값은 -1.8208×10^{-4}이고, Python의 fsolve에 의한 근은 -19.9960이고 함수값은 -1.7274×10^{-122}이다. ■

연립방정식 (5.9.1)의 좌변에 점 $\boldsymbol{x}^*=[x_1^*,x_2^*,\cdots,x_K^*]^t$에서 1차 Taylor근사를 적용하면, 다음 근사방정식들을 얻는다.

$$f_1(x^*)+[x_1-x_1^*]\frac{\partial f_1}{\partial x_1}+\cdots+[x_k-x_K^*]\frac{\partial f_1}{\partial x_K}\approx 0 \tag{5.9.3}$$

$$f_2(x^*)+[x_1-x_1^*]\frac{\partial f_2}{\partial x_1}+\cdots+[x_k-x_K^*]\frac{\partial f_2}{\partial x_K}\approx 0 \tag{5.9.4}$$

$$\vdots$$

$$f_K(x^*)+[x_1-x_1^*]\frac{\partial f_K}{\partial x_1}+\cdots+[x_k-x_K^*]\frac{\partial f_K}{\partial x_K}\approx 0 \tag{5.9.5}$$

여기서 각 편미분은 점 $\boldsymbol{x}^*$에서 계산된 것이다. 따라서, 만약 점 $\boldsymbol{x}^*$가 주어졌다면, 식 (5.9.3)~식 (5.9.5)는 선형연립방정식을 구성한다. 이 선형연립방정식을 행렬로 표기하면, 다음과 같다.

$$J^*\Delta\boldsymbol{x}^*=-\boldsymbol{f}^* \tag{5.9.6}$$

여기서 행렬과 벡터들은 다음과 같다.

$$J^*\doteq\begin{bmatrix}\frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_K}\\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_K}\\ \vdots & \vdots & & \vdots\\ \frac{\partial f_K}{\partial x_1} & \frac{\partial f_K}{\partial x_2} & \cdots & \frac{\partial f_K}{\partial x_K}\end{bmatrix},\quad \Delta\boldsymbol{x}^*\doteq\begin{bmatrix}x_1-x_1^*\\ x_2-x_2^*\\ \vdots\\ x_K-x_K^*\end{bmatrix},\quad \boldsymbol{f}\doteq\begin{bmatrix}f_1(x^*)\\ f_2(x^*)\\ \vdots\\ f_K(x^*)\end{bmatrix} \tag{5.9.7}$$

이 J^*를 Jacobian행렬(Jacobian matrix)이라 부른다. 연립방정식 (5.9.6)을 풀기 위해서, 다음과 같은 방정식을 생각할 수 있다.

$$J^{(k)}\Delta\boldsymbol{x}^{(k)} = -\boldsymbol{f}^{(k)} \tag{5.9.8}$$

여기서 $J^{(k)}$, $\Delta\boldsymbol{x}^{(k)}$ 그리고 $\boldsymbol{f}^{(k)}$는 점 $\boldsymbol{x}^{(k)}$에서 계산된 것이다. 방정식 (5.9.8)의 해를 $\boldsymbol{x}^{(k+1)}$라고 하면, 다음 점화식을 얻는다.

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - [J^{(k)}]^{-1}\boldsymbol{f}^{(k)} \tag{5.9.9}$$

점화식 (5.9.9)는 비선형연립방정식 (5.9.1)을 풀기 위한 Newton-Raphson알고리즘이다. 그러나, 이 Newton-Raphson알고리즘은 각 단계에서 Jacobian행렬과 그 역행렬을 계산해야 하므로, 많은 계산을 필요로 하고 또한 오차를 많이 발생시킨다. 따라서, 식 (5.9.2)에서 기술한 최적화문제를 푸는 것과 같은 준Newton-Raphson법(quasi-Newton-Raphson method)들이 자주 사용된다.

다음은 최적화를 사용해서 비선형연립방정식을 푸는 예제이다.

예제 5.9.9 다음 함수들을 정의하자.

$$f_1(x_1, x_2) = \exp(x_1 + x_2) - \exp(3) \tag{1}$$

$$f_2(x_1, x_2) = [x_1 - x_2]^3 - 1 \tag{2}$$

다음 비선형연립방정식을 살펴보자.

$$f_1(x_1, x_2) = 0, \quad f_2(x_1, x_2) = 0 \tag{3}$$

이 연립방정식을 다음과 같이 쓸 수 있다.

$$x_1 + x_2 = 3 \tag{4}$$

$$x_1 - x_2 = 1 \tag{5}$$

즉, 이 연립방정식의 해가 $(x_1, x_2) = (2, 1)$임을 쉽게 알 수 있다.

이 예제에서는 MATLAB함수 fsolve.m을 사용해서, 연립방정식 (3)을 풀기로 하자. 이 연립방정식의 Jacobian행렬은 다음과 같다.

$$J = \begin{bmatrix} \exp(x_1 + x_2) & \exp(x_1 + x_2) \\ 3[x_1 - x_2]^2 & -3[x_1 - x_2]^2 \end{bmatrix} \tag{6}$$

식 (6)의 Jacobian행렬을 이용하는 선형화된 연립방정식을 풀기 위해서, 다음 MATLAB 프로그램 USEfsolve103.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: USEfsolve103.m
%  Solving simultaneous nonlinear equations
%  using optimization techniques
%  with MATLAB function fsolve
% ------------------------------------------------------------------------
function USEfsolve103
% x0 is a starting guess of K and r.
x0 = [10 0.5];                  % Initial solution
[ x fx ] = fsolve(@myNSfun103,x0)     % Call solver
end
% ------------------------------------------------------------------------
function F = myNSfun103(x)
% Write the equation in the form F(x) = 0
F = [ exp(x(1) + x(2)) - exp(3);
      (x(1) - x(2))^3 -  1 ];
end
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 USEfsolve103.m을 실행하면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

```
Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.


<stopping criteria details>

x =  2.0000    1.0000

fx = 1.0e-11 *

     0.0043    0.1243
```

■

예제 5.9.10 Python을 사용해서 예제 5.9.9을 다시 다루기 위해서, 다음 Python프로그램 USEfsolve103.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy.optimize import fsolve

def myNSfun103(p):
    x1, x2 = p
    f1 = np.exp(x1+x2) - np.exp(3.0)
    f2 = (x1-x2)**3 - 1
    return (f1,f2)

x1, x2 = fsolve(myNSfun103, (10, 0.5))
print(x1, x2)
print(myNSfun103((x1,x2)))

# End of program
```

이 Python프로그램을 수행한 결과는 예제 5.9.9의 결과와 같다. ■

예제 5.9.11 다음 함수들을 정의하자.

$$f_1(x_1, x_2) = \exp\left(-\exp\left(-x_1 - x_2\right)\right) - x_2[1 + x_1^2] \tag{1}$$

$$f_2(x_1, x_2) = x_1 \cos(x_2) + x_2 \sin(x_1) - \frac{1}{2} \tag{2}$$

다음 비선형연립방정식을 살펴보자.

$$f_1(x_1, x_2) = 0, \quad f_2(x_1, x_2) = 0 \tag{3}$$

이 예제에서는 MATLAB함수 fsolve.m을 사용해서, 연립방정식 (3)을 풀기 위해서, 다음 MATLAB프로그램 USEfsolve104.m을 실행하라.

```
% --------------------------------------------------------------------
%  Filename: USEfsolve104.m
%  Example of MATLAB function fsolve.m w/ Iterations
%  Programmed by MathWorks
% --------------------------------------------------------------------
function USEfsolve104
clear all, clear all, format long
fun104 = @myNSfun104;
x0 = [0,0];                          % Initial vector
```

```
options=optimset('Display','iter'); % Option to display output
x = fsolve(fun104,x0,options)       % Call solver
end
% End of program
% ------------------------------------------------------------------------
function f = myNSfun104(x)
f(1) = exp(-exp(-x(1)+x(2))) - x(2)*(1+x(1)^2);
f(2) = x(1)*cos(x(2)) + x(2)*sin(x(1)) - 0.5;
end
%------------------------------------------------------------------------
```

이 MATLAB프로그램 USEfsolve104.m에서는 MATLAB함수 optimset.m의 옵션 Display에 iter을 지정했다. 따라서, 이 MATLAB프로그램을 실행하면, 각 반복단계별 결과물이 출력된다. 즉, 이 MATLAB프로그램 USEfsolve104.m을 실행하면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

```
                                        Norm of      First-order   Trust-region
 Iteration  Func-count     f(x)          step         optimality    radius
     0          3        0.385335                        0.503             1
     1          6       0.0336737       0.642449         0.206             1
     2          9     0.000110235       0.122659        0.0162          1.61
     3         12     8.13142e-11     0.00681475      1.13e-05          1.61
     4         15     4.11698e-22     7.0962e-06      3.06e-11          1.61

Equation solved.
fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>
x =  0.393113400037231   0.336627015155633
```

■

예제 5.9.12 Python을 사용해서 예제 5.9.11을 다시 다루기 위해서, 다음 Python프로그램 USEfsolve104.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
```

```
from scipy.optimize import fsolve

def equations(p):
    x1, x2 = p
    f1 = np.exp(-np.exp(-x1 + x2 )) - x2*(1+x1**2)
    f2 = x1*np.cos(x2) + x2*np.sin(x1) - 0.5
    return (f1,f2)

x0 = (0, 0)
x1, x2 = fsolve(equations, x0)
print(x1, x2)
print(equations((x1,x2)))

# End of program
```

이 Python프로그램을 수행한 결과는 예제 5.9.11의 결과와 같다. ■

예제 5.9.13 다음 함수들을 정의하자.

$$f_1(\boldsymbol{x}) = 3x_1 - 2x_1^2 - 2x_2 + 1 \tag{1}$$

$$f_i(\boldsymbol{x}) = 3x_i - 2x_i^2 - x_{i-1} - 2x_{i+1} + 1, \quad (i = 2, 3, \cdots, 999) \tag{2}$$

$$f_{1000}(\boldsymbol{x}) = 3x_{1000} - 2x_{1000}^2 - x_{999} + 1 \tag{3}$$

다음 비선형연립방정식을 살펴보자.

$$f_i(\boldsymbol{x}) = 0, \quad (i = 1, 2, \cdots, 1000) \tag{4}$$

이 비선형연립방정식을 풀기 위한 Jacobian행렬 J의 (i,j)-원소 $J(i,j)$는 다음과 같다.

$$J(i,j) = \begin{cases} 3 - 4x_i, & (i = j = 1, 2, \cdots, 1000) \\ -1, & (i = j + 1 = 2, 3, \cdots, 1000) \\ -2, & (i = j - 1 = 12, 3, \cdots, 999) \\ 0, & (\text{otherwise}) \end{cases} \tag{5}$$

MATLAB함수 fsolve.m을 사용해서, 연립방정식 (1)~(3)을 풀기 위해서, 다음 MATLAB 프로그램 USEfsolve105.m를 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: USEfsolve105.m
```

```
%  MATLAB function fsolve.m w/ Jacobian matrix
%  Programmed by MathWorks
% ------------------------------------------------------------------------
function USEfsolve105
tic
clear all, clear all, format long
xstart = -ones(1000,1);
options = optimoptions(@fsolve,'Jacobian','on', ...
                               'DerivativeCheck','on');
[x,fval,exitflag,output] = fsolve(@myNSfun105,xstart,options);
exitflag, output
toc
end
% End of program
% ------------------------------------------------------------------------
function [f,fJacob] = myNSfun105(x)
% Evaluate the vector function
n = length(x);
f = zeros(n,1);
  f(1) = (3-2*x(1)).*x(1)-2*x(2) + 1;
  f(n) = (3-2*x(n)).*x(n)-x(n-1) + 1;
i = 2:(n-1);
  f(i) = (3-2*x(i)).*x(i)-x(i-1)-2*x(i+1) + 1;
% Evaluate the Jacobian if nargout > 1
if nargout > 1
   d = -4*x + 3*ones(n,1);
       D = sparse(1:n,1:n,d,n,n);
   c = -2*ones(n-1,1);
       C = sparse(1:n-1,2:n,c,n,n);
   e = -ones(n-1,1);
       E = sparse(2:n,1:n-1,e,n,n);
   fJacob = C + D + E
end
end
%------------------------------------------------------------------------
```

이 MATLAB프로그램 USEfsolve105.m에서는 다음과 같이 MATLAB함수 optimoptions.m을 사용하였다.

```
options = optimoptions(@fsolve,'Display','iter', ...
                'Jacobian','on','DerivativeCheck','on');
```

여기서 옵션 Jacobian이 'on'되었으므로, 사용자가 Jacobian행렬을 제공해야 한다. 이 경우에는 MATLAB함수 fsolve.m의 첫 번째 입력변수에는 2개 요소들이 포함되어야 하는데, 그 중에서 첫 번째 요소는 풀고자하는 목적함수 $\boldsymbol{f}(\boldsymbol{x})$ 그리고 두 번째 요소는 이 목적함수의 Jacobian행렬 J이다. 이 예제에서는 입력변수 '@myNSfun105'에 의해서 목적함수인 f와 Jacobian행렬인 fJacob을 제공한다. 만약 옵션 Jacobian이 'off'되면, 유한차분법(finite difference method)에 의해서 Jacobian행렬이 계산된다. 또한, 옵션 DerivativeCheck가 'on'되었으므로, 사용자가 제공한 Jacobian행렬을 유한차분법에 의한 Jacobian행렬과 비교한다. 출력변수 exitflag에는 알고리즘에서 벗어나는 조건이 포함되고, 출력변수 output에는 최적화

에 대한 정보가 포함된다. 이 MATLAB프로그램 USEfsolve105.m을 실행하면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

```
Equation solved.
fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>
exitflag =     1
output =
       iterations: 6
        funcCount: 7
        algorithm: 'trust-region-dogleg'
    firstorderopt: 5.854314845843616e-13
          message: 'Equation solved.…'

경과 시간은 1.008295초입니다.
```

이 결과물에서 알 수 있듯이, 사용자가 제시한 Jacobian행렬과 유한차분법에 의한 Jacobian 행렬에 의한 편미분들의 차이가 최대 6.50×10^{-8} 이다. 또한, MATLAB함수 fsolve.m은 점화식 (5.9.8)을 반복적용하다 멈춤기준(stopping criterion)이 만족되면 알고리즘의 실행을 멈춘다. 이 문제에서는 제6번째 반복에서 멈춤기준이 만족된다. 출력변수 exitflag이 1이므로, 이를 확인할 수 있다. ■

MATLAB함수 fsolve.m을 적용할 때, Jacobian행렬을 잘못 명시하는 예를 살펴보자.

예제 5.9.14 예제 5.9.13의 연립방정식을 다시 살펴보기 위해서, 다음 MATLAB프로그램 USEfsolve106.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: USEfsolve106.m
%  MATLAB function fsolve.m w/ wrong Jacobian matrix
%  Programmed by MathWorks
% ------------------------------------------------------------------------
function USEfsolve106
tic
clear all, clear all, format long
xstart = -ones(1000,1);
options = optimoptions(@fsolve,'Jacobian','on', ...
                               'DerivativeCheck','on');
```

```
[x,fval,exitflag,output] = fsolve(@myNSfun106,xstart,options);
exitflag, output
toc
end
% End of program
% ------------------------------------------------------------------------
function [f,fJacob] = myNSfun106(x)
% Evaluate the vector function
n = length(x);
f = zeros(n,1);
  f(1) = (3-2*x(1)).*x(1)-2*x(2) + 1;
  f(n) = (3-2*x(n)).*x(n)-x(n-1) + 1;
i = 2:(n-1);
  f(i) = (3-2*x(i)).*x(i)-x(i-1)-2*x(i+1) + 1;
% Evaluate the Jacobian if nargout > 1
if nargout > 1
   d = -2*x + 3*ones(n,1);
       D = sparse(1:n,1:n,d,n,n);
   c = -2*ones(n-1,1);
       C = sparse(1:n-1,2:n,c,n,n);
   e = -ones(n-1,1);
       E = sparse(2:n,1:n-1,e,n,n);
   fJacob = C + D + E;
end
end
%------------------------------------------------------------------------
```

이 MATLAB프로그램 USEfsolve106.m에서는 MATLAB함수 fsolve.m의 옵션 Jacobian이 'on'되었으나, 예제 5.9.13과는 달리 다음과 같이 잘못된 Jacobian행렬을 지정한다고 하자.

$$J(i,j) = \begin{cases} 3-2x_i, & (i=j=1,2,\cdots,1000) \\ -1, & (i=j+1=2,3,\cdots,1000) \\ -2, & (i=j-1=12,3,\cdots,999) \\ 0, & (\text{otherwise}) \end{cases} \tag{4}$$

이 MATLAB프로그램 USEfsolve106.m을 실행하면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

```
------------------------------------------------------------
   DerivativeCheck Information

Objective function derivatives:
Maximum relative difference between user-supplied
and finite-difference derivatives = 0.400239.
  User-supplied derivative element (345,345):     5.002
```

```
  Finite-difference derivative element (345,345): 7.00399

------------------------------------------------------------

다음 사용 중 오류가 발생함: validateFirstDerivatives (line 97)
DerivativeCheck failed:
User-supplied and forward finite-difference derivatives do not match within 1e-06 relative tolerance.

오류 발생: fsolve (line 339)
    validateFirstDerivatives(funfcn,confcn,x, ...

오류 발생: USEfsolve106 (line 12)
[x,fval,exitflag,output] = fsolve(@myNSfun106,xstart,options);
```

사용자가 제시한 Jacobian행렬과 유한차분법에 의한 Jacobian행렬에 의한 편미분들의 차이가 최대 0.4002이고, MATLAB함수 fsolve.m는 수렴하는 해를 제공하지 못한다. ■

예제 5.9.14에서 발생하는 문제점을 해결하는 방법은 간단하다. 컴퓨터가 디폴트로 제공하는 편미분값들을 사용하면 된다. 즉, 유한차분법에 의해 계산되는 Jacobian행렬을 사용한다. 다음 예제를 살펴보자.

예제 5.9.15 예제 5.9.13의 연립방정식을 다시 살펴보기 위해서, 다음 MATLAB프로그램 USEfsolve107.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: USEfsolve107.m
%  MATLAB function fsolve.m w/ Jacobian matrix
%  Programmed by MathWorks
% ------------------------------------------------------------------------
function USEfsolve107
tic
clear all, clear all, format long
xstart = -ones(1000,1);
[x,fval,exitflag,output,Jacobian] = fsolve(@myNSfun107,xstart);
exitflag, output
toc
end
% End of program
% ------------------------------------------------------------------------
function [f,fJacob] = myNSfun107(x)
% Evaluate the vector function
n = length(x);
f = zeros(n,1);
  f(1) = (3-2*x(1)).*x(1)-2*x(2) + 1;
  f(n) = (3-2*x(n)).*x(n)-x(n-1) + 1;
i = 2:(n-1);
```

```
  f(i) = (3-2*x(i)).*x(i)-x(i-1)-2*x(i+1) + 1;
end
%--------------------------------------------------------------------------
```

이 MATLAB프로그램 USEfsolve107.m에서는 디폴트인 유한차분법에 의해서 Jacobian 행렬이 계산된다. MATLAB함수 fsolve.m의 출력변수 Jacobian에는 Jacobian행렬이 출력된다. 이 MATLAB프로그램을 실행하면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

```
Equation solved.
fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>
exitflag =  1
output =
       iterations: 6
        funcCount: 7007
        algorithm: 'trust-region-dogleg'
    firstorderopt: 6.351979523018102e-13
          message: 'Equation solved.…'

경과 시간은 1.183086초입니다.
```

이 결과물은 예제 5.9.13의 결과물과 동일하다. 예제 5.9.13에서는 올바른 Jacobian행렬을 제공했음에도 불구하고, 이 예제에서와 마찬가지로 제6번째 반복에서 알고리즘의 실행을 멈추었다. 이는 좀 더 많은 올바른 정보를 제공하면 좀 더 빨리 해를 구할 수 있다는 통념을 깨트리는 것이다. 이러한 결과가 나타나는 이유는 MATLAB함수 fsolve.m이 Newton-Raphson 법을 사용해서 비선형방정식을 푸는 것이 아니기 때문이다. MATLAB함수 fsolve.m은 여러 최적화기법들을 바탕으로 하며 또한 많은 옵션들을 포함하고 있고, 이들은 알고리즘의 반복회수나 속도에 큰 영향을 끼친다. ■

제5.10절 예제들

이 절에서는 지금까지 배운 내용을 복습하기 위해서 좀 더 심도있는 예제들을 살펴보자.

5.10.1 이분할법과 시컨트법

다음 함수를 살펴보자.

$$f(x) = \sqrt{x} - 1.1 \tag{5.10.1}$$

방정식 $f(x) = 0$의 근이 1.21임은 명백하다. 이 소절에서는 이 방정식을 이분할법과 시컨트법으로 풀고, 오차분석을 하자. 먼저, 이분할법을 적용하기로 하자.

예제 5.10.1 이분할법을 사용해서 방정식 $\sqrt{x} - 1.1 = 0$의 근을 구하고 오차분석을 하기 위해서, 다음 MATLAB프로그램 BisectionMethodDY901.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename: BisectionMethodDY901.m
% Bisection Method
% Programmed by CDY
% ------------------------------------------------------------------
clear all, close all, format short g
% Initial stage
ftn = @(x) sqrt(x)-1.1
TrueSol = 1.1^2
MaxIter = 100
MaxWidthUnc = 1.0e-8
xLeft = 0; fLeft = ftn(xLeft)
xRight = 2; fRight = ftn(xRight)
% Iteration
for ii = 1:1:MaxIter;
    x(ii) = (xRight+xLeft)/2;
    ff(ii) = ftn(x(ii));
    AbsErr(ii) = abs(x(ii)-TrueSol)
    WidthUnc(ii) = abs(xLeft-xRight)
    if  WidthUnc(ii) <= MaxWidthUnc,
        xstar = x(ii);
        break
    end
    if ff(ii)*fRight > 0,
       xRight = x(ii), fRight = ff(ii)
    else
        xLeft = x(ii), fLeft = ff(ii)
    end
    xstar = x(ii)
end
ii, xstar
ConverErr = (x(ii)-x(ii-1))/2
% Plotting
iidum = (1:1:ii);
OutP = [ iidum; x(iidum); AbsErr(iidum); WidthUnc(iidum) ]'
% figure 1
plot(iidum,AbsErr(iidum),'r-o',iidum,WidthUnc(iidum), ...
     'k--','LineWidth',2);
legend('\bf Absolute Error','\bf Width of Uncertainty Interval',...
        'location','NE')
set(gca,'fontsize',11,'fontweigh','bold')
axis([ -0.5 ii+0.5 -0.01 1.02*WidthUnc(1)])
% title('\bf Bisection Method')
xlabel('\bf Iteration Number \it n','fontsize',12)
```

```
ylabel('\bf Absolute Error','fontsize',12)
saveas(gcf,'BisectionMethod901DY.jpg')
save('BisectionMethod901DY.txt','x','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 BisectionMethodDY901.m에서 초기구간은 $[0,2]$이다. 또한, 불확실성 구간(uncertainty interval), 즉 근을 구하는 소구간의 길이가 10^{-8}보다 작거나 또는 반복회수가 100회가 넘으면, 이분할법 알고리즘의 실행을 멈춘다.

이 MATLAB프로그램 BisectionMethod901DY.m을 실행한 결과의 일부분은 다음과 같다.

n	x(n)	Abs Err	Width
25	1.21	2.1458e-08	1.1921e-07
26	1.21	8.3447e-09	5.9605e-08
27	1.21	6.5565e-09	2.9802e-08
28	1.21	8.9407e-10	1.4901e-08
29	1.21	2.8312e-09	7.4506e-09

이 결과물에서 알 수 있듯이, 이 이분할법 알고리즘은 제29번째 단계에서 반복을 멈춘다. 첫 번째 단계에서 소구간 $[0,2]$의 길이가 2이므로 제n번째 단계에서 소구간의 길이는 2^{2-n}이다. 다음 명제가 성립한다.

$$2^{2-n} \le 10^{-8} \quad \Leftrightarrow \quad k \ge 2 + 8\frac{\ln 10}{\ln 2} = 28.5754 \tag{1}$$

또한, 제29번째 단계에서 절대오차(absolute error)는 다음과 같다.

$$|\epsilon_{29}| = |x_{29} - 1.21| = 2.81 \times 10^{-9} \tag{2}$$

또한 그림 5.10.1이 그려진다. 그림 5.10.1에서 알 수 있듯이, 불확실성 구간의 길이와 절대오차는 지수적으로 감소한다. ■

시컨트법을 적용해서, 방정식 $f(x) = 0$의 근을 구하고 오차분석을 하자.

예제 5.10.2 이분할법을 사용해서 방정식 $\sqrt{x} - 1.1 = 0$의 근을 구하고 오차분석을 하기 위해서, 다음 MATLAB프로그램 FalsePositionMethod901DY.m을 실행하라.

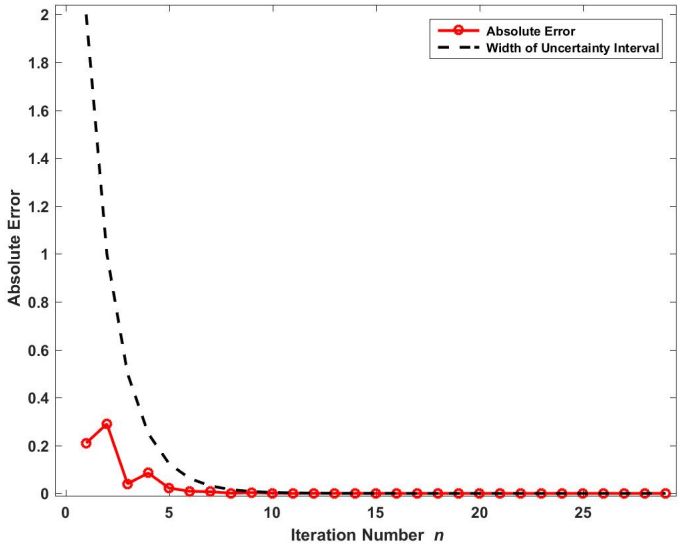


그림 5.10.1. 이분할법

```
% ------------------------------------------------------------------------
% Filename: FalsePositionMethod901DY.m
% False Position Method
% Programmed by CDY
% ------------------------------------------------------------------------
clear all, close all, format short g
% Initial stage
ftn = @(x) sqrt(x)-1.1
TrueSol = 1.1^2
MaxIter = 100
MaxWidthUnc = 1.0e-8
xLeft = 0; fLeft = ftn(xLeft)
xRight = 2; fRight = ftn(xRight)
% Iteration
for ii = 1:1:MaxIter;
    x(ii) = xRight-fRight*(xRight-xLeft)/(fRight-fLeft);
    ff(ii) = ftn(x(ii));
    AbsErr(ii) = abs(x(ii)-TrueSol)
    WidthUnc(ii) = abs(xLeft-xRight)
    % if ff(ii) == 0,
    %    xstar = x(ii);
    %    break
    % end
    if  WidthUnc(ii) <= MaxWidthUnc,
        xstar = x(ii);
        break
    end
    if ff(ii)*fRight > 0,
       xRight = x(ii), fRight = ff(ii)
    else
        xLeft = x(ii), fLeft = ff(ii)
    end
    xstar = x(ii)
end
ii, xstar
% Plotting
```

```
iidum = (1:1:ii);
OutP = [ iidum; x(iidum); AbsErr(iidum); WidthUnc(iidum) ]'
plot(iidum,AbsErr(iidum),'r-o',iidum,WidthUnc(iidum),'k--', ...
     'LineWidth',2);
legend('\bf Absolute Error','\bf Width of Uncertainty Interval', ...
       'location','NE')
set(gca,'fontsize',11,'fontweigh','bold')
axis([ -0.5 ii+0.5 -0.01 1.02*WidthUnc(1)])
xlabel('\bf Iteration Number \it n','fontsize',12)
ylabel('\bf  Absolute Error','fontsize',12)
% title('\bf False Posiiton Method')
saveas(gcf,'FalsePositionMethod901DY.jpg')
save('FalsePositionMethod901DY.txt','x','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 FalsePositionMethod901DY.m에서 초기구간은 $[0, 2]$이다. 또한, 불확실성 구간(uncertainty interval), 즉 근을 구하는 소구간의 길이가 10^{-8}보다 작거나 또는 반복회수가 100회가 넘으면, 시컨트법 알고리즘의 실행을 멈춘다.

이 MATLAB프로그램 FalsePositionMethod901DY.m을 실행한 결과의 일부분은 다음과 같다.

n	x(n)	Abs Err	Width
46	1.21	8.6597e-15	1.21
47	1.21	4.2188e-15	1.21
48	1.21	1.9984e-15	1.21
49	1.21	1.1102e-15	1.21
50	1.21	6.6613e-16	1.21
51	1.21	4.4409e-16	1.21
52	1.21	2.2204e-16	1.21
53	1.21	2.2204e-16	2.2204e-16

이 결과물에서 알 수 있듯이, 이 시컨트법 알고리즘은 제53번째 단계에서 반복을 멈춘다. 초기값이 -2.75에서 0.25씩 증가할 때, Brent법에 의한 근을 출력하고 또한 그림 5.10.2를 출력한다. 그림 5.10.2에서 알 수 있듯이, 절대오차는 지수적으로 감소한다. 그러나, 이분할법과는 달리 시컨트법에서는 확실성 구간의 길이는 1.21에 수렴하다 제53번째 단계에서 0으로 감소한다. 이것이 시컨트법의 약점이다. 시컨트법에서는 반복이 계속될 때 블래키팅점들(blacketing point) 중 하나가 고정되는(fixed) 경향이 있으므로, 시컨트법은 수렴성에 문제가 있을 수도 있다. 특히, 함수 $f(x)$의 곡률이 큰 경우에는 이러한 현상이 자주 나타난다. ■

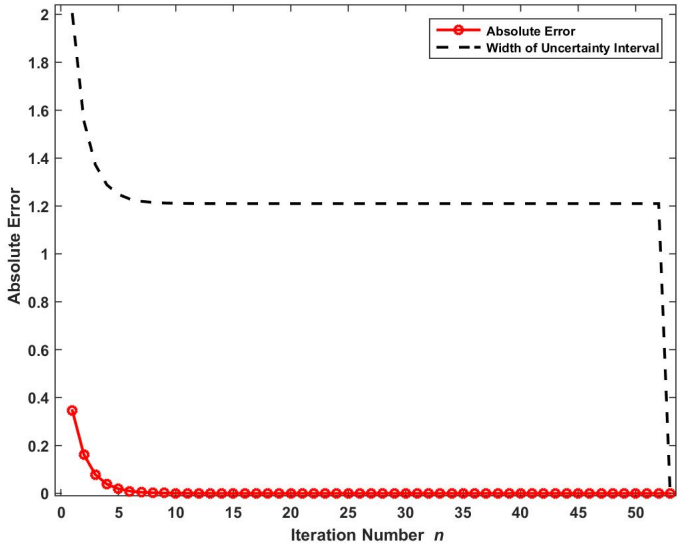


그림 5.10.2. 시컨트법

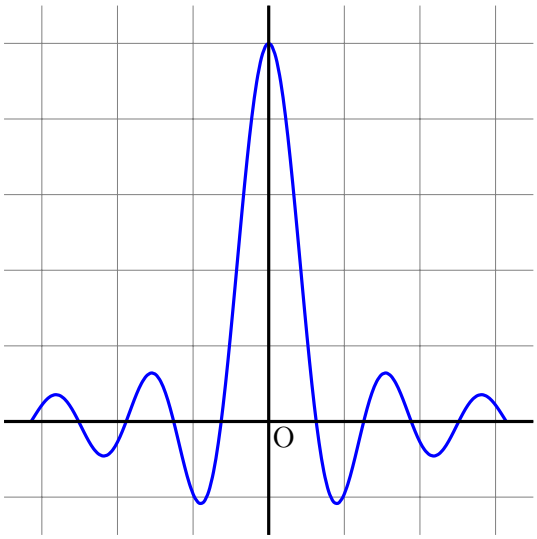
5.10.2 이분할법과 Newton-Raphson법

다음 함수를 살펴보자.

$$f(x) = \mathrm{sinc}(x) \doteq \begin{cases} \frac{\sin x}{x}, & (x \neq 0) \\ 1, & (x = 0) \end{cases} \tag{5.10.2}$$

다음과 같은 LaTeX문장을 사용해서, 함수 $f(x) = \mathrm{sinc}(x)$를 그릴 수 있다.

```
\begin{center}
 \begin{tikzpicture}
    \draw[help lines] (-3.5,-1.5) grid (3.5,5.5);
    \begin{scope}[very thick, domain=-pi:pi, samples=100, smooth]
      \draw[blue] plot (\x,{sin(5*\x r)/\x});
    \end{scope}
    \draw (0.2,-0.2 ) node {0};
    \draw[black,very thick] (-3.5,0) -- (3.5,0);
    \draw[black,very thick] (0,-1.5) -- (0,5.5);
  \end{tikzpicture}
\end{center}
```



방정식 $f(x) = 0$의 근들이 $k\pi, (k = \pm 1, \pm 2, \cdots)$임은 명백하다. 이분할법과 Newton-Raphson법을 같이 사용해서 이 방정식을 풀어보자.

예제 5.10.3 이분할법을 사용해서 방정식 $\mathrm{sinc}(x) = 0$의 근을 구하고 오차분석을 하기 위해서, 다음 MATLAB프로그램 BisectionMethodDY901.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: BisectionNewton901DY.m
% Combined Bisection and Newton-Raphson Method
% Programmed by CDY
% ------------------------------------------------------------------------
close all, clear all
% (Step 1) Preliminary
syms z
g = sinc(z/pi)  % g = sin(x)/x
dg = diff(g)
sol = solve(g)
solution = double(sol)
% (Step 2) Plotting
xx = linspace(-9*pi,9*pi,1001);
yy = sinc(xx/pi);
plot(xx,yy,'r-',xx,xx-xx,'k-',[0 0],[-1, 2],'k-','linewidth',1.5)
set(gca,'fontsize',11,'fontweigh','bold')
grid on
xlabel('\bf\it x')
ylabel('\bf\ity','rotation',0);
axis([-25 25 -0.3 1.1])
% % Combined Newton Method
% (Step 3) Problem Setup
f = @(x) sinc(x/pi)
df = @(x) cos(x)/x-sin(x)/x^2
a = -10; b = 10; nprob = 11;
subpoint = linspace(a,b,nprob);
a = subpoint(1:nprob-1)
b = subpoint(2:nprob)
tol = 10^-7;
```

```
RootNumber = 1;
% (Step 4) Finding Roots
for ii=1:nprob-1
    if f(a(ii)) == 0     % if-100
        RootOut(RootNumber) = a(ii);
        RootNumber = RootNumber + 1;
    elseif f(b(ii)) == 0
        RootOut(RootNumber) = b(ii);
        RootNumber = RootNumber + 1;
    elseif f(a(ii))*f(b(ii)) < 0
        x = b(ii);
        while abs(f(x))>= tol || abs(b(ii)-a(ii))...
                                >= tol*(1+abs(a(ii)));
             xnt = x - f(x)/df(x);
             if (a(ii) < xnt)*(xnt < b(ii)) && ...
                     (abs(f(xnt)) <= 0.5*abs(f(x)));
                 x = xnt;
             else
                 x = (a(ii)+b(ii))/2;
             end
             if f(a(ii))*f(x) < 0;
                b(ii) = x;
             else
                a(ii) = x;
             end
        end  % end_while
   end % end_if-100
   RootOut(RootNumber) = x;
   RootNumber = RootNumber + 1;
end % end_for
% (Step 5) Output
RootOut
Roots = unique(RootOut)   % eliminate replicated roots
saveas(gcf,'BisectionNewton901DY.jpg')
save('BisectionNewton901DY.txt','RootOut','Roots','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 BisectionNewton901DY.m을 실행하면, 제1단계에서 sinc 함수의 도함수를 구하고 또한 MATLAB의 Symbolic Math Toolbox를 사용해서 $\mathrm{sinc}(x) = 0$의 근을 구한다.

이 MATLAB프로그램을 실행하면, 함수 $f(x) = \mathrm{sinc}(x)$의 도함수는 다음과 같음을 알 수 있다.

$$f'(x) = \frac{\cos x}{x} - \frac{\sin x}{x^2} \tag{1}$$

또한, π가 방정식 $\mathrm{sinc}(x) = 0$의 근임을 알 수 있다. 제2단계로 함수 $f(x) = \mathrm{sinc}(x)$의 그래프를 그린 그림 5.10.3가 출력된다. 제3단계에서는 구간 $[-10, 10]$를 20개 소구간들로 나눈다. 제4단계에서는 Newton-Raphson법과 이분할법을 적용한다. 즉, 만약 제$n+1$번째 단계에서 $|x_n - f(x_n)/f'(x_n)|$의 절대값이 작으면, 이 값을 x_{n+1}으로 하고, 그렇지 않으면 다시 이분할법을 적용해서 x_{n+1}을 구한다. 제5단계에서는 출력된 근들을 정리한다. 즉, 다음과 같이 근들이 출력된다.

```
-9.4248   -9.4248   -9.4248   -6.2832   -6.2832   -6.2832
-3.1416   -3.1416   -3.1416   -3.1416   -3.1416   -3.1416
-3.1416    3.1416    3.1416    3.1416    6.2832    6.2832
 6.2832    9.4248
```

MATLAB함수 unique.m을 사용해서, 중복된 값들을 제외하면, 근들은 -9.4248, -6.2832, -3.1416, 3.1416, 6.2832, 그리고 9.4248이 방정식 $\mathrm{sinc}(x) = 0$의 근들이다. ■

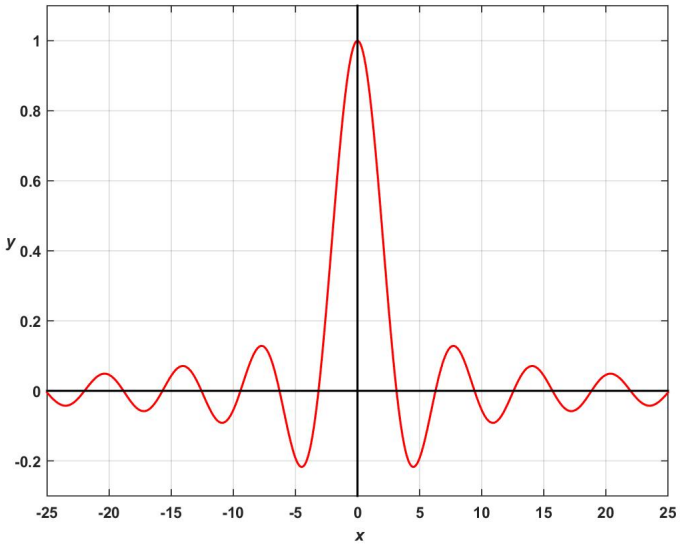


그림 5.10.3. sinc함수

제 6 장

최적화

함수의 최대값이나 최소값을 구하는 문제를 최적화문제(optimization problem)라 부른다. 함수 $f(x)$의 최대값은 함수 $-f(x)$의 최소값이므로, 지금부터는 함수 $f(x)$의 최소화에 대해서 주로 다룬다. 좀 더 엄밀히 말하자면, 최소(global minimum)가 아닌 극소(local minimum)를 주로 다루고자 한다. 최적화에 대한 자세한 내용은 Boyd & Vandenberghe [12]와 Cornuejols & Tütüncü [14]를 참조하라.

제6.1절 단변수함수의 최적화

단변수함수 $f(x)$가 폐구간 $[a,b]$에서 단봉이고(unimodal) 극소점을 하나만 갖는다고 가정하자. 구간의 하한인 a와 상한인 b를 선택하는 자체가 쉽지 않은 경우가 많지만, 이 절에서는 그냥 구간을 넓게 잡는 것으로 이 문제를 피해가기로 하자. 우선, 반복법을 적용해서 극소점의 존재범위를 좁혀 가는 방법에 대해서 살펴보고, 다음으로 2차함수를 사용해서 극소점을 찾는 방법을 살펴보자.

6.1.1 황금섹션탐색법

구간 (a,b)에 함수 $f(x)$의 극소점이 존재한다고 가정하자. 또한 부등식 $f(a) > f(b)$가 성립한다고 가정하자. 그림 6.1.1의 좌측상단 그래프에서는 부등식 $f\left(\frac{a+b}{2}\right) > f(b)$가 성립하고 구간 $\left(\frac{a+b}{2}, b\right)$에서 극소값을 갖는다. 우측상단 그래프에서는 부등식 $f\left(\frac{a+b}{2}\right) > f(b)$가 성립하고 구간 $\left(\frac{a+b}{2}, b\right)$에서 극소값을 갖는다. 즉, 세 그래프들 모두에서 부등식 $f\left(\frac{a+b}{2}\right) > f(b)$가 성립하지만, 이 부등식이 극소값의 위치를 정하는 데 기여를 하지 못한다. 반면에, 좌측하단 그래프에서는 부등식 $f\left(\frac{a+b}{2}\right) > f(b)$가 성립하고 구간 $\left(a, \frac{a+b}{2}\right)$에서 극소값을 갖는다. 따라서,

극소값을 구할 때 이분할법은 적당하지가 않다. 우측하단 그래프처럼 구간 (a,b)를 3등분해서 조사하면, 부등식 $f\left(\frac{a+2b}{3}\right) < f(a)$인 경우 구간 $\left(a, \frac{a+2b}{3}\right]$에서 극소값을 갖는다는 것을 알 수 있다. 따라서, 최적화문제에서는 이분할법보다 삼분할법(trisection method)이 더 유용하다. 그러나, 이렇게 등간격으로 나누는 삼분할법은 점 $\frac{2a+b}{3}$에서 계산된 함수값 $f\left(\frac{2a+b}{3}\right)$을 다음 단계에서 이용할 수 없기때문에 계산효율적이 아니다.

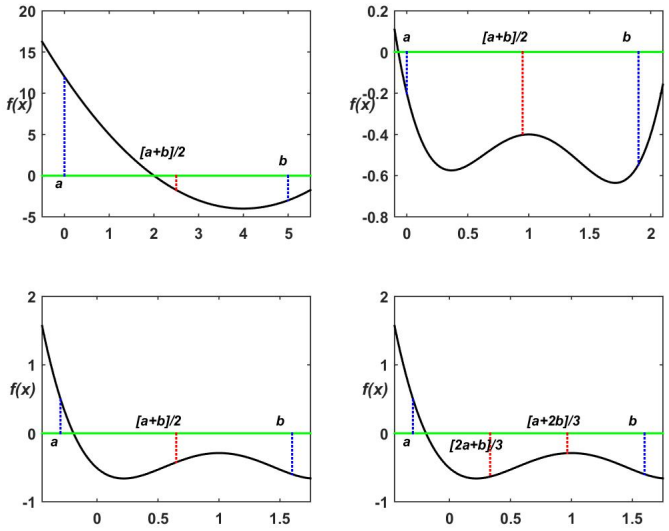


그림 6.1.1. 황금섹션탐색법 I

삼분할법을 효율적으로 적용할 수 있는 두 내점들을 선택하는 방법을 모색해보자. 구간 $[a,b]$를 세 소구간들 $[a,u]$, $[u,v]$, $[v,b]$로 나눈다고 하자. 선택된 v에 대해서 다음 단계에서는 구간 $[u,b]$를 세 소구간들 $[u,v]$, $[v,w]$, $[w,b]$로 나눈다고 하자. 다음 식들이 성립한다면, 우리는 효율적으로 삼분할법을 적용할 수 있다.

$$\frac{v-u}{u-a} = \frac{w-v}{v-u} = \frac{b-w}{b-v} = \frac{b-u}{b-a} \tag{6.1.1}$$

식 (6.1.1)의 비를 ϕ라 하면, 다음 식들이 성립한다.

$$b-u = \phi[b-a], \tag{6.1.2}$$

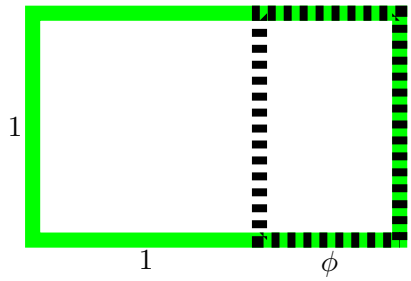
$$v-u = 1-\phi[b-u] = \phi[1-\phi][b-a], \tag{6.1.3}$$

$$b-v = [1-\phi][b-a] \tag{6.1.4}$$

식 (6.1.2) $\sim$ 식 (6.1.4)에서 알 수 있듯이, 다음 식이 성립한다.

$$\phi^2 + \phi - 1 = 0 \tag{6.1.5}$$

즉, $\phi = [\sqrt{5}-1]/2$이다. 이 ϕ를 황금비율(golden ratio)이라 부른다. 왜 ϕ를 황금비율이라 부르는 것일까? 이것은 일종의 신화(myth)에서 온 것이다. 즉, 황금비율로 보이는 모양이 가장 아름답다는 ... . Fibonacci수열, 피라미드, 인간의 얼굴, 심지어는 주가분석에 사용되는 Elliott파동 등이 황금비율과 연관되어 있다. 눈치가 빠른 독자들 중에는 중학교나 고등학교 미술시간에 황금비율을 정의할 때 사용된 직사각형들을 구성하는 변들이 식 (6.1.1)을 만족함을 알았을 것이다. 그 시절로 돌아가서 다음 그림을 살펴보자.



이 그림에서 녹색 실선으로 표시된 큰 직사각형의 가로 대 세로 비율과 흑색 점선으로 표시된 작은 직사각형의 가로 대 세로 비율은 같다. 이러한 두 직사각형들은 닮은꼴이고, 이들은 척도유사성(scale similarity)을 갖는다고 한다. 척도유사성에 의해서 다음 식이 성립함을 알 수 있다.

$$\frac{1}{1+\phi} = \frac{\phi}{1} \tag{6.1.6}$$

식 (6.1.5)와 식 (6.1.6)은 같다. 즉, 황금비율은 이 두 직사각형들이 척도유사성을 갖게하는 비율이다.

황금비율을 바탕으로 극소점을 탐색하는 황금섹션탐색법(golden section method)을 살펴보기 위해서, 다음 MATLAB프로그램 GoldenSectionGraph101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: GoldenSectionGraph101.m
%  Golden Section Graph for Optimization
%  Programmed by CBS
% ------------------------------------------------------------------------
close all, clear all, format short
ftn1 = @(x) (x-0.35).^2;
ftn2 = @(x) (x-0.65).^2;
phi = (sqrt(5)-1)/2
phi2 = phi*phi
phi3 = phi*phi2
phi4 = 1 - phi + phi2
epss = 0.02;
xx = linspace(0,1,201);
y1 = ftn1(xx);
y2 = ftn2(xx);
% Upper plot
```

```
subplot(2,1,1)
hold on
plot(xx,y1,'b-','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold')
axis([ -0.2, 1.2, -0.3, 0.5 ])
plot([-0.2, 1.2], [-0.1,-0.1],'g-','LineWidth',2)
plot(0,-0.1,'k.',phi2,-0.1,'k.',phi,-0.1,'k.',1,-0.1,'k.','LineWidth',8)
plot([0,0],[-0.2,ftn1(0)],'r:','LineWidth',2)
plot([phi2,phi2],[-0.2,ftn1(phi2)],'r:','LineWidth',2)
plot([phi,phi],[-0.2,ftn1(phi)],'r:','LineWidth',2)
plot([1,1],[-0.1,ftn1(1)],'r:','LineWidth',2)
text(0+epss/1.5,-0.1-epss,'a'), text(phi2+epss/1.5,-0.1-epss,'c')
text(phi+epss/1.5,-0.1-epss,'d'), text(1+epss/1.5,-0.1-epss,'b')
plot([-0.2, 1.2], [-0.2,-0.2],'g-','LineWidth',2)
plot([phi3,phi3],[-0.2,ftn1(phi3)],'k--','LineWidth',2)
text(0,-0.2-2*epss,'a^*'),  text(phi3,-0.2-2*epss,'c^*')
text(phi2,-0.2-2*epss,'d^*'), text(phi,-0.2-2*epss,'b^*')
axis('off')
hold off
% Lower plot
subplot(2,1,2)
hold on
plot(xx,y2,'b-','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold')
axis([ -0.2, 1.2, -0.3, 0.5 ])
plot([-0.2, 1.2], [-0.1,-0.1],'g-','LineWidth',2)
plot(0,-0.1,'k.',phi2,-0.1,'k.',phi,-0.1,'k.',1,-0.1,'k.','LineWidth',8)
plot([0,0],[-0.1,ftn2(0)],'r:','LineWidth',2)
plot([phi2,phi2],[-0.2,ftn2(phi2)],'r:','LineWidth',2)
plot([phi,phi],[-0.2,ftn2(phi)],'r:','LineWidth',2)
plot([1,1],[-0.2,ftn2(1)],'r:','LineWidth',2)
text(0+epss/1.5,-0.1-epss,'a'), text(phi2+epss/1.5,-0.1-epss,'c')
text(phi+epss/1.5,-0.1-epss,'d'), text(1+epss/1.5,-0.1-epss,'b')
plot([-0.2, 1.2], [-0.2,-0.2],'g-','LineWidth',2)
plot([phi4,phi4],[-0.2,ftn2(phi4)],'k--','LineWidth',2)
text(phi2,-0.2-2*epss,'a^*'), text(phi,-0.2-2*epss,'c^*'),
text(phi4,-0.2-2*epss,'d^*'), text(1,-0.2-2*epss,'b^*'),
axis('off')
hold off
saveas(gcf,'GoldenSectionGraph101','jpg')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 GoldenSectionGraph101.m을 실행하면, 그림 6.1.2를 출력한다. 그림 6.1.2에서처럼 폐구간 $[a,b]$의 내부에 식 $a < c < d < b$를 만족하는 점들 c와 d를 선택하고, 함수값들 $f(a), f(b), f(c)$와 $f(d)$를 계산한다. 만약 구간 $[a,b]$에서 한 점 c만 선택한다면, 극소점이 구간 $[a,c]$와 구간 $[c,b]$ 중 어느 쪽에 있는지를 판단할 수 없다. 따라서, 구간 $[a,b]$에서 두 점들 c와 d를 선택할 필요가 있다. 만약 식 $f(c) < f(d)$가 성립하면, 극소점은 구간 $[d,b]$에 존재하지 않고 구간 $[a,d]$에 존재한다. 반면에, 만약 식 $f(c) > f(d)$가 성립하면, 극소점은 구간 $[a,c]$에 존재하지 않고 구간 $[c,b]$에 존재한다. 만약 식 $f(c) = f(d)$가 성립하면, 극소점은 구간 $[c,d]$에 존재하지만, 편의상 극소점이 구간 $[a,d]$에 존재한다고 하자. 따라서, 만약 식 $f(c) \leq f(d)$가 성립하면, 구간 $[a,d]$를 새로운 $[a^*,b^*]$로 놓는다. 만약 식 $f(c) > f(d)$

가 성립하면, 구간 $[c,b]$를 새로운 $[a^*,b^*]$로 놓는다. 이러한 과정을 새로운 구간 새로운 $[a^*,b^*]$의 길이 b^*-a^*가 미리 정한 값 $\epsilon(>0)$보다 작아질 때까지 반복 수행한다.

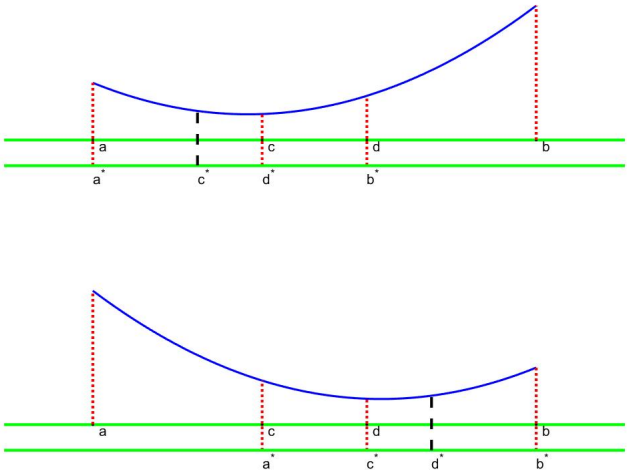


그림 6.1.2. 황금섹션탐색법 I

각 단계에서 구간폭을 갱신할 때, 전단계에서 계산한 함수값들을 다음 탐색에서도 사용할 수 있도록 하는 것은 매우 바람직하다. 문제를 간단히 하기 위해서, 다음과 같이 점들을 선택한다고 하자.

$$a=0, \quad c=\phi^2, \quad d=\phi, \quad b=1 \tag{6.1.7}$$

여기서 ϕ는 식 (6.1.5)를 만족하는 황금비율이다. 이 경우에 이 네 점들의 간격은 다음과 같은 비를 이룬다.

$$\phi^2:[\phi-\phi^2]:[1-\phi]=[1-\phi]:[\phi-\phi^2]:[1-\phi]=1:\phi:1 \tag{6.1.8}$$

그림 6.1.2의 상단 그래프처럼 구간폭을 1회 갱신해서 새로운 구간 $[a^*,b^*]$가 $[0,\phi]$라고 하면, 새로운 d^*로서 바로 전단계 c인 ϕ^2를 사용하고 새로운 c^*로 ϕ^3을 사용하자. 다음 식들이 성립한다.

$$\phi^3=\phi^2\phi=[1-\phi]\phi \tag{6.1.9}$$

따라서 새로운 네 점들 a^*, c^*, d^*, b^*의 간격들은 다음과 같은 비를 이룬다.

$$\phi^3:[\phi^2-\phi^3]:[\phi-\phi^2]=\phi[1-\phi]:\phi^2[1-\phi]:\phi[1-\phi]=1:\phi:1 \tag{6.1.10}$$

즉, 네 점들사이 간격들은 전단계 네 점들사이 간격들과 비례한다. 식 (6.1.8)을 사용해서 식 (6.1.10)을 증명할 수도 있다. 그림 6.1.2의 하단 그래프처럼 구간폭을 1회 갱신해서 새로운 구간 $[a^*, b^*]$가 $[\phi^2, 1]$인 경우에도 마찬가지이다. 이 경우에, 새로운 c^*로서 바로 전단계 d인 ϕ를 사용하고 새로운 d^*로 $1-\phi+\phi^2 = 2[1-\phi]$를 사용하자. 이 경우에 새로운 네 점들 a^*, c^*, d^*, b^*사이 간격들은 다음과 같은 비를 이룬다.

$$[\phi-\phi^2] : \{2[1-\phi]-\phi\} : \{1-2[1-\phi]\} = [\phi-\phi^2] : [1-2\phi+\phi^2] : [\phi-\phi^2]$$
$$= [2\phi-1] : \phi[2\phi-1] : [2\phi-1] = 1 : \phi : 1 \tag{6.1.11}$$

예제 6.1.1 Python을 사용해서 그림 6.1.2를 그리기 위해서, 다음 Python프로그램 GoldenSectionGraph101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

ftn1 = lambda x: (x-0.35)**2;
ftn2 = lambda x: (x-0.65)**2;
phi = (np.sqrt(5.0)-1.0)/2.0;
phi2 = phi*phi;
phi3 = phi*phi2;
phi4 = 1 - phi + phi2;
epss = 0.04;
xx = np.linspace(0,1,201);
y1 = ftn1(xx);
y2 = ftn2(xx);

# plotting
fig = plt.figure()
ax1 = fig.add_subplot(211)
plt.plot(xx,y1,'b-',lw=1.5);
ax1.axis([ -0.2, 1.2, -0.3, 0.5 ])
plt.plot([-0.2, 1.2], [-0.1,-0.1],'g-',lw=2)
plt.plot(0,-0.1,'k.',phi2,-0.1,'k.',phi,-0.1,'k.',1,-0.1,'k.',lw=8)
plt.plot([0,0],[-0.2,ftn1(0)],'r:',lw=2)
plt.plot([phi2,phi2],[-0.2,ftn1(phi2)],'r:',lw=2)
plt.plot([phi,phi],[-0.2,ftn1(phi)],'r:',lw=2)
plt.plot([1,1],[-0.1,ftn1(1)],'r:',lw=2)
plt.text(0+epss/1.5,-0.1-epss,'a'), plt.text(phi2+epss/1.5,-0.1-epss,'c')
plt.text(phi+epss/1.5,-0.1-epss,'d'), plt.text(1+epss/1.5,-0.1-epss,'b')
plt.plot([-0.2, 1.2], [-0.2,-0.2],'g-',lw=2)
plt.plot([phi3,phi3],[-0.2,ftn1(phi3)],'k--',lw=2)
plt.text(0,-0.2-2*epss,r'$a^*$'),  plt.text(phi3,-0.2-2*epss,r'$c^*$')
plt.text(phi2,-0.2-2*epss,r'$d^*$'), plt.text(phi,-0.2-2*epss,r'$b^*$')
ax1.axis('off')
plt.xlabel('x'); plt.ylabel('f(x)')
# plt.legend(loc='upper right', numpoints=1)
ax2 = fig.add_subplot(212)
```

```
plt.plot(xx,y2,'b-',lw=1.5);
ax2.axis([ -0.2, 1.2, -0.3, 0.5 ])
plt.plot([-0.2, 1.2], [-0.1,-0.1],'g-',lw=2)
plt.plot(0,-0.1,'k.',phi2,-0.1,'k.',phi,-0.1,'k.',1,-0.1,'k.',lw=8)
plt.plot([0,0],[-0.1,ftn2(0)],'r:',lw=2)
plt.plot([phi2,phi2],[-0.2,ftn2(phi2)],'r:',lw=2)
plt.plot([phi,phi],[-0.2,ftn2(phi)],'r:',lw=2)
plt.plot([1,1],[-0.2,ftn2(1)],'r:',lw=2)
plt.text(0+epss/1.5,-0.1-epss,'a'), plt.text(phi2+epss/1.5,-0.1-epss,'c')
plt.text(phi+epss/1.5,-0.1-epss,'d'), plt.text(1+epss/1.5,-0.1-epss,'b')
plt.plot([-0.2, 1.2], [-0.2,-0.2],'g-',lw=2)
plt.plot([phi4,phi4],[-0.2,ftn2(phi4)],'k--',lw=2)
plt.text(phi2,-0.2-2*epss,r'$a^*$'), plt.text(phi,-0.2-2*epss,r'$c^*$'),
plt.text(phi4,-0.2-2*epss,r'$d^*$'), plt.text(1,-0.2-2*epss,r'$b^*$'),
ax2.axis('off')
plt.show()
fig.savefig('GoldenSectionGraph101Py.png')

# End of Program
```

■

식 (6.1.10)과 식 (6.1.11)에서 알 수 있듯이, 황금섹션탐색법의 과정을 1회 반복마다 각 구간길이는 ϕ배가 되므로, 이러한 구간 갱신을 M회 반복하면 구간폭은 원래 구간폭의 ϕ^M 배가 된다. 즉, 구간폭은 지수적으로 감소한다. 구간 $[a,b]$에서 황금비를 이용한 최적화 알고리즘을 정리하면 다음과 같다.

알고리즘 6.1.1: 황금비를 사용한 직접탐색법

함수 $f(x)$가 구간 $[a,b]$에서 단봉이고 극소점을 하나 갖는다고 가정하자. 다음과 같은 단계들을 수행해서, 극소점을 구한다.

(1단계) 다음 값들을 계산한다.

$$c = a + [b-a]\phi^2, \quad d = a + [b-a]\phi$$

또한 함수값들 $f(a), f(b), f(c)$와 $f(d)$를 계산한다.

(2단계) 만약 $b-a < \epsilon$이면, 이 알고리즘의 수행을 멈추고 결과를 출력한다. 그렇지 않으면서, 만약 $f(c) \le f(d)$이면, 제3A단계로 넘어간다. 반면에, 만약 $f(c) > f(d)$이면, 제3B단계로 넘어간다.

(3A단계) 다음과 같이 새로운 값들을 할당하고, 함수값 $f(c)$를 계산한 다음 제2단계로

돌아간다.

$$d \Rightarrow b, \quad c \Rightarrow d, \quad a + [b-a]\phi^2 \Rightarrow c$$

(3B단계) 다음과 같이 새로운 값들을 할당하고, 함수값 $f(d)$를 계산한 다음 제2단계로 돌아간다.

$$c \Rightarrow a, \quad d \Rightarrow c, \quad a + [b-a]\phi \Rightarrow d$$

황금섹션탐색법은 만능(fool-proof) 알고리즘이다. 만약 $f(x)$가 구간 $[a, b]$에서 극소값을 갖는 단봉(unimodal) 연속함수이면, 황금섹션탐색법을 사용해서 항상 극소값을 구할 수 있다. 그러나, 황금섹션탐색법은 매우 느린 알고리즘이다. 즉, 극소값을 포함하는 구간의 길이를 머신엡스(machine eps)까지 줄이는데 75 반복단계를 필요로 한다.

예제 6.1.2 다음과 같은 2차함수를 살펴보자.

$$f(x) = [x-1]^2, \quad (0 \le x \le 2) \tag{1}$$

이 함수가 점 $x = 1$에서 극소값 0을 갖음은 확실하다. 또한, 이 함수는 황금섹션탐색법의 조건을 만족한다. 황금섹션탐색법을 적용해서 이 극소값을 구하기 위해서, 다음 MATLAB 프로그램 GoldenSectionMethod101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename: GoldenSctionMethod101.m
% Golden Section Method 1 for searching a local maximum
% Programmed by CBS
% ------------------------------------------------------------------
fff = inline('(x-1).^2','x')
tol = 10^-8;
% Initialize
a = 0; b = 2;
phi = (sqrt(5)-1)/2; mphi = 1-phi;
x1 = a + (b-a)*mphi; x2 = a + (b-a)*phi;
f1 = fff(x1); f2 = fff(x2);
for ii = 1:1:100
    if f1 > f2,
        a = x1; x1 = x2; f1 = f2;
        x2 = a + (b-a)*phi;
        f2 = fff(x2);
    else
        b = x2; x2 = x1; f2 = f1;
        x1 = a + (b-a)*mphi;
        f1 = fff(x1);
    end
    aa(ii) = a; bb(ii) = b;
    xx1(ii) = x1; xx2(ii) = x2;
    ff1(ii) = f1; ff2(ii) = f2;
    if abs(b-a) < tol,
```

```
        break
    end
end
ii, b-a, [ aa; bb; xx1; xx2; ff1; ff2 ]'
% plot
nn = 1:1:ii;
plot(nn,aa,'k:',nn,bb,'m-.',nn,xx1,'r--',nn,xx2,'b-','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
legend('a','b','c','d','location','SE')
xlabel('\bf Iteration Number','fontsize',12)
ylabel('Value','fontsize',12)
axis( [ 1 ii 0 1.5 ] )
saveas(gcf,'GoldenMethod101','jpg')
save('GoldenMethod101.txt','xx1','xx2','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 GoldenSectionMethod101.m을 실행하면, 제40번째 구간 갱신 후 다음 조건이 만족된다.

$$b - a < 10^{-8} \tag{2}$$

이 경우에, $b - a = 8.7 \times 10^{-9}$ 이다. 또한 다음 값들을 출력한다.

$$a = c = d = b = 1 \tag{3}$$

$$f(c) = f(d) = 0 \tag{4}$$

이 MATLAB프로그램 GoldenSectionMethod101.m을 실행하면, 그림 6.1.3도 출력한다. ■

예제 6.1.3 Python을 사용해서 예제 6.1.2를 다시 다루기 위해서, 다음 Python프로그램 GoldenSectionMethod101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

fff = lambda x: (x-1)**2;
tol = 10**-8;
# Initialize
a = 0; b = 2;
phi = (np.sqrt(5)-1.0)/2.0; mphi = 1-phi;
x1 = a + (b-a)*mphi; x2 = a + (b-a)*phi;
f1 = fff(x1); f2 = fff(x2);
aa = [0.0]*100; bb = [0.0]*100;
xx1 = [0.0]*100; xx2 = [0.0]*100;
```

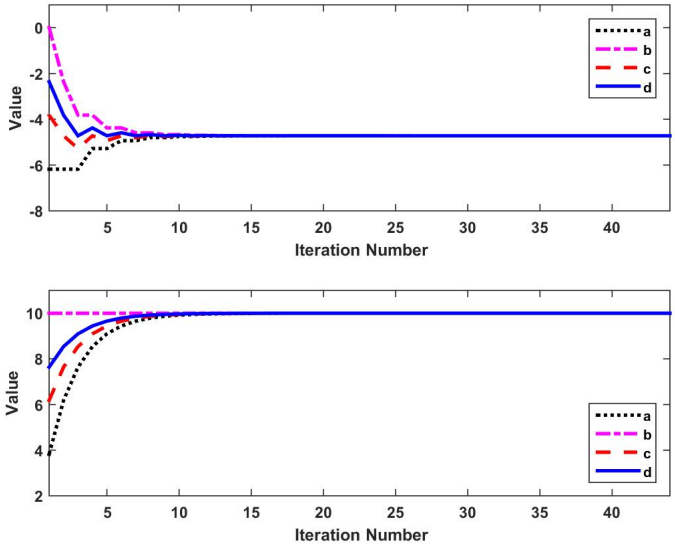


그림 6.1.3. 황금섹션탐색법 1

```
ff1 = [0.0]*100; ff2 = [0.0]*100;
for ii in range(1,100):
    if f1 > f2:
        a = x1; x1 = x2; f1 = f2;
        x2 = a + (b-a)*phi;
        f2 = fff(x2);
    else:
        b = x2; x2 = x1; f2 = f1;
        x1 = a + (b-a)*mphi;
        f1 = fff(x1);
    aa[ii] = a; bb[ii] = b;
    xx1[ii] = x1; xx2[ii] = x2;
    ff1[ii] = f1; ff2[ii] = f2;
    if abs(b-a) < tol:
        break

print(ii, b-a)
print( aa[0:ii], bb[0:ii], xx1[0:ii], xx2[0:ii], ff1[0:ii], ff2[0:ii] )

# Plotting
fig = plt.figure()
nn = np.arange(ii);
plt.plot(nn,aa[0:ii],'k:',lw=2,label='a')
plt.plot(nn,bb[0:ii],'m-.',lw=2,label='b')
plt.plot(nn,xx1[0:ii],'r--',lw=2,label='c')
plt.plot(nn,xx2[0:ii],'b-',lw=2,label='d')
plt.legend(loc='lower right', numpoints=1)
plt.xlabel('Iteration Number', fontsize=12)
plt.ylabel('Value', fontsize=12)
plt.axis( [ 1, ii, 0, 1.5 ] )
plt.show()
fig.savefig('GraphicalMethod101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 6.1.2의 결과와 같다. ■

황금섹션탐색법을 사용해서 Cauchy확률분포의 모수를 추정하기로 하자.

예제 6.1.4 다음과 같이 위치모수를 θ로 하는 Cauchy확률밀도함수를 살펴보자.

$$f(x|\theta) = \frac{1}{\pi\{1+[x-\theta]^2\}} \tag{1}$$

이 Cauchy확률밀도함수에서 추출된 관찰값들이 $-5, 2$ 그리고 4라고 하면, 우도함수 $L(\theta)$와 로그우도함수 $l(\theta)$는 각각 다음과 같다.

$$L(\theta) = \frac{1}{\pi^3\{1+[-5-\theta]^2\}\{1+[2-\theta^2]\}\{1+[4-\theta^2]\}} \tag{2}$$

$$l(\theta) = -3\ln\pi - \ln(1+[-5-\theta]^2) - \ln(1+[2-\theta^2]) - \ln(1+[4-\theta^2]) \tag{3}$$

황금섹션탐색법을 적용해서 최우추정값을 구하기 위해서, 다음 MATLAB프로그램 GoldenSectionMethod102.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: GoldenSctionMethod102.m
% Golden Section Method 2 for searching a local maximum
% Programmed by CBS
% ------------------------------------------------------------------------
function GoldenSectionMethod102
clear all, close all
mLL = @(theta) 3*log(pi)+log(1+(5+theta).^2) ...
               +log(1+(2-theta).^2)+log(1+(4-theta).^2)
tol = 1.0e-8;
% Plotting  the function
xxg = -10:0.01:10;
ffg = mLL(xxg);
plot(xxg,ffg,'k-','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf \theta','fontsize',12)
ylabel('\bf l(\theta)','fontsize',12,'rotation',0)
saveas(gcf,'GoldenSectionMethod102a','jpg')
figure
% For interval [-10,0]
a = -10; b = 0;
[ ii, aa, bb, xx1, xx2, ff1, ff2 ] = GoldenSection(mLL,a,b,tol)
ii, b-a, [ aa; bb; xx1; xx2; ff1; ff2 ]'
% Plotting
subplot(2,1,1)
nn = 1:1:ii;
plot(nn,aa,'k:',nn,bb,'m-.',nn,xx1,'r--',nn,xx2,'b-','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('a','b','c','d','location','NE')
xlabel('\bf Iteration Number','fontsize',12)
ylabel('Value','fontsize',12)
axis([ 1 ii -8 1 ])
% For interval [0,10]
a = 0; b =10;
[ ii, aa, bb, xx1, xx2, ff1, ff2 ] = GoldenSection(mLL,a,b,tol)
```

```
ii, b-a, [ aa; bb; xx1; xx2; ff1; ff2 ]'
% Plotting
subplot(2,1,2)
nn = 1:1:ii;
plot(nn,aa,'k:',nn,bb,'m-.',nn,xx1,'r--',nn,xx2,'b-','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
axis([ 1 ii 2 11 ])
legend('a','b','c','d','location','NE')
xlabel('\bf Iteration Number','fontsize',12)
ylabel('Value','fontsize',12)
saveas(gcf,'GoldenSectionMethod102b','jpg')
save('GoldenSectionMethod102.txt','xx1','xx2','-ascii')
end
% End of program
% ------------------------------------------------------------------------
function [ ii, aa, bb, xx1, xx2, ff1, ff2 ] = GoldenSection(ftn,a,b,tol)
phi = (sqrt(5.0)-1.0)/2.0;  mphi = 1-phi;
x1 = a + (b-a)*mphi; x2 = a + (b-a)*phi;
f1 = ftn(x1); f2 = ftn(x2);
for ii = 1:1:100
    if f1 > f2,
        a = x1; x1 = x2; f1 = f2;
        x2 = a + (b-a)*phi;
        f2 = ftn(x2);
    else
        b = x2; x2 = x1; f2 = f1;
        x1 = a + (b-a)*mphi;
        f1 = ftn(x1);
    end
    aa(ii) = a; bb(ii) = b;
    xx1(ii) = x1; xx2(ii) = x2;
    ff1(ii) = f1; ff2(ii) = f2;
    if abs(b-a) < tol,
        break
    end
end
end
% ------------------------------------------------------------------------
```

황금섹션탐색법을 적용하기 위해서, 먼저 다음과 같이 로그우도함수 $l(\theta)$에 $[-1]$을 곱한 함수 $l_{-}(\theta)$를 만들어야 한다.

$$l_{-}(\theta) = 3\ln(\pi) + \ln(1 + [-5 - \theta]^2) + \ln(1 + [2 - \theta]^2) + \ln(1 + [4 - \theta]^2) \tag{1}$$

이 MATLAB프로그램 GoldenSctionMethod102.m을 실행해서 출력한 $l_{-}(\theta)$의 그래프가 그림 6.1.4에 수록되어 있다. 그림 6.1.4에서 알 수 있듯이, 이 목적함수는 구간 $[-10, 10]$에서 극소값들을 2개 갖는다. 하나는 $\theta = -5$ 부근이고 다른 하나는 $\theta = 2.5$ 부근이다.

첫 번째 극소값을 구하기 위해서, 초기구간을 $[a, b] = [-10, 0]$으로 하는 황금섹션탐색법을 적용하면, 제44번째 구간 갱신 후 다음 조건이 만족된다.

$$b - a < 10^{-8} \tag{8}$$

이 경우에, $b-a=6.4\times10^{-9}$ 이다. 또한, 다음 값들을 출력한다.

$$a=c=d=b=-4.7211 \tag{5}$$

$$f(c)=f(d)=11.6861 \tag{6}$$

이 a,b,c와 d의 변화를 그린 것이 그림 6.1.5의 상단 그래프이다. 이 그래프에서 이 네 값들이 한 점으로 모아짐을 알 수 있다.

두 번째 극소값을 구하기 위해서, 초기구간을 $[a,b]=[0,10]$ 으로 하는 황금섹션탐색법을 적용하면, 제44번째 구간 갱신 후 다음 조건이 만족된다.

$$b-a<10^{-8} \tag{7}$$

이 경우에, $b-a=6.4\times10^{-9}$ 이다. 또한 다음 값들을 출력한다.

$$a=c=d=b=2.3462 \tag{8}$$

$$f(c)=f(d)=8.8719 \tag{9}$$

이 a,b,d와 d의 변화를 그린 것이 그림 6.1.5의 하단 그래프이다. 이 그래프에서 이 네 값들이 한 점으로 모아짐을 알 수 있다. 결론적으로, 최우추정값은 $\hat{\theta}=2.3462$ 이다. ■

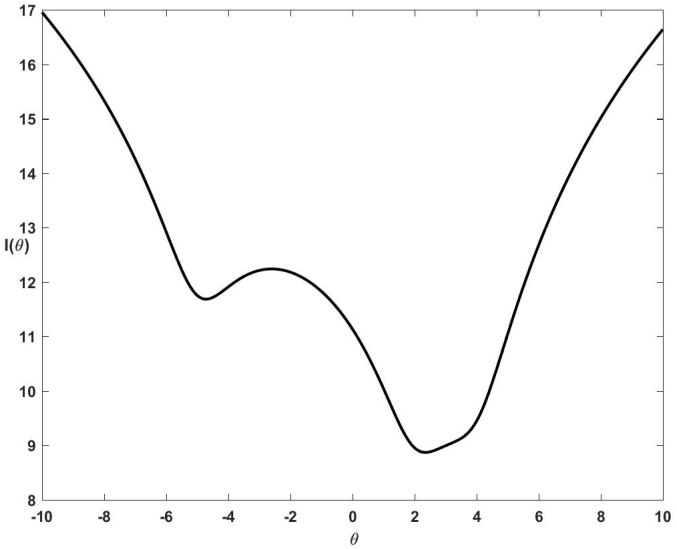


그림 6.1.4. $[-1]\cdot$로그우도함수

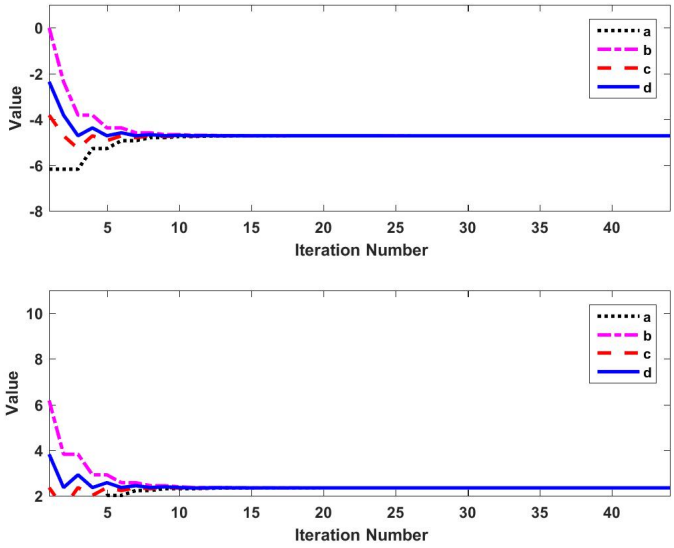


그림 6.1.5. 황금섹션탐색법 II

예제 6.1.5 Python을 사용해서 예제 6.1.4을 다시 다루기 위해서, 다음 Python프로그램 GoldenSectionMethod102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

def GoldenSection(ftn,a,b,tol):
    phi = (np.sqrt(5)-1.0)/2.0; mphi = 1-phi;
    x1 = a + (b-a)*mphi; x2 = a + (b-a)*phi;
    f1 = ftn(x1); f2 = ftn(x2);
    aa = [0.0]*100; bb = [0.0]*100;
    xx1 = [0.0]*100; xx2 = [0.0]*100;
    ff1 = [0.0]*100; ff2 = [0.0]*100;
    for ii in range(0,100):
        if f1 > f2:
           a = x1; x1 = x2; f1 = f2;
           x2 = a + (b-a)*phi;
           f2 = ftn(x2);
        else:
           b = x2; x2 = x1; f2 = f1;
           x1 = a + (b-a)*mphi;
           f1 = ftn(x1);
        aa[ii] = a; bb[ii] = b;
        xx1[ii] = x1; xx2[ii] = x2;
        ff1[ii] = f1; ff2[ii] = f2;
        if abs(b-a) < tol:
           break
    return ( ii, aa, bb, xx1, xx2, ff1, ff2 )

# Log Likelihood
mLL = lambda theta: 3*np.log(np.pi)+np.log(1+(5+theta)**2)        \
```

```
                +np.log(1+(2-theta)**2)+np.log(1+(4-theta)**2);
tol = 1.0e-8;

# Plotting  the function
fig = plt.figure()
xxg = np.linspace(-10,10,2001);
ffg = mLL(xxg);
plt.plot(xxg,ffg,'k-',lw=2)
plt.xlabel(r'$\theta$',fontsize=12)
plt.ylabel(r'$l(\theta)$',fontsize=12)
fig.savefig('GoldenSectionMethod102aPy.png')

# For interval [-10,0]
a = -10; b = 0;
[ iiA, aaA, bbA, xx1A, xx2A, ff1A, ff2A ] = GoldenSection(mLL,a,b,tol)
print(iiA, b-a)
print( aaA[0:iiA], bbA[0:iiA], xx1A[0:iiA], xx2A[0:iiA], \
       ff1A[0:iiA], ff2A[0:iiA] )
# For interval [0,10]
a = 0; b = 10;
[ iiB, aaB, bbB, xx1B, xx2B, ff1B, ff2B ] = GoldenSection(mLL,a,b,tol)
print(iiB, b-a)
print( aaB[0:iiB], bbB[0:iiB], xx1B[0:iiB], xx2B[0:iiB], \
       ff1B[0:iiB], ff2B[0:iiB] )

# Plotting
fig = plt.figure()
# For interval [-10,0]
ax1 = fig.add_subplot(211)
nn = np.arange(iiA);
plt.plot(nn,aaA[0:iiA],'k:',lw=2,label='a');
plt.plot(nn,bbA[0:iiA],'m-.',lw=2,label='b');
plt.plot(nn,xx1A[0:iiA],'r--',lw=2,label='c');
plt.plot(nn,xx2A[0:iiA],'b-',lw=2,label='d');
plt.legend(loc='upper right', numpoints=1);
plt.xlabel('Iteration Number');     plt.ylabel('Value')
ax1.axis([ 0, iiA, -8, 0 ])
# For interval [0,10]
ax1 = fig.add_subplot(212)
nn = np.arange(iiB);
plt.plot(nn,aaB[0:iiB],'k:',lw=2,label='a');
plt.plot(nn,bbB[0:iiB],'m-.',lw=2,label='b');
plt.plot(nn,xx1B[0:iiB],'r--',lw=2,label='c');
plt.plot(nn,xx2B[0:iiB],'b-',lw=2,label='d');
plt.legend(loc='upper right', numpoints=1);
plt.xlabel('Iteration Number');     plt.ylabel('Value')
ax1.axis([ 0, iiB, 0, 8])
plt.show()
fig.savefig('GoldenSectionMethod102Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 6.1.4의 결과와 같다. ■

황금섹션탐색법은 이분할법과 블래키팅법을 변형해서 만든 것이라고 할 수 있다. 황금비율에 해당하는 특성방정식 $x^2 - x - 1 = 0$의 근에서 최소값을 갖는 최적화문제를 살펴보자.

예제 6.1.6 다음 함수를 살펴보자.

$$f(x) = -\frac{x^3}{3} + \frac{x^2}{2} + x - 1 \tag{1}$$

이 함수를 미분하면 다음 식이 성립한다.

$$f'(x) = -x^2 + x + 1 \tag{2}$$

방정식 $f'(x) = 0$의 해는 $-\phi$와 ϕ^2이다. 여기서 ϕ는 황금섹션탐색법의 특성근이다. 황금섹션탐색법을 사용해서 함수 $f(x)$의 최소값을 구하기 위해서, 다음 MATLAB프로그램 GoldenSctionMethod103.m을 실행하자.

```
% ------------------------------------------------------------------------
% Filename: GoldenSctionMethod103.m
% Golden Section Method 3 for searching a local maximum
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
f = @(x) - 1/3*x.^3 + 1/2*x.^2 + x - 1
xmin = -1;                          % smallest point
xmax = 2;                           % largest point
a = xmin; b = xmax;
eps = 10^-8;                        % tolerance
Nmax = 50;                          % maximum number of iterations
phi = double((sqrt(5)-1)/2);        % golden proportion coefficient
jj = 0;                             % number of iterations
xc = a+(1-phi)*(b-a);               % computing x values
xd = a+phi*(b-a);
fc = f(xc); fd = f(xd);
xx = linspace(a,b,101);
ff = f(xx);
hold on;
plot(xx,ff,'g-',xx,xx-xx,'b-','LineWidth',1.5)
set(gca,'fontsize',11,'fontweigh','bold')
plot(xc,fc,'ko','LineWidth',1.5)
plot(xd,fd,'ko','LineWidth',1.5)
while ((abs(b-a)>eps) && (jj<Nmax))
    jj = jj+1;
    if fc<fd
        b = xd;
        xd = xc;
        xc = a+(1-phi)*(b-a);
        fc = f(xc); fd = f(xd);
        plot(xc,fc,'ko');
    else
        a = xc;
        xc = xd;
        xd = a+phi*(b-a);
        fc=f(xc); fd=f(xd);
        plot(xd,fd,'ko')
    end
    jj=jj+1;
```

```
end
% Minimum point
if(fc < fd)
    xstar = xc;
    sprintf('xmin = %f', xc)
    sprintf('f(xmin) = %f ', fc)
    plot(xc,fc,'r*',[xc xc],[0 fc],'k:','LineWidth',1.5)
else
    xstar = xd;
    sprintf('xmin = %f', xd)
    sprintf('f(xmin) = %f ', fd)
    plot(xd,fd,'r*',[xc xc],[0 fc],'k:','LineWidth',1.5)
end
fstar = min(fc,fd);
xlabel('\bf \it x','fontsize',12)
ylabel('\bf \it y','fontsize',12,'rotation',0)
saveas(gcf,'GoldenSectionMethod103','jpg')
save('GoldenSectionMethod103.txt','xstar','fstar','-ascii')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 그림 6.1.6이 그려진다. 그림 6.1.6에서 확인할 수 있듯이, 함수 $f(x)$는 $-\phi = \frac{1-\sqrt{5}}{2}$에서 최소값 -1.348362을 갖는다. ■

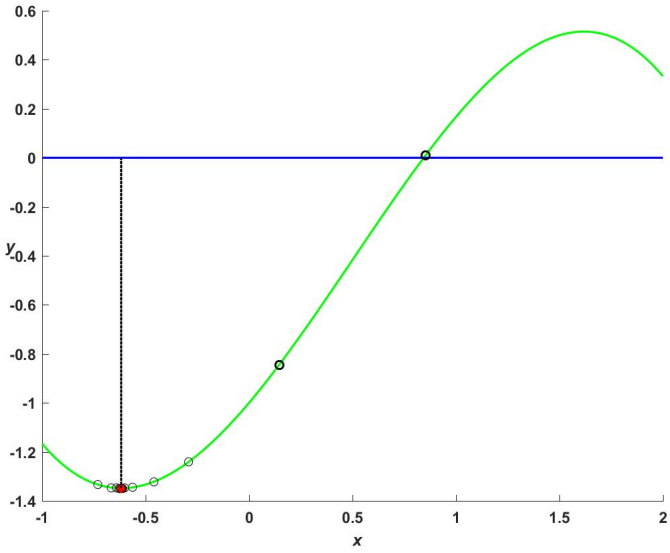


그림 6.1.6. 황금비율

예제 6.1.7 Python을 사용해서 예제 6.1.6을 다시 다루기 위해서, 다음 Python프로그램 GoldenSectionMethod103.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
```

```
5
6  import numpy as np
7  import matplotlib.pyplot as plt
8
9  f = lambda x: - 1/3*x**3 + 1/2*x**2 + x - 1;
10 xmin = -1;                          # smallest point
11 xmax = 2;                           # largest point
12 a = xmin; b = xmax;
13 eps = 10**-8;                       # tolerance
14 Nmax = 50;                          # maximum number of iterations
15 phi = (np.sqrt(5.0)-1.0)/2.0;       # golden proportion coefficient
16 jj = 0;                             # number of iterations
17 xc = a+(1-phi)*(b-a);               # computing x values
18 xd = a+phi*(b-a);
19 fc = f(xc); fd = f(xd);
20 xx = np.linspace(a,b,101);
21 ff = f(xx);
22
23 # Plotting
24 fig = plt.figure()
25 plt.plot(xx,ff,'g-',xx,xx-xx,'b-',lw=1.5)
26 plt.plot(xc,fc,'ko',lw=1.5)
27 plt.plot(xd,fd,'ko',lw=1.5)
28 while ((abs(b-a)>eps) and (jj<Nmax)):           # && -> and
29     jj = jj+1;
30     if fc<fd:
31         b = xd;
32         xd = xc;
33         xc = a+(1-phi)*(b-a);
34         fc = f(xc); fd = f(xd);
35         plt.plot(xc,fc,'ko');
36     else:
37         a = xc;
38         xc = xd;
39         xd = a+phi*(b-a);
40         fc=f(xc); fd=f(xd);
41         plt.plot(xd,fd,'ko');
42     jj=jj+1;
43 # #Minimum point
44 if(fc < fd):
45     xstar = xc;
46     print('xmin = %f', xc)
47     print('f(xmin) = %f ', fc)
48     plt.plot(xc,fc,'r*',[xc, xc],[0, fc],'k:',lw=1.5)
49 else:
50     xstar = xd;
51     print('xmin = %f', xd)
52     print('f(xmin) = %f ', fd)
53     plt.plot(xd,fd,'r*',[xc, xc],[0, fc],'k:',lw=1.5)
54 fstar = min(fc,fd);
55
56 plt.xlabel('x',fontsize=12)
57 plt.ylabel('y',fontsize=12)
58 fig.savefig('GoldenSectionMethod103Py.png')
59
60 # End of Program
```

이 Python프로그램을 수행한 결과는 예제 6.1.6의 결과와 같다. ■

Venkataraman [63, 제5.2.5소절]에는 황금섹션탐색법이 자세히 소개되어 있다. 또한, 이 방법을 구현하기 위한 MATLAB프로그램 GoldSection_1Var.m이 제공되어 있다. 다음 웹페이지에서 Venkataraman 책에 수록된 프로그램을 다운받을 수 있다.

```
http://bcs.wiley.com/he-bcs/Books?action=resource&bcsId=4907
        &itemId=047008488X&resourceId=16186
http://www.wiley.com/legacy/products/subject/engineering/venkat/
```

다음 예제는 MATLAB프로그램 GoldSection_1Var.m을 사용하는 예이다.

예제 6.1.8 MATLAB프로그램 GoldSection_1Var.m을 사용하기 위해서는 MATLAB 프로그램 UpperBound_m1Var.m을 필요로 한다. 또한, 극대값을 구하고자 하는 목적함수를 지정하는 MATLAB프로그램이 필요하다.

Cauchy확률밀도함수의 모수 θ의 최우추정값을 구하는 문제를 다시 살펴보자. 이러한 목적을 달성하기 위해서는 먼저 로그우도함수에 $[-1]$을 곱한 함수 $l_{-}(\theta)$를 지정하는 MATLAB 프로그램을 작성해야 한다.

$$l_{-}(\theta) = 3\ln(\pi) + \ln(1+[-5-\theta]^2) + \ln(1+[2-\theta^2]) + \ln(1+[4-\theta^2]) \qquad (1)$$

여기서 로그우도함수에 $[-1]$을 곱하는 이유는 MATLAB프로그램 GoldSection_1Var.m이 극대값을 구하는 것이 아니라 극소값을 구하는 프로그램이기 때문이다. 즉, 다음과 같은 MATLAB프로그램 CauchyLogLikelihood.m을 작성한다.

```
% --------------------------------------------------------------------
function mLL = CauchyLogLikelihood(theta)
% Input function: (-1)*loglikelihood for GoldSection_1Var.m
mLL = 3*log(pi)+log(1+(5+theta)^2)+log(1+(2-theta)^2)+log(1+(4-theta)^2);
end
% --------------------------------------------------------------------
```

MATLAB프로그램 GoldSection_1Var.m의 사용법은 다음과 같다.

```
>> GoldSection_1Var('CauchyLogLikelihood',tol,lowbound,intvl,ntrials)
```

여기서 첫 번째 변수에는 'CauchyLogLikelihood'를 지정하였다. 이는 MATLAB프로그램 CauchyLogLikelihood.m을 의미하는 것으로, 이 MATLAB프로그램과 MATLAB프로그램 GoldSection_1Var.m은 동일한 디렉토리 안에 저장되어야 한다. 두 번째 입력변수 tol에는 최종적으로 원하는 구간의 길이 ϵ을 지정한다. 세 번째 입력변수 lowbound는 탐색하고자 하는 구간의 최소값 a를 지정한다. 네 번째와 다섯 번째는 탐색하고자 하는 구간의 최대값

b를 찾기 위한 MATLAB프로그램 UpperBound_m1Var.m에 입력되는 변수들을 지정한다. 이 MATLAB프로그램은 $a < a_1 < a_2 < b$를 만족하는 점들 a_1, a_2 그리고 b를 구하기 위한 것으로, 네 번째 변수 intvl은 이 네 점들 사이가 등간격으로 생각하고 이 등간격의 초기값을 지정하는 것이다. 다섯 번째 변수 ntrials는 이 점들을 찾는 시도를 몇 번할 것인가를 지정한다.

첫 번째 극대값을 구하기 위해서, MATLAB커맨드행에 다음 명령문을 입력하자.

```
>> GoldSection_1Var('CauchyLogLikelihood',1.0e-8,-10,2,9)
```

이 MATLAB명령문이 실행되면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

```
alphal(low)   alpha(1)   alpha(2)  alpha{up)
  -10.0000   -6.9442   -5.0558   -2.0000
   16.9522   14.1870   11.7838   12.1809

alphal(low)   alpha(1)   alpha(2)  alpha{up)
 -1.0000e+001 -6.9442e+000 -5.0558e+000 -2.0000e+000
  1.6952e+001  1.4187e+001  1.1784e+001  1.2181e+001

iteration      1
alphal(low)   alpha(1)   alpha(2)  alpha{up)
   -6.9442   -5.0558   -3.8886   -2.0000
   14.1870   11.7838   11.9598   12.1809

iteration      2
alphal(low)   alpha(1)   alpha(2)  alpha{up)
   -6.9442   -5.7771   -5.0558   -3.8886
   14.1870   12.5958   11.7838   11.9598

          ⋮         ⋮
iteration     42
alphal(low)   alpha(1)   alpha(2)  alpha{up)
   -4.7211   -4.7211   -4.7211   -4.7211
   11.6861   11.6861   11.6861   11.6861

iteration     43
alphal(low)   alpha(1)   alpha(2)  alpha{up)
   -4.7211   -4.7211   -4.7211   -4.7211
   11.6861   11.6861   11.6861   11.6861
```

이 결과물에서 alpha1(low)는 a, alpha(1)은 c, alpha(2)는 d 그리고 alpha(up)은 b이다. 이 MATLAB프로그램 UpperBound_m1Var.m에 의해서 선택된 b는 -2이다. 즉, 초기구간을 $[a,b]=[-10,-2]$로 하는 황금섹션탐색법을 적용하면, 43번째 구간 갱신 후 다음 식이 성립한다.

$$b-a<10^{-8} \tag{2}$$

또한, 다음 값들을 출력한다.

$$a=c=d=b=-4.7211 \tag{3}$$

$$f(c)=f(d)=11.6861 \tag{4}$$

두 번째 극대값을 구하기 위해서, MATLAB커맨드행에 다음 명령문을 입력하자.

```
>> GoldSection_1Var('CauchyLogLikelihood',1.0e-8,0,2,9)
```

이 MATLAB명령문이 실행되면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

```
alphal(low)   alpha(1)   alpha(2)  alpha{up)
         0    1.5279    2.4721    4.0000
   11.1349    9.3725    8.8798    9.4503

iteration      1
alphal(low)   alpha(1)   alpha(2)  alpha{up)
    1.5279    2.4721    3.0557    4.0000
    9.3725    8.8798    9.0085    9.4503

iteration      2
alphal(low)   alpha(1)   alpha(2)  alpha{up)
    1.5279    2.1115    2.4721    3.0557
    9.3725    8.9083    8.8798    9.0085

          ⋮         ⋮
iteration     40
alphal(low)   alpha(1)   alpha(2)  alpha{up)
    2.3462    2.3462    2.3462    2.3462
    8.8719    8.8719    8.8719    8.8719

iteration     41
```

```
alphal(low)   alpha(1)   alpha(2)  alpha{up)
   2.3462     2.3462     2.3462     2.3462
   8.8719     8.8719     8.8719     8.8719
```

이 MATLAB프로그램 UpperBound_m1Var.m에 의해서 선택된 b는 4이다. 즉, 초기구간을 $[a, b] = [0, 4]$로 하는 황금섹션탐색법을 적용하면, 41번째 구간 갱신 후 다음 식이 성립한다.

$$b - a < 10^{-9} \tag{5}$$

또한, 다음 값들을 출력한다.

$$a = c = d = b = 2.3462 \tag{6}$$

$$f(c) = f(d) = 8.8719 \tag{7}$$

항상 조심해야 하지만, 어떤 MATLAB프로그램을 돌리기 전에 반드시 'clear all'과 'close all'을 실행할 것을 권한다. 그렇지 않으면, 엉뚱한 결과에 매달리게 될수도 있다. ■

6.1.2 2차함수탐색법

제5.6절에서 다룬 Müller법과 같은 맥락으로, 목적함수 $f(x)$를 2차함수로 근사시킨 다음, 이 2차함수를 사용해서 극소값을 구하는 2차함수탐색법 (quadratic function search method)을 살펴보자.

구간 (a, b)에 속하는 점 c에 대해, 세 점들 $(a, f(a))$, $(c, f(c))$ 그리고 $(b, f(b))$를 지나는 2차함수 $P(x) \doteq Ax^2 + Bx + C$의 최소값을 함수 $f(x)$의 극소값의 근사값으로 선택한다. 다음 식들이 성립한다.

$$Aa^2 + Ba + C = f(a) \tag{6.1.12}$$

$$Ab^2 + Bb + C = f(b) \tag{6.1.13}$$

$$Ac^2 + Bc + C = f(c) \tag{6.1.14}$$

다음 값을 정의하자.

$$D \doteq -[a - b][b - c][c - a] \tag{6.1.15}$$

식 (6.1.12)∼식 (6.1.15)에서 알 수 있듯이, 다음 식들이 성립한다.

$$A = \frac{1}{D}\{[b-c]f(a) + [c-a]f(b) + [a-b]f(c)\} \tag{6.1.16}$$

$$B = -\frac{1}{D}\{[b^2-c^2]f(a) + [c^2-a^2]f(b) + [a^2-b^2]f(c)\} \tag{6.1.17}$$

$$C = -\frac{1}{D}\{bc[c-b]f(a) - ca[c-a]f(b) + ab[b-a]f(c)\} \tag{6.1.18}$$

함수 $P(x)$가 다음과 같은 점 x^*에서 극소값을 갖는 것은 명백하다.

$$x^* = -\frac{B}{2A} = \frac{1}{2}\left\{\frac{[b^2-c^2]f(a) + [c^2-a^2]f(b) + [a^2-b^2]f(c)}{[b-c]f(a) + [c-a]f(b) + [a-b]f(c)}\right\} \tag{6.1.19}$$

식 (6.1.19)을 좀 더 간단하게 구해보자. 다음 Lagrange보간함수를 살펴보자.

$$\begin{aligned} P_2(x) =& \frac{f(a)}{[a-b][a-c]}[x-b][x-c] + \frac{f(b)}{[b-c][b-a]}[x-c][x-a] \\ &+ \frac{f(c)}{[c-a][c-b]}[x-a][x-b] \end{aligned} \tag{6.1.20}$$

다음 식들이 성립한다.

$$P_2(a) = f(a), \quad P_2(b) = f(b), \quad P_2(c) = f(c) \tag{6.1.21}$$

즉, 항등식 $P_2(x) \equiv P(x)$가 성립한다. 따라서, 2차함수 $P_2(x)$가 점 x^*에서 최소값을 갖음을 알 수 있다.

예제 6.1.9 예제 6.1.4에서 다룬 Cauchy확률밀도함수의 모수 θ의 최우추정값을 구하는 문제를 다시 살펴보자. 다음과 같이 로그우도함수에 $[-1]$을 곱한 함수 $l_-(\theta)$를 정의해야 한다.

$$l_-(\theta) \doteq 3\ln(\pi) + \ln(1+[-5-\theta]^2) + \ln(1+[2-\theta]^2) + \ln(1+[4-\theta]^2) \tag{1}$$

이 목적함수의 극소값을 구하기 위해서, 다음 MATLAB프로그램 QFSM101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: QFSM101.m
% Quadratic Function Search Method for a local minimum
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
```

```
mLL = @(theta) 3*log(pi)+log(1+(5+theta).^2) ...
               +log(1+(2-theta).^2)+log(1+(4-theta).^2)
% ftn = @(x) x.^4 - 2*x.^2
tol = 10^(-7);
IterMax = 100;
a = 1;  b = 5; c = 2;   % a < c < b
x(1) = c; aa(1) = c; bb(1) = c;
fval(1) = mLL(x(1));
for ii=2:IterMax
    aa(ii) = a; bb(ii) = b;
    fa = mLL(a); fb = mLL(b); fc = mLL(c);
    D = - (a-b)*(b-c)*(c-a);
    A = (b-c)*fa+(c-a)*fb+(a-b)*fc; A = A/D;
    B = (b^2-c^2)*fa+(c^2-a^2)*fb+(a^2-b^2)*fc; B = -B/D;
    x(ii) = -B/(2*A);
    if x(ii) > c
        a = c; c = min([x(ii),b]); b = max([x(ii),b]);
    else
        b = c; c = max([x(ii),a]); a = min([x(ii),a]);
    end
    fval(ii) = mLL(x(ii));
    if abs(x(ii)-x(ii-1)) < tol
        break
    end
end
nn = 1:1:ii;
[ nn' x' fval' aa' bb' ]
% Plotting
plot(nn,x,'k-o',nn,aa,'r:',nn,bb,'b--','LineWidth',1.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('\it x','\it a','\it b','location','NE')
xlabel('\bf \it n','fontsize',12)
ylabel('\bf \it x_{n}','fontsize',12,'rotation',0)
saveas(gcf,'QFSM101','jpg')
save('QFSM101.txt','x','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB함수 QFSM101.m을 실행한 결과는 다음과 같다.

n	x(n)	f(x(n))	a	b
1.0000	2.0000	8.9557	2.0000	2.0000
2.0000	2.7182	8.9259	1.0000	5.0000
3.0000	2.4232	8.8750	2.0000	5.0000
4.0000	2.4001	8.8734	2.0000	2.7182
5.0000	2.3595	8.8720	2.0000	2.4232
6.0000	2.3531	8.8719	2.0000	2.4001
7.0000	2.3482	8.8719	2.0000	2.3595
8.0000	2.3471	8.8719	2.0000	2.3531
9.0000	2.3465	8.8719	2.0000	2.3482
10.0000	2.3463	8.8719	2.0000	2.3471
11.0000	2.3462	8.8719	2.0000	2.3465
12.0000	2.3462	8.8719	2.0000	2.3463

```
13.0000    2.3462    8.8719    2.0000    2.3462
14.0000    2.3462    8.8719    2.0000    2.3462
15.0000    2.3462    8.8719    2.0000    2.3462
16.0000    2.3462    8.8719    2.0000    2.3462
17.0000    2.3462    8.8719    2.0000    2.3462
18.0000    2.3462    8.8719    2.0000    2.3462
19.0000    2.3462    8.8719    2.0000    2.3462
```

이 결과물에서 알 수 있듯이, 19번째 반복단계에서 정지조건이 만족되어 알고리즘을 멈춘다. 이때 θ값은 2.3462이고 이에 해당하는 극소값은 8.8719이다. 이 MATLAB프로그램을 실행하면, 그림 6.1.7이 그려진다. 그림 6.1.7에서 볼 수 있듯이, 2차함수탐색법은 빨리 수렴한다. 그러나, 정지조건이 더 강해지면, 예를 들어 오차허용값 tol을 10^{-7}에서 10^{-9}로 바꾸면, 반복회수가 100회가 넘게 된다. 이 예제에서 알 수 있듯이, 2차함수탐색법은 정밀한 해를 구하는데 적합하지 않다. ■

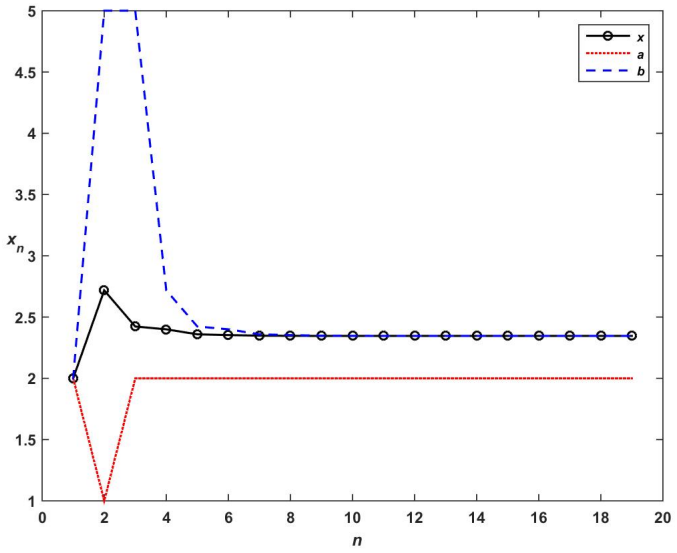


그림 6.1.7. 2차함수탐색법

예제 6.1.10 Python을 사용해서 예제 6.1.9을 다시 다루기 위해서, 다음 Python프로그램 QFSM101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
```

```
5
6  import numpy as np
7  import matplotlib.pyplot as plt
8
9  mLL = lambda theta: 3*np.log(np.pi)+np.log(1+(5+theta)**2)        \
10                 +np.log(1+(2-theta)**2)+np.log(1+(4-theta)**2);
11 tol = 10**(-7);
12 IterMax = 100;
13 a = 1;  b = 5; c = 2;   # a < c < b
14 x = [0.0]*IterMax; aa = [0.0]*IterMax; bb = [0.0]*IterMax;
15 fval = [0.0]*IterMax;
16 x[0] = c; aa[0] = c; bb[0] = c;
17 fval[0] = mLL(x[0]);
18 for ii in range(1,IterMax):
19     aa[ii] = a; bb[ii] = b;
20     fa = mLL(a); fb = mLL(b); fc = mLL(c);
21     D = - (a-b)*(b-c)*(c-a);
22     A = (b-c)*fa+(c-a)*fb+(a-b)*fc; A = A/D;
23     B = (b**2-c**2)*fa+(c**2-a**2)*fb+(a**2-b**2)*fc; B = -B/D;
24     x[ii] = -B/(2*A);
25     if x[ii] > c:
26         a = c; c = min([x[ii],b]); b = max([x[ii],b]);
27     else:
28         b = c; c = max([x[ii],a]); a = min([x[ii],a]);
29     fval[ii] = mLL(x[ii]);
30     if abs(x[ii]-x[ii-1]) < tol:
31         break
32 nn = np.arange(ii);
33 print(nn, x, fval, aa, bb)
34
35 # Plotting  the function
36
37 fig = plt.figure()
38 plt.plot(nn,x[0:ii],'k-o',lw=2,label='x')
39 plt.plot(nn,aa[0:ii],'r:',lw=2,label='a')
40 plt.plot(nn,bb[0:ii],'b--',lw=2,label='b')
41 plt.xlabel(r'$n$',fontsize=12)
42 plt.ylabel(r'$x(n)$',fontsize=12)
43 fig.savefig('QFSM101Py.png')
44
45 # End of Program
```

이 Python프로그램을 수행한 결과는 예제 6.1.9의 결과와 같다. ■

MATLAB함수 fminbnd.m에서는 느리지만 어느 경우라도 극소값을 구할 수 있는 황금섹션탐색법과 속도가 빠른 2차함수근사탐색법 (parabolic interpolation)을 함께 사용해서 단변수함수의 극소값을 계산한다. 만약 주어진 구간에서 함수 $f(x)$가 여러 극소값들을 갖으면, MATLAB함수 fminbnd.m은 극소값을 찾아가지 못할 수도 있다

예제 6.1.11 예제 6.1.4에서 다룬 Cauchy확률밀도함수의 모수 θ의 최우추정값을 구하는 문제를 다시 살펴보자. 이러한 목적을 달성하기 위해서는 다음과 같이 로그우도함수에 $[-1]$

을 곱한 함수 $l_{-}(\theta)$을 만들어야 한다.

$$l_{-}(\theta) = 3\ln(\pi) + \ln(1 + [-5 - \theta]^2) + \ln(1 + [2 - \theta]^2) + \ln(1 + [4 - \theta]^2) \qquad (1)$$

여기서 로그우도함수에 을 곱하는 이유는 MATLAB함수 fminbnd.m이 극대값을 구하는 것이 아니라 극소값을 구하기 때문이다. 이 목적함수의 극소값을 구하기 위해서, 다음 MATLAB 프로그램 GoldenSectionParabolic101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename GoldenSectionParabolic101.m
% Golden Section and Parabolic Optimization
% Programmed by CBS
% ------------------------------------------------------------------
function GoldenSectionParabolic101
options = optimset('TolX',1.0e-8,'Display','iter', ...
                   'MaxFunEvals',20,'MaxIter',99);
% First Interval
[theta, fval, EF, GoldenSectionParabolic101a] ...
      = fminbnd(@CauchyLogLikelihood,-10,0,options)
% Second Interval
[theta, fval, EF, GoldenSectionParabolic101b] ...
      = fminbnd(@CauchyLogLikelihood,0,10,options)
end
% ------------------------------------------------------------------------
function mLL = CauchyLogLikelihood(theta)
%  (-1)*loglikelihood function
mLL = 3*log(pi)+log(1+(5+theta)^2) ...
      +log(1+(2-theta)^2)+log(1+(4-theta)^2);
end
% ------------------------------------------------------------------------
```

첫 번째 MATLAB함수 fminbnd의 첫 번째 변수 @CauchyLoglikelihood는 목적함수이고, 두 번째 변수와 세 번째 변수는 탐색구간 $(a, b) = (-10, 0)$를 지정하고, 네 번째 변수는 MATLAB함수 optimoptions.m을 사용해서 최적화구조를 지정하는 것이다. 여기서 유의할 점은 탐색구간의 양 끝점들 a와 b는 탐색구간에 포함하지 않는다는 것이다. 또한, MATLAB 함수 optimoptions.m에서 옵션 TolX에는 오차허용값(termination tolerance) 10^{-8}을 지정하였고, 옵션 Display에는 각 단계에서 결과물을 출력하게 하는 'iter'을 지정하였다. 첫 번째 MATLAB함수 fminbnd.m를 실행한 결과는 다음과 같다.

```
Func-count     x          f(x)         Procedure
    1       -6.18034      13.1756       initial
    2       -3.81966      11.9879       golden
    3       -2.36068      12.2306       golden
    4       -3.56466      12.0816       parabolic
    5       -4.72136      11.6861       golden
```

```
     6       -5.27864      11.9645        golden
     7       -4.56316      11.7032        parabolic
     8       -4.70609      11.6862        parabolic
     9       -4.72194      11.6861        parabolic
    10       -4.72114      11.6861        parabolic
    11       -4.72114      11.6861        parabolic
    12       -4.72114      11.6861        parabolic
    13       -4.72114      11.6861        parabolic

최적화가 종료됨:
 현재 x가 1.000000e-08의 OPTIONS.TolX로 종료 조건을 충족시킵니다.

theta = -4.7211
fval = 11.6861
EF = 1

GoldenSectionParabolic101a =
    iterations: 12
     funcCount: 13
     algorithm: 'golden section search, parabolic interpolation'
       message: '최적화가 종료됨:
 현재 x가 1.000000e-08의 OPTIONS.TolX로 종료 조건을 충족시킵니다.
```

이 결과물에서 알 수 있듯이, 12번째 반복 후 정지조건이 만족되어 알고리즘을 멈춘다. 이때 θ값은 -4.7211이고 이에 해당하는 극대값은 11.6861이다. 이 결과는 예제 6.1.4의 결과와 같다.

두 번째 MATLAB함수 fminbnd의 첫 번째 변수에는 같은 목적함수 @CauchyLoglikelihood, 두 번째 변수와 세 번째 변수는 탐색구간 $(a, b) = (0, 10)$를 지정하였다. 이 두 번째 MATLAB함수 fminbnd를 실행한 결과는 다음과 같다.

```
 Func-count     x          f(x)         Procedure
    1        3.81966      9.29414        initial
    2        6.18034      12.9368        golden
    3        2.36068      8.87199        golden
    4        2.64939      8.91045        parabolic
    5        2.00583      8.95265        parabolic
    6        2.38614      8.87274        parabolic
    7        2.33615      8.87193        parabolic
```

```
     8        2.34615      8.87188        parabolic
     9        2.34623      8.87188        parabolic
    10        2.34617      8.87188        parabolic
    11        2.34617      8.87188        parabolic
    12        2.34617      8.87188        parabolic
    13        2.34617      8.87188        parabolic
    14        2.34617      8.87188        golden

최적화가 종료됨:
 현재 x가 1.000000e-08의 OPTIONS.TolX로 종료 조건을 충족시킵니다.


theta = 2.3462
fval = 8.8719
EF = 1


GoldenSectionParabolic101b =
    iterations: 13
     funcCount: 14
     algorithm: 'golden section search, parabolic interpolation'
       message: '최적화가 종료됨:
 현재 x가 1.000000e-08의 OPTIONS.TolX로 종료 조건을 충족시킵니다.
```

이 결과물에서 알 수 있듯이, 13번째 반복단계 후 정지조건이 만족되어 알고리즘을 멈춘다. 이때 θ값은 2.3462이고 이에 해당하는 극대값은 8.87188이다. ■

예제 6.1.12 Python을 사용해서 예제 6.1.11을 다시 다루기 위해서, 다음 Python프로그램 GoldenSectionParabolic101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import scipy.optimize

banana = lambda x: 100*(x[1]-x[0]**2)**2+(1-x[0])**2;
xopt = scipy.optimize.fmin(func=banana, x0=[-1.2,1]);
fval = banana(xopt);
print('xopt = ', xopt, 'fval = ', fval)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 6.1.11의 결과와 같다. ■

제6.2절 다변수함수의 최적화이론

이 절에서는 벡터 $\boldsymbol{x} = [x_1, x_2, \cdots, x_m]^t$를 설명변수로 하는 m 변수함수 $f(\boldsymbol{x}) = f(x_1, x_2, \cdots, x_m)$의 극대값과 극소값을 구하는 문제를 살펴보자. 여기서 f는 C^2 급 함수라고 가정하자. 함수 f의 그래디언트벡터(gradient vector)와 Hessian행렬을 각각 다음과 같이 정의하자.

$$\nabla f(\boldsymbol{x}) \doteq \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \frac{\partial f(\boldsymbol{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x_m} \end{bmatrix}, \quad H(\boldsymbol{x}) \doteq \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_m} \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_m \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_m \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_m \partial x_m} \end{bmatrix} \tag{6.2.1}$$

Taylor전개를 바탕으로 함수 $f(\boldsymbol{x})$의 극대값과 극소값을 구할 수 있다. 다음 정리들이 성립한다. 이 정리들에 대한 자세한 내용은 **IM&F9** [4]의 제2장을 참조하라.

정리 6.2.1: 다변수함수의 극소값

함수 $y = f(\boldsymbol{x}) = f(x_1, x_2, \cdots, x_n)$가 개집합 $V(\subset R^n)$에서 연속인 1차편도함수와 2차편도함수를 갖는다고 가정하자. 함수 $f(\boldsymbol{x})$가 점 $\boldsymbol{x}^*$에서 극소값을 갖기 위한 필요충분조건은 다음과 같다.

(a) 1차조건 : $\nabla f(\boldsymbol{x}^*) = 0$

(b) 2차조건 : Hessian행렬 $H(\boldsymbol{x}^*)$는 양반정치(positive-semidefinite)이다.

정리 6.2.2: 다변수함수의 극대값

함수 $y = f(\boldsymbol{x}) = f(x_1, x_2, \cdots, x_n)$가 개집합(open set) $V(\subset R^n)$에서 연속인 1차편도함수와 2차편도함수를 갖는다고 가정하자. 함수 $f(\boldsymbol{x})$가 점 $\boldsymbol{x}^*$에서 극대값을 갖기 위한 필요충분조건은 다음과 같다.

(a) 1차조건 : $\nabla f(\boldsymbol{x}^*) = 0$

(a) 2차조건 : Hessian행렬 $H(\boldsymbol{x}^*)$는 음반정치(negative-semidefinite)이다.

예제 6.2.1 다음 함수를 살펴보자.

$$f(x,y) = \frac{[x-1]^2+4}{2y} + \ln y \tag{1}$$

이 함수의 편도함수들은 다음과 같다.

$$\frac{\partial f}{\partial x} = \frac{x-1}{y}, \qquad \frac{\partial f}{\partial y} = -\frac{[x-1]^2+4}{2y^2} + \frac{1}{y} \tag{2}$$

$$\frac{\partial^2 f}{\partial x^2} = \frac{1}{y}, \qquad \frac{\partial^2 f}{\partial y^2} = \frac{[x-1]^2+4}{y^3} - \frac{1}{y^2} \tag{3}$$

$$\frac{\partial^2 f}{\partial x \partial y} = -\frac{x-1}{y^2} \tag{4}$$

따라서, 그래디언트벡터와 Hessian행렬은 각각 다음과 같다.

$$\nabla f = \begin{bmatrix} \frac{x-1}{y} \\ -\frac{[x-1]^2+4}{2y^2} + \frac{1}{y} \end{bmatrix}, \quad H = \begin{bmatrix} \frac{1}{y} & -\frac{x-1}{y^2} \\ -\frac{x-1}{y^2} & \frac{[x-1]^2+4}{y^3} - \frac{1}{y^2} \end{bmatrix} \tag{5}$$

최적화의 1차조건 $\nabla f(\boldsymbol{x}^*) = \mathbf{0}$을 만족하는 벡터는 $\boldsymbol{x}^* = [1,\, 2]$이다. 이 점에서 Hessian 행렬은 다음과 같다.

$$H(1,2) = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{4} \end{bmatrix} \tag{6}$$

이 Hessian행렬 $H(1,2)$는 양정치행렬이므로, 정리 6.2.1에서 알 수 있듯이 함수 f는 점 $\boldsymbol{x}^* = [1,\, 2]$에서 극소값을 갖는다. 또한, 점 $\boldsymbol{x}^* = [1,\, 2]$에서 2차 근사식은 다음과 같다.

$$f(x,y) = 2 + \ln 2 + \frac{1}{4}[x-1]^2 + \frac{1}{8}[y-2]^2 \tag{7}$$

■

다음 예제에서 알 수 있듯이, MATLAB의 Symbolic Math Toolbox의 함수들을 사용해서 그래디언트벡터와 Hessian행렬을 구할 수 있다.

예제 6.2.2 MATLAB의 Symbolic Math Toolbox의 함수들을 사용해서 2변수함수의 극소값을 구하기 위해서, 다음 MATLAB프로그램 MultiVariOptimi101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename MultiVariOptimi101.m
% Optimization of the Rosenbrock banana function
% ------------------------------------------------------------------------
syms x y
f = 100*(y-x^2)^2 + (1-x)^2
% Partial derivatives
fx = diff(f,x), fy = diff(f,y)
fxx = diff(fx,x), fyy = diff(fy,y), fxy = diff(fx,y)
sol = solve(fx,fy)
a = sol.x, b = sol.y
fab = subs(subs(f,x,a),y,b)
grad(1) = subs(subs(fx,x,a),y,b);
grad(2) = subs(subs(fy,x,a),y,b);
H(1,1) = subs(subs(fxx,x,a),y,b);
H(2,2) = subs(subs(fyy,x,a),y,b);
H(1,2) = subs(subs(fxy,x,a),y,b);
H(2,1) = H(1,2);
grad, H
det(H)
[evec eval] = eig(H)
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 MultiVariOptimi101.m에서는 다음과 같은 Rosenbrock banana 함수의 극소값을 구하고자 한다.

$$f = 100[y - x^2]^2 + [1 - x]^2 \tag{1}$$

이 MATLAB프로그램을 실행하면, 편도함수들이 다음과 같음을 알 수 있다.

$$f_x = 2x - 400x[y - x^2] - 2, \qquad f_y = 200[y - x^2] \tag{2}$$

$$f_{xx} = 1200x^2 - 400y + 2, \qquad f_{yy} = 200, \qquad f_{xy} = -400x \tag{3}$$

식 (2)에서 알 수 있듯이, 점 $[1,1]$에서 1차조건 $f_x = 0$와 $f_y = 0$를 만족한다. 점 $[1,1]$에서 다음 식들이 성립함을 알 수 있다.

$$f(1,1) = 0, \quad \nabla = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad H = \begin{bmatrix} 802 & -400 \\ -400 & 200 \end{bmatrix} \tag{4}$$

식 (4)에서 알 수 있듯이, 다음 식들이 성립한다.

$$f_{xx}(1,1) > 0, \quad \det(H) > 0 \tag{5}$$

따라서, Hessian행렬 H는 양정치이다. 즉, 함수 f는 점 $[1,1]$에서 극소값 0을 갖는다. Hessian 행렬 H의 가장 작은 고유값이 0.39936이므로, H가 양정치행렬임을 확인할 수 있다. ■

예제 6.2.3 Python을 사용해서 예제 6.2.2을 다시 다루기 위해서, 다음 Python프로그램 MultiVariOptimi101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import scipy.optimize

banana = lambda x: 100*(x[1]-x[0]**2)**2+(1-x[0])**2;
xopt = scipy.optimize.fmin(func=banana, x0=[-1.2,1]);
fval = banana(xopt);
print('xopt = ', xopt, 'fval = ', fval)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 6.2.2의 결과와 같다. ■

예제 6.2.4 MATLAB함수 fminsearch.m을 사용해서 예제 6.2.2의 문제를 풀기 위해서, 다음 MATLAB프로그램 MultiVariOptimi102.m을 실행해 보자.

```
% --------------------------------------------------------------------
% Filename MultiVariOptimi102.m
% Optimization of the Rosenbrock banana function
% Using fminsearch.m
% --------------------------------------------------------------------
function [x fvalue] = MultiVariOptimi102
f = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2
x0 = [-1.2 1]          % Initial vector
[x fvalue] = fminsearch(f,x0)
save('MultiVariOptimi102.txt','x','fvalue','-ascii')
% End of program
% --------------------------------------------------------------------
```

이 MATLAB프로그램에서도 Rosenbrock banana함수의 극소값을 구하고자 한다. 이 MATLAB프로그램에서는 초기벡터로 $\boldsymbol{x}_0 = [-1.2,\ 1]$을 사용하였다. 그 결과 함수 f가 점 $[1,1]$에서 극소값 8.2×10^{-10}을 갖음을 확인할 수 있다. ■

예제 6.2.5 Python을 사용해서 예제 6.2.4을 다시 다루기 위해서, 다음 Python프로그램 MultiVariOptimi102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import scipy.optimize

banana = lambda x: 100*(x[1]-x[0]**2)**2+(1-x[0])**2;
xopt = scipy.optimize.fmin(func=banana, x0=[-1.2,1]);
fval = banana(xopt);
print('xopt = ', xopt, 'fval = ', fval)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 6.2.4의 결과와 같다. ■

제6.3절 다변수함수의 방향탐색법

이 절에서는 반복법을 적용해서 다변수함수 $f(\boldsymbol{x})$의 극소값을 구하는 방법을 살펴보자. 정리 6.2.1에서 알 수 있듯이, $\boldsymbol{x}^*$가 극소점이 되기 위한 필요조건은 다음과 같다.

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0} \tag{6.3.1}$$

함수 $f(\boldsymbol{x})$가 비선형인 경우에는 방정식 (6.3.1)의 닫힌해를 구하는 것이 어려운 경우가 많다. 이 경우에는 반복법을 사용해서 극소점 $\boldsymbol{x}^*$로 수렴하는 점열 $\{\boldsymbol{x}^{(n)}\}$을 구한다. 여기서 $\boldsymbol{x}^{(n)}$은 제n번째 단계에서 설명변수를 나타내는 벡터이다.

방향탐색으로 극소점을 탐색하는 방향탐색법(directional search method)을 살펴보기 위해서, 다음 MATLAB프로그램 DirectionalSearchGraph101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename DirectionalSearch101.m
% Directional Search Method Graph 1
% Programmed by CBS
% ------------------------------------------------------------------------
close all; clear all; clc
% Function and its Gradient vector and Hessian matrix
f = @(x1,x2) 2.0*x1^2 + x1*x2 + x2^2 - 5*x1 -3*x2;
grad = @(x) [ 4*x(1)+x(2)-5, x(1)+2.0*x(2)-3 ]
H = [ 4, 1; 1, 2 ];
% Function for Arrow
drawArrow = @(x,y,varargin) ...
              quiver( x(1),y(1),x(2)-x(1),y(2)-y(1),0, varargin{:})
% Design variables at mesh points
ii = -4.0:0.1:4.0;
jj = -4.0:0.1:4.0;
[x1Mesh, x2Mesh] = meshgrid(ii, jj);
```

```
fMesh = 2.0*x1Mesh.^2+x1Mesh.*x2Mesh+x2Mesh.^2-5*x1Mesh-3*x2Mesh;
% Create a contour plot
fig = figure();
hold on
lines = -2:4:40;  % Specify contour lines
CS = contour(x1Mesh, x2Mesh, fMesh,lines);
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('x1'), ylabel('x2','rotation',0)
xStar = [0, -2]
gn = grad(xStar), gn = 2*gn/norm(gn)
Gdum11 = [xStar(1),xStar(1)+gn(1)]
Gdum12 = [xStar(2),xStar(2)+gn(2)]
drawArrow(Gdum11,Gdum12,'linewidth',1.2,'color','k')
plot([xStar(1), xStar(1)-gn(1)], ...
     [xStar(2), xStar(2)-gn(2)],'b--','linewidth',1.2)
tn = [gn(2), -gn(1)], tn = 2.0*tn/norm(tn)
plot([xStar(1)-tn(1),xStar(1)+tn(1)], ...
     [xStar(2)-tn(2),xStar(2)+tn(2)],'r-','linewidth',1.2)
Gdum21 = [xStar(1),xStar(1)-gn(1)]
Gdum22 = [xStar(2),xStar(2)-0.5*gn(2)]
drawArrow(Gdum21,Gdum22,'linewidth',1.2,'color','b')
axis('equal')
text(1.5,-1.2,'d (\bf{x}_n)')
text(-2.0,-3.7,' \nabla f (\bf{x}_n)')
hold off
saveas(gcf,'DirectionalSearchGraph101','jpg')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 DirectionalSearchGraph101.m을 실행하면, 그림 6.3.1을 출력한다. 그림 6.3.1을 바탕으로 다음 식을 만족하는 벡터 $\boldsymbol{d}(\boldsymbol{x}^{(n)})$을 생각해보자.

$$\nabla f(\boldsymbol{x}^{(n)})^t \boldsymbol{d}(\boldsymbol{x}^{(n)}) < 0 \tag{6.3.2}$$

즉, 점 $\boldsymbol{x}^{(n)}$에서 함수 $f(\boldsymbol{x})$의 최대상승방향을 나타내는 그래디언트벡터 $\nabla f(\boldsymbol{x}^*)$와 벡터 $\boldsymbol{d}(\boldsymbol{x}^{(n)})$는 둔각을 이루고, 벡터 $\boldsymbol{d}(\boldsymbol{x}^{(n)})$는 함수 $f(\boldsymbol{x})$의 하강방향을 나타낸다. 따라서, 다음 점화식을 만족하는 벡터열 $\{\boldsymbol{x}^{(n)}\}$은 극소점 $\boldsymbol{x}^*$로 수렴할 것으로 기대된다.

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \lambda^{(n)} \boldsymbol{d}(\boldsymbol{x}^{(n)}), \quad (n = 1, 2, \cdots) \tag{6.3.3}$$

여기서 $\boldsymbol{d}(\boldsymbol{x}^{(n)})$은 탐색방향을 나타내는 방향벡터(directional vector), 양수 $\lambda^{(n)}$은 함수 $f(\boldsymbol{x}^{(n)} + \lambda^{(n)} \boldsymbol{d}(\boldsymbol{x}^{(n)}))$이 작은 값을 갖도록 하는 스텝크기(step size)이다. 식 (6.3.3)을 사용하는 최적화법을 방향탐색법(directional search method)이라 부른다. 이 $\boldsymbol{d}(\boldsymbol{x}^{(n)})$과 $\lambda^{(n)}$을 선택하는 방법에 따라 다양한 방향탐색법이 구성된다. 가장 단순한(naive) 방향탐색법은 급하강법(steepest descent method)이다. 어떤 문제에서는 급하강법이 매우 비효율적이다. 여러 방향을 시도해보고 그 중 좋은 방향을 선택하는 방법으로는 공액방향법(conjugate

directional method)이 있다. 그리고, Newton-Raphson법이나 준Newton법은 일반적으로 수렴속도가 빠른 방향탐색법들이다.

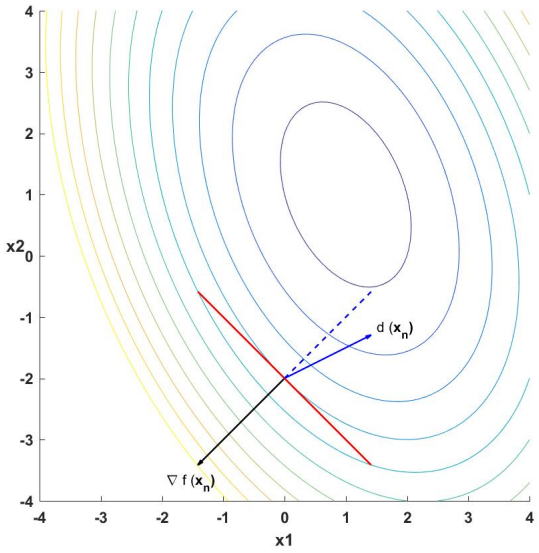


그림 6.3.1. 방향탐색법

예제 6.3.1 Python을 사용해서 그림 6.3.1을 그리기 위해서는 다음 Python프로그램 DirectionalSearchGraph101.Py를 실행하라.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

#  Function and its Gradient vector and Hessian matrix
def f(x1,x2):
    return 2.0*x1**2 + x1*x2 + x2**2 - 5*x1 -3*x2;
def grad(x1,x2):
    return 4*x1+x2-5, x1+2.0*x2-3;
H = np.matrix('4,1; 1, 2');

# Design variables at mesh points
ii = np.linspace(-4.0, 4.0, 81);
jj = np.linspace(-4.0, 4.0, 81);
X1, X2 = np.meshgrid(ii,jj);
Z = 2.0*X1**2 + X1*X2 + X2**2 - 5*X1 -3*X2;

# Plotting
fig = plt.figure()
levels = np.arange(-2,40,4);
CS = plt.contour(X1, X2, Z, levels);
plt.xlabel('x1',fontsize=12)
plt.ylabel('x2',fontsize=12)
# Normal vector, tangent vector
```

```
xStar = [0, -2]
gn = grad(0,-2);
gn = gn/np.linalg.norm(gn)
ax1 = plt.axes()
ax1.arrow(xStar[0], xStar[1], 2.0*gn[0], 2.0*gn[1], \
          head_width=0.05, head_length=0.1, fc='k', ec='k')
plt.plot([xStar[0], xStar[0]-2.0*gn[0]], [xStar[1], xStar[1]-2.0*gn[1]], \
     'b--',lw=1.2)
tn = [ gn[1], -gn[0] ];
tn = tn/np.linalg.norm(tn)
plt.plot([xStar[0]-2.0*tn[0],xStar[0]+2.0*tn[0]], \
     [xStar[1]-2.0*tn[1],xStar[1]+2.0*tn[1]],'r-',lw=1.2)
ax2 = plt.axes()
ax2.arrow(xStar[0], xStar[1], -2.0*gn[0], -0.5*2.0*gn[1], \
          head_width=0.05, head_length=0.1, fc='k', ec='k')
plt.axis('equal')
plt.text(1.5,-1.2,r'$d (\bf{x}_n)$')
plt.text(-2.0,-3.7,r'$\nabla f (\bf{x}_n)$')
plt.show()
fig.savefig('DirectionalSearchGraph101Py.png')

# End of program
```

■

방향탐색법을 요약하면, 다음과 같다.

알고리즘 6.3.1: 방향탐색법

(1단계) 제n단계에서 선택한 점 $\boldsymbol{x}^{(n)}$에서 위에 기술한 방향탐색법을 적용해서 적당한 방향벡터 $\boldsymbol{d}(\boldsymbol{x}^{(n)})$을 선택한다.

(2단계) 제1단계에서 선택한 방향의 선분 상에서 목적함수값을 최소화하는 수 $\lambda^{(n)}$을 찾고, $\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \lambda^{(n)}\boldsymbol{d}(\boldsymbol{x}^{(n)})$을 계산한다. 이 단계를 직선탐색(line search)이라 한다.

(3단계) 만약 멈춤규칙(stopping rule)이 만족되면, 이 알고리즘의 수행을 멈춘다. 그렇지 않으면, n을 $n+1$로 증가시킨 다음 제2단계로 넘어간다.

6.3.1 급하강법

등산을 할 때 산 정상으로 가는 가장 빠른 지름길은 경사가 가장 급한 방향으로 오르는 것이다. 이 직감적인 방법을 급상승법(steepest ascent method)이라 한다. 극소점 탐색에서는 이에 해당하는 급하강법(steepest descent method)을 사용한다. 또한, 이들을 합해서 급경사법

(steepest method) 이라 부른다. 우리의 목적은 극소점을 찾는 것이므로 이 소절에서는 급하강법에 대해서 살펴보자.

급하강법에서는 다음과 같은 방향벡터 $\boldsymbol{d}(\boldsymbol{x}^{(n)})$ 을 사용한다.

$$\boldsymbol{d}(\boldsymbol{x}^{(n)}) = -\nabla f(\boldsymbol{x}^*) \tag{6.3.4}$$

다음 식들이 성립한다.

$$\nabla f(\boldsymbol{x}^{(n)})^t \boldsymbol{d}(\boldsymbol{x}^{(n)}) = -\|\nabla f(\boldsymbol{x}^{(n)})\|^2 < 0 \tag{6.3.5}$$

점 $\boldsymbol{x}^{(n)}$ 에서 함수 $f(\boldsymbol{x})$ 는 가장 급격하게 하강한다. 그러나, 점 $\boldsymbol{x}^{(n)}$ 에서 점 $\boldsymbol{x}^{(n)} + \boldsymbol{d}(\boldsymbol{x}^{(n)})$ 으로 이동하면서 함수 $f(\boldsymbol{x})$ 가 처음에는 감소하다가 다시 증가할 수도 있다. 따라서, 다음 식을 만족하는 $\lambda^{(n)}(>0)$ 을 사용한다.

$$\lambda^{(n)} \doteq \arg\min_{\lambda} f\left(\boldsymbol{x}^{(n)} - \lambda \nabla f(\boldsymbol{x}^{(n)})\right) \tag{6.3.6}$$

앞에서 언급했듯이, 식 (6.3.6) 을 만족하는 $\lambda^{(n)}$ 을 구하는 것을 직선탐색 (line search) 이라고 한다. 급하강법을 요약하면, 다음과 같다.

알고리즘 6.3.2: 급하강법

(1단계) 초기벡터 $\boldsymbol{x}^{(1)}$ 에서 함수값 $f(\boldsymbol{x}^{(1)})$ 를 구하고, $n = 1$ 이라 한다.

(2단계) 식 $\lambda^{(n)} \doteq \arg\min_{\lambda} f\left(\boldsymbol{x}^{(n)} - \lambda \nabla f(\boldsymbol{x}^{(n)})\right)$ 을 만족하는 양수 $\lambda^{(n)}$ 을 구한다.

(3단계) 다음 벡터를 계산한다.

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} - \lambda^{(n)} \nabla f(\boldsymbol{x}^{(n)})$$

(4단계) 만약 주어진 상수 ϵ 에 대해서 멈춤규칙 $\|\boldsymbol{x}^{(n+1)} - \boldsymbol{x}^{(n)}\| < \epsilon_1$ 이나 멈춤규칙 $\left|f(\boldsymbol{x}^{(n)})\right| < \epsilon_2$ 가 만족되면, 알고리즘의 수행을 멈춘다. 그렇지 않으면, n 을 $n+1$ 로 증가시킨 다음 제2단계로 넘어간다.

예제 6.3.2 급하강법을 사용해서, 다음 목적함수의 극소값을 구해보자.

$$f(\boldsymbol{x}) = 2x_1^2 + x_1x_2 + x_2^2 - 5x_1 - 3x_2 \tag{1}$$

이 함수 $f(\boldsymbol{x})$의 그래디언트벡터는 다음과 같다.

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} 4x_1 + x_2 - 5 \\ x_1 + 2x_2 - 3 \end{bmatrix} \tag{2}$$

초기점을 $\boldsymbol{x}^{(0)} = [1, 2]^t$라고 하면, 다음 식이 성립한다.

$$\nabla f(\boldsymbol{x}^{(0)}) = [1, 2]^t \tag{3}$$

다음 식이 성립한다.

$$\boldsymbol{x}^{(0)} - \lambda \nabla f(\boldsymbol{x}^{(0)}) = [1 - \lambda, 2 - 2\lambda]^t \tag{4}$$

직선탐색은 다음 식을 만족하는 $\lambda^{(0)}$를 구하는 것이다.

$$\lambda^{(0)} = \arg\min_{\lambda} f\left(1 - \lambda, 2 - 2\lambda\right) \tag{5}$$

이 $\lambda^{(0)}$는 다음 식들을 만족한다.

$$\frac{d}{d\lambda} f\left(1 - \lambda, 2 - 2\lambda\right)\Big|_{\lambda=\lambda^{(0)}} = 16\lambda^{(0)} - 5 = 0 \tag{6}$$

즉, $\lambda^{(0)} = 5/16$이다. 따라서, 제1단계에서 설명변수벡터는 다음과 같다.

$$\boldsymbol{x}^{(1)} = \begin{bmatrix} 1 - \lambda^{(0)} \\ 2 - 2\lambda^{(0)} \end{bmatrix} = \begin{bmatrix} \frac{11}{16} \\ \frac{11}{8} \end{bmatrix} \tag{7}$$

같은 방법으로 다음 식들을 구할 수 있다.

$$\nabla f(\boldsymbol{x}^{(1)}) = \left[-\frac{7}{8}, \frac{7}{16}\right]^t \tag{8}$$

$$\boldsymbol{x}^{(1)} - \lambda \nabla f(\boldsymbol{x}^{(1)}) = \left[\frac{11 + 14\lambda}{16}, \frac{22 - 7\lambda}{16}\right]^t \tag{9}$$

직선탐색은 다음 식을 만족하는 $\lambda^{(1)}$를 구하는 것이다.

$$\lambda^{(1)} = \arg\min_{\lambda} f\left(\frac{11+14\lambda}{16}, \frac{22-7\lambda}{16}\right) \tag{10}$$

이 $\lambda^{(1)}$은 다음 식들을 만족한다.

$$\frac{d}{d\lambda} f\left(\frac{11+14\lambda}{16}, \frac{22-7\lambda}{16}\right)\Big|_{\lambda=\lambda^{(1)}} = \frac{686\lambda^{(1)} - 189}{256} = 0 \tag{11}$$

즉, $\lambda^{(1)} = 189/686 \approx 0.276$이다. 따라서, 제2단계에서 설명변수벡터는 다음과 같다.

$$\boldsymbol{x}^{(2)} = \begin{bmatrix} \frac{11+14\lambda^{(1)}}{16} \\ \frac{22-7\lambda^{(1)}}{16} \end{bmatrix} = \begin{bmatrix} \frac{13}{14} \\ \frac{281}{224} \end{bmatrix} \approx \begin{bmatrix} 0.9286 \\ 1.2545 \end{bmatrix} \tag{12}$$

■

알고리즘 6.3.2의 제2단계를 계산하기 위해서는 많은 계산을 필요로 한다. 따라서, 이 단계를 쉽게 피해가는 방법들이 제시되었다. 대표적인 Armijo규칙을 적용하면, 제2단계를 다음과 같이 쓸 수 있다.

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} - \theta^{(n)} \nabla f(\boldsymbol{x}^{(n)}) \tag{6.3.7}$$

여기서 $\theta^{(n)}(>0)$은 어떤 상수 β의 지수함수로서 다음 부등식을 만족하는 가장 큰 값이다.

$$f\left(\boldsymbol{x}^{(n+1)}\right) \le f\left(\boldsymbol{x}^{(n)}\right) - \alpha \|\nabla f(\boldsymbol{x}^{(n)})\|^2 \theta^{(n)} \tag{6.3.8}$$

여기서 $\alpha(>0)$는 주어진 상수이다. Armijo규칙에 대해서는 Armijo [10]를 참조하라.

급경사법은 제n번째 반복에서 $\boldsymbol{x}^{(n)}$을 바탕으로 가장 유효한 방향으로 다음 벡터 $\boldsymbol{x}^{(n+1)}$을 선택한다. 따라서, 전체적으로 가장 좋은 결과를 부여한다고 생각하기 쉽다. 그러나, 이러한 생각은 잘못된 것이다. 왜냐하면, $\boldsymbol{x}^{(n)}$에서는 $\nabla f(\boldsymbol{x}^{(n)})$이 가장 좋은 방향이지만, 그 방향으로 조금 진행한 $\boldsymbol{x}^{(n)} + \boldsymbol{\epsilon}$에서 $\nabla f(\boldsymbol{x}^{(n)} + \boldsymbol{\epsilon})$이 반드시 최선의 방향이라고 할 수는 없기 때문이다.

예제 6.3.3 급하강법을 사용해서, 다음 Rosenbrock banana함수의 극소값을 구해보자.

$$f = 100[y - x^2]^2 + [1 - x]^2 \tag{1}$$

이 목적함수의 극소값을 구하기 위해서, 다음 MATLAB프로그램 SteepestDescentMethod101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename SteepestDescentMethod101.m
% Steepest Descent Method
% Using Armijo's stepsize rule
% Programmed by CBS
% ------------------------------------------------------------------------
function SteepestDescentMethod101
close all; clear all; clc; format long
% Inputs
xinit = [0,0 ];          % Initial vector
x = xinit';
% Armijo stepsize rule parameters
  alpha = .1;
  beta = .5;
  fx = fun(x);
  gx = grad(x);
  Iter = 0;              % Iteration Number
  nf = 1;          % No of funtion eval
% Iterations
while  norm(gx) > 1e-7
    theta = 1;
    newfx = fun(x - theta*gx);
    nf = nf+1;
    while fx - newfx < alpha*norm(gx)^2*theta
      theta = theta*beta;
      newfx = fun(x - theta*gx);
      nf = nf+1;
    end
    if (mod(Iter,1000)==1)
       fprintf('%5.0f %8.0f %15.8e \n',Iter,nf,fx);
    end
    x = x - theta*gx;
    fx = newfx;
    gx = grad(x);
    Iter = Iter + 1;
  end
% Output
  Iter, x, fun(x)
save('SteepestDescentMethod101.txt','Iter','x','-ascii')
end
% ------------------------------------------------------------------------
function y = fun(x)
% Rosenbrock funtion
y = 100*(x(1)^2 - x(2))^2 + (x(1)-1)^2;
end
function y = grad(x)
y = zeros(2,1);          % Column vector
y(1) = 100*(2*(x(1)^2-x(2))*2*x(1)) + 2*(x(1)-1);
y(2) = 100*(-2*(x(1)^2-x(2)));
end
% End of progrm
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 SteepestDescentMethod101.m을 실행한 결과의 일부는 다음과 같다.

```
      Iter      nf         fx
      11001   109129  1.18120111e-10
```

```
12001   119089  2.33391336e-11
13001   129050  4.60535018e-12
14001   139010  9.10396955e-13
15001   148971  1.79577240e-13
16001   158932  3.54606785e-14
17001   168892  7.00294321e-15

Iter = 17018

x =
   0.999999917484196
   0.999999834784009

ans = 6.812257822900501e-15
```

이 결과물에서 알 수 있듯이, 제17018번째 반복 후 정지조건이 만족되어 알고리즘을 멈춘다. 그때까지 함수값을 168,892번 계산한다. 또한, 함수 f는 $\boldsymbol{x} = [1.00000, 1.0000]^t$에서 극소값 6.8×10^{-15}을 갖는다. ■

예제 6.3.4 Python을 사용해서 예제 6.3.3을 다시 다루기 위해서, 다음 Python프로그램 SteepestDescentMethod101.Py를 실행해 보자.

```
"""
Created on Thu Jan 12 20:13:46 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt

# Rosenbrock funtion and its gradient
fun = lambda x: 100*(x[0]**2 - x[1])**2 + (x[0]-1)**2;
grad = lambda x: [ 100*(2*(x[0]**2-x[1])*2*x[0]) + 2*(x[0]-1), \
                   100*(-2*(x[0]**2-x[1])) ]

# Inputs
xinit = [0,0];          # Initial vector
x = [0, 0];
# Armijo stepsize rule parameters
alpha = 0.1;
beta = 0.5;
fx = fun(x);
gx = grad(x);
Iter = 0;               # Iteration Number
nf = 1;          # No of funtion eval

# Iterations
```

```
while  np.linalg.norm(gx) > 1e-7:
    theta = 1;
    dum1 = x - np.multiply(gx,theta)
    newfx = fun(dum1);
    nf = nf+1;
    while fx - newfx < alpha*np.linalg.norm(gx)**2*theta:
      theta = theta*beta;
      dum2 = x - theta*np.array(gx)
      newfx = fun(dum2);
      nf = nf+1;
    if (np.mod(Iter,1000)==1):
       print(Iter,nf,fx);
    x = x - np.multiply(gx,theta);
    fx = newfx;
    gx = grad(x);
    Iter = Iter + 1;
print('Iter =', Iter, 'x =', x, 'f(x) = ', fun(x))

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 6.3.3의 결과와 같다. ■

Venkataraman [63, 제6.3.1소절]에는 급하강법이 자세히 소개되어 있다. 또한, 이 방법을 구현하기 위한 MATLAB프로그램 SteepestDescent.m이 제공되어 있다. 다음 예제는 이 MATLAB프로그램을 사용하는 예이다.

예제 6.3.5 MATLAB프로그램 SteepestDescent.m을 사용하기 위해서는 MATLAB프로그램 UpperBound_nVar.m, gradfunction.m과 GoldSection_nVar.m을 필요로 한다. 또한, 극소값을 구하고자 하는 목적함수를 지정하는 MATLAB프로그램이 필요하다. 다음 2차함수의 극소값을 구해보자.

$$f(x) = [x_1 - 2]^2 + [x_1 - 2][x_2 - 3] + [x_2 - 3]^2 \tag{1}$$

다음과 같은 MATLAB프로그램 SDMfunction.m을 먼저 작성하자.

```
% --------------------------------------------------------------------
function fcn = SDMfunction(x)
fcn =  (x(1)-2)^2 + (x(1)-2)*(x(2)-3) + (x(2)-3)^2;
% --------------------------------------------------------------------
```

MATLAB프로그램 SteepestDescent.m의 사용법은 다음과 같다.

```
>> SteepestDescent('SDMfunction',[2.5 0.2],20,1e-8,0,1,20)
```

여기서 첫 번째 변수에는 'SDMfunction'을 지정한다. 이는 MATLAB프로그램 SDMfunction.m을 의미하는 것으로, 이 MATLAB프로그램과 MATLAB프로그램 SteepestDescent.m은 동일한 디렉토리 안에 저장되어야 한다. 두 번째 입력변수 [2.5 0.2]는 초기벡터 $\boldsymbol{x}^{(1)}$이다. 세 번째 입력변수 20은 급하강법의 반복횟수를 나타내며, 네 번째 입력변수 1e-08은 황금섹션 탐색법과 이 알고리즘의 오차허용값을 나타낸다. 다섯 번째 입력변수 0와 여섯 번째 입력변수 1은 탐색하는 상계(upper bound)의 초기길이(initial stepsize)와 탐색간격(step interval)을 나타낸다. 또한, 일곱 번째 입력변수는 상계를 탐색하는 단계들의 개수를 나타낸다.

이 MATLAB명령문이 실행되면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

```
i = 15
    x(1)        x(2)        f(x)
2.5000e+000  2.0000e-001  6.6900e+000
3.1848e+000  2.1411e+000  1.1239e+000
2.0848e+000  2.5282e+000  1.8978e-001
2.1997e+000  2.8554e+000  3.1909e-002
2.0150e+000  2.9193e+000  5.5274e-003
2.0342e+000  2.9754e+000  9.3345e-004
2.0033e+000  2.9850e+000  1.8526e-004
2.0064e+000  2.9956e+000  3.1754e-005
2.0014e+000  2.9964e+000  1.0104e-005
2.0013e+000  2.9994e+000  1.2733e-006
2.0006e+000  2.9991e+000  5.9696e-007
2.0004e+000  2.9992e+000  5.3807e-007
2.0001e+000  2.9992e+000  4.9535e-007
1.9998e+000  2.9995e+000  4.0072e-007
1.9996e+000  3.0000e+000  1.6660e-007
1.9996e+000  3.0000e+000  1.6660e-007
```

이 결과물에서 알 수 있듯이, 제15번째 단계에서 $x_1 = 1.9996$과 $x_2 = 3.0000$에서 극소값이 1.6660×10^{-7}임을 알 수 있다.다. 또한, 이 MATLAB프로그램을 실행하면, 함수 $f(\boldsymbol{x})$의 레벨함수(contour map)와 극소점을 찾아가는 경로가 그림 6.3.2이 그려진다. 그림 6.3.2에서 볼 수 있듯이, 이 문제에서는 급하강법 알고리즘이 해석해로 빨리 수렴함을 알 수 있다. ■

예제 6.3.6 총에너지의 최소값을 구하는 문제를 다루기 위해서, 우선 다음 그림을 살펴보자.

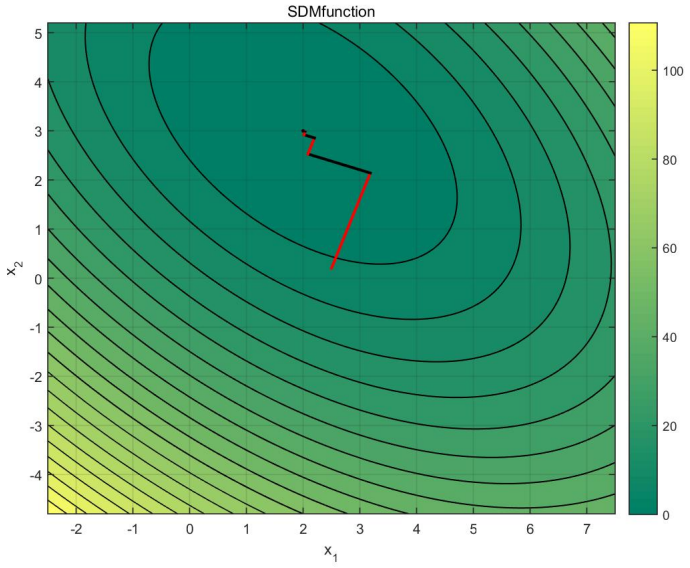
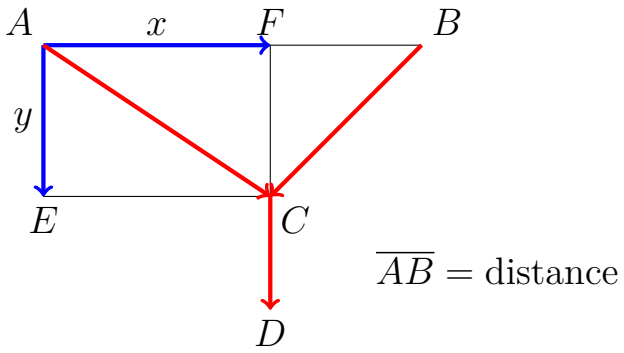


그림 6.3.2. 급하강법



이 그림에서 A와 B는 지면과 평행하며 거리가 $d = 1$인 점들이다. 이 점들에 길이가 $l = 1$이며 탄력상수(elastic constant)가 $k = 20\,\mathrm{Nt/m}$인 로프를 묶고, 이 로프에 질량이 $m = 1$인 물체를 걸면, 로프가 늘어나서 이 물체가 점 C에 위치한다고 하자. 또한, 벡터 $\overrightarrow{AB}$를 $[x, y]$로 표기하자. 이 물체의 총에너지(total energy) E는 다음과 같다.

$$E = E(x, y) = \frac{k}{2}\left[\sqrt{x^2 + y^2} + \sqrt{[d - x]^2 + y^2} - l\right]^2 - mgy \tag{1}$$

여기서 g는 중력가속도 $9.806\,\mathrm{m/s^2}$이다.

총에너지의 최소값을 관찰하기 위해서, 다음 MATLAB프로그램 EnergyMinimum101aDY.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: EnergyMinimum101aDY.m
```

```
% Energy Minimization
% Programmed by CDY
% ------------------------------------------------------------------------
close all, clear all
mass = 1, dist = 1, length = 1, elasticity = 20, g = 9.806
v = @(x,y) (elasticity*((x.^2 + y.^2).^(1/2) - length ...
            + (y.^2 + (dist - x).^2).^(1/2)).^2)/2 - g*mass*y
NoPoints = 100       % Number of points for each axis(100 is enough)
x = linspace(0,dist,NoPoints);
y = linspace(0,1,NoPoints);
[X,Y] = meshgrid(x,y);
Z = v(X,Y);
surf(X,Y,Z)
set(gca,'fontsize',11,'fontweigh','bold')
title('Energy Minimization','fontweigh','bold','fontsize',13)
xlabel('X'); ylabel('Y'); zlabel('Z')
saveas(gcf,'EnergyMinimum101aDYa.jpg')
% Contours
figure
contour(Z,30);
title('Isoquants of Energy Minimization','fontweigh','bold','fontsize',13)
set(gca,'fontsize',11,'fontweigh','bold','fontweigh','bold')
colorbar
xlabel('X'); ylabel('Y');
saveas(gcf,'EnergyMinimum101aDYb.jpg')
save('EnergyMinimum101aDY.txt','Z','-ascii')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 총에너지 E의 3차원 그래프와 등고선그림이 그려진다. 이 3차원 그래프는 그림 6.3.3에 그려져 있다. 또한 이 등고선그림이 그림 6.3.4에 그려져 있다. 이 그림들에서 알 수 있듯이, x가 0.5 그리고 y가 0.46 주변에서 총에너지 E가 최소값을 갖는다.

총에너지 E를 최소화하기 위해, 그래디언트벡터 (gradient)를 구하면 다음과 같다.

$$\nabla E = \begin{bmatrix} \frac{\partial E}{\partial x} \\ \frac{\partial E}{\partial y} \end{bmatrix} = \begin{bmatrix} k\left[\sqrt{x^2+y^2} + \sqrt{[d-x]^2+y^2} - l\right]\left[\frac{x}{\sqrt{x^2+y^2}} + \frac{x-d}{\sqrt{[d-x]^2+y^2}}\right] \\ k\left[\sqrt{x^2+y^2} + \sqrt{[d-x]^2+y^2} - l\right] y\left[\frac{1}{\sqrt{x^2+y^2}} + \frac{1}{\sqrt{[d-x]^2+y^2}}\right] - mg \end{bmatrix} \tag{2}$$

식 (2)의 그래디언트벡터를 사용해서 급하강법을 적용하기 위해, 다음 MATLAB프로그램 EnergyMinimum101bDY.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: EnergyMinimum101bDY.m
% Energy Minimization
% Programmed by CDY
% ------------------------------------------------------------------------
function EnergyMinimum101bDY
xold = 0, yold = 1
```

```
MaxIter = 100;
for kk = 1:MaxIter
    [v(kk),s(kk),x(kk),y(kk)] = LinesearchPotential(xold,yold);
    grad =  GradPotential(x(kk),y(kk));
    normDelV(kk) = norm(grad);
    normxy(kk) = norm([x(kk),y(kk)]);
    if normDelV(kk) < 10^(-7) || normDelV(kk) < 10^(-4)*normxy(kk)
        break;
    end
    xold = x(kk); yold = y(kk);
end
format long g
IterNum = 1:kk;
[IterNum; x(IterNum); y(IterNum); v(IterNum); ]'
save('EnergyMinimum101bDY.txt','x','y','-ascii')
% Plotting
subplot(2,1,1)
plot(IterNum,x,'r-o',IterNum,y,'k--*','LineWidth',1)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0,kk+1],'ylim',[0.4 0.65])
legend('\bf x','\bf y','location','NE')
subplot(2,1,2)
plot(IterNum,v,'b-*','LineWidth',1)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0,kk+1], ...
    'ylim',[-3.4 -3.0])
legend('\bf E','location','NE')
saveas(gcf,'EnergyMinimum101bDY.jpg')
end
% End of program
% ------------------------------------------------------------------------
function [v,s,xls,yls] = LinesearchPotential(x,y)
% Initial stage
grad =  GradPotential(x,y);
v = Potential(x,y);
vls = v; s = 1; n = 1; nmax = 50; eta = 1.0e-10; rho = 0.5;
% Iteration stage
while vls >= v-s*eta*norm(grad)^2 && n <= nmax
    s = rho*s;
    xls = x - s*grad(1);              % change from  x = x - s*grad(1)
    yls = y - s*grad(2);              % change from  y = y - s*grad(2)
    vls = Potential(xls,yls);
    n = n+1;
end
end
% ------------------------------------------------------------------------
function v = Potential(x,y)
   mass = 1, dist = 1, length = 1, elasticity = 20, g = 9.806
   v = (elasticity*((x.^2 + y.^2).^(1/2) - length ...
       + (y.^2 + (dist - x).^2).^(1/2)).^2)/2 - g*mass*y;
end
% ------------------------------------------------------------------------
function g = GradPotential(x,y)
       mass = 1, dist = 1, length = 1, elasticity = 20, g = 9.806;
       g1 = elasticity*(sqrt(x.^2+y.^2)+sqrt((dist-x).^2+y.^2)-length) ...
            *(x./sqrt(x.^2+y.^2)+(x-dist)./sqrt((x-dist).^2+y.^2));
       g2 = elasticity*(sqrt(x.^2+y.^2)+sqrt((dist-x).^2+y.^2)-length) ...
             .*(y./sqrt(x.^2+y.^2)+y./sqrt((x-dist).^2+y.^2)) - g*mass;
       g = [g1 ;  g2];
end
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 급하강법의 결과인 $\{[x^{(k)}, y^{(k)}]\}$가 그림 6.3.5의 상단 그래프에 그리고 최소값들 $\{E^{(k)} = E(x^{(k)}, y^{(k)})\}$가 그림 6.3.5의 하단 그래프에 그려진다. 이 그래프들에서 알 수 있듯이, $[x^*, y^*] = [0.50000, 0.46183]$에서 총에너지 E가 최소값 $E^* = -3.22330$을 갖는다. ■

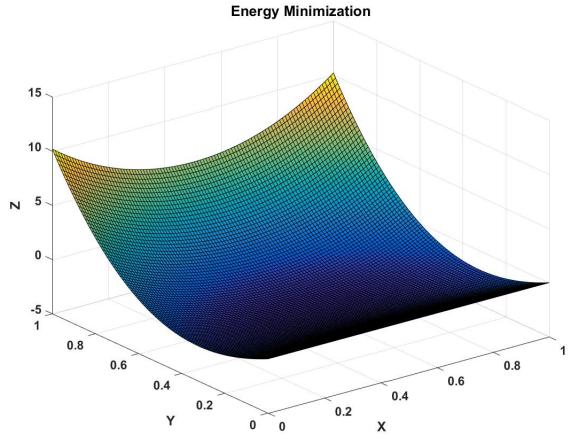


그림 6.3.3. 총에너지의 3차원 그래프

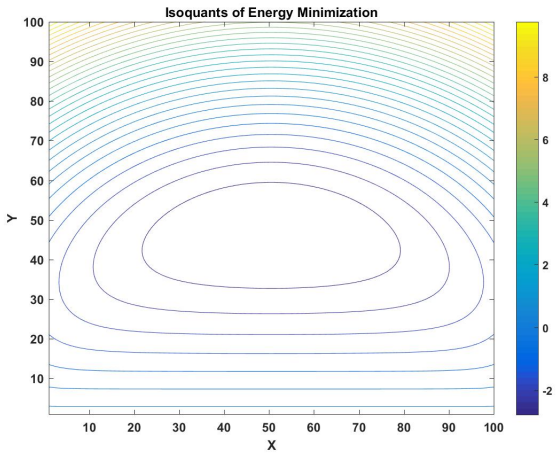


그림 6.3.4. 총에너지의 등고선그림

6.3.2 Newton-Raphson법

급하강법과 마찬가지로 Newton-Raphson법도 목적함수 $f(x)$의 그래디언트벡터 $\nabla f(x)$에 의존하는 방법으로서 목적함수의 극대점, 극소점 그리고 안장점(saddle point)에 관계없이 극점(critical point)을 찾는 기법이다. 즉, 그래디언트벡터가 영벡터가 되는 점을 찾는다.

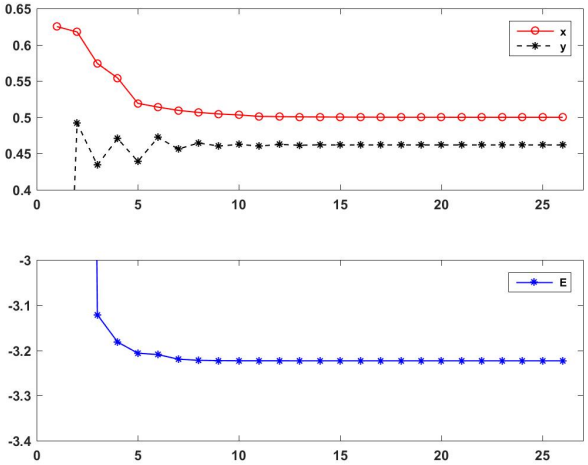


그림 6.3.5. 급하강해

어떤 점 $\boldsymbol{x}_0$ 근방에서 함수 $f(\boldsymbol{x})$의 2차 근사식은 다음과 같다.

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}_0) + [\boldsymbol{x}-\boldsymbol{x}_0]^t \nabla f(\boldsymbol{x}_0) + \frac{1}{2}[\boldsymbol{x}-\boldsymbol{x}_0]^t H(\boldsymbol{x}_0)[\boldsymbol{x}-\boldsymbol{x}_0] \tag{6.3.9}$$

식 (6.3.9)에서 알 수 있듯이, 함수 $f(\boldsymbol{x})$의 극값을 부여하는 $\boldsymbol{x}^*$는 다음 식을 만족한다.

$$\boldsymbol{x}^* = \boldsymbol{x}_0 - H^{-1}(\boldsymbol{x}_0)\nabla f(\boldsymbol{x}_0) \tag{6.3.10}$$

식 (6.3.10)이 다변수 Newton-Raphson식이다. 식 (6.3.10)에서 알 수 있듯이, 다변수 Newton-Raphson알고리즘은 다음 점화식을 사용해서 목적함수 $f(\boldsymbol{x})$의 극값을 갖는 점 $\boldsymbol{x}=a$에 수렴하는 벡터열 $\{\boldsymbol{x}^{(n)}\}$을 구하는 것이다.

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} - H^{-1}(\boldsymbol{x}^{(n)})\nabla f(\boldsymbol{x}^{(n)}), \quad (n=1,2,\cdots) \tag{6.3.11}$$

즉, 급경사법에서 탐색방향이 $\nabla f\left(\boldsymbol{x}^{(n)}\right)$인 반면에, 다변수 Newton-Raphson법은 탐색방향이 $H^{-1}(\boldsymbol{x}^{(n)})\nabla f(\boldsymbol{x}^{(n)})$이다.

다변수 Newton-Raphson알고리즘은 1변수인 경우와 마찬가지로, 극점 근방에서는 2차 수렴속도를 가지고 빠르게 수렴한다. 그러나, 극점 근방에서 초기점을 잡지 못하면, 구해진 $\{\boldsymbol{x}^{(n)}\}$이 수렴하지 않는 경우도 많다. 즉, Newton-Raphson알고리즘의 결과는 초기값에 의존한다. 또한, 극소점을 구하는 경우에 극대점이나 안장점으로 수렴하는 경우도 있다. 더구나, 각 반복단계에서 Hessian행렬의 역행렬을 계산할 필요가 있어, 이 방법을 적용하는데는 많은 계산시간이 걸린다.

예제 6.3.7 다변수 Newton-Raphson 법을 사용해서, 다음 함수의 극소값을 구해보자.

$$f(x) = [x_1 - 2]^4 + [x_1 - 2]^2[x_2 - 3]^2 + [x_2 - 3]^4 \tag{1}$$

점 (x_1, x_2) 에서 목적함수 $f(\boldsymbol{x})$ 의 그래디언트벡터와 Hessian 행렬은 각각 다음과 같다.

$$\nabla f = \begin{bmatrix} 4[x_1 - 2]^3 + [2x_1 - 4][x_2 - 3]^2 \\ 4[x_2 - 3]^3 + [2x_2 - 6][x_1 - 2]^2 \end{bmatrix} \tag{2}$$

$$H = \begin{bmatrix} 12[x_1 - 2]^2 + 2[x_2 - 3]^2 & 4[x_1 - 2][x_2 - 3] \\ 4[x_1 - 2][x_2 - 3] & 2[x_1 - 2]^2 + 12[x_2 - 3]^2 \end{bmatrix} \tag{3}$$

이 목적함수 $f(\boldsymbol{x})$ 의 극소값을 구하기 위해서, 다음 MATLAB 프로그램 MultiNewtonRaphson101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename MultiNewtonRaphson101.m
% Multivariate Newton-Raphson Method
% Programmed by CBS
% ------------------------------------------------------------------------
close all; clear all; clc; format long g
% Objective function, Gradient and Hessian matrix
syms x1 x2
fcn = '(x1-2)^4+(x1-2)^2*(x2-3)^2+(x2-3)^4'
grad = [diff(fcn,x1); diff(fcn,x2)];
H = [diff(diff(fcn,x1),x1),diff(diff(fcn,x1),x2); ...
    diff(diff(fcn,x2),x1),diff(diff(fcn,x2),x2)];
grad = simplify(grad)
H = simplify(H)
% Initialization
iiMax = 100;
xx = [ 1 ; 4]
ff = double(subs(subs(fcn,x1,xx(1)),x2,xx(2)));
xxmat(1,:) = xx'; ffvec(1) = ff;
% Iteration for ii = 1,2, ...
for ii = 1:1:iiMax
    Delfcn = subs(subs(grad,x1,xx(1)),x2,xx(2));
    Hfcn = subs(subs(H,x1,xx(1)),x2,xx(2));
    xxN = xx - Hfcn\Delfcn;
    ff = double(subs(subs(fcn,x1,xxN(1)),x2,xxN(2)));
    if norm(xx-xxN) < 1.0e-6
        break
    end
    xxmat(ii+1,:) = xxN'; ffvec(ii+1) = ff;
    xx = xxN;
end
ii
[ xxmat, ffvec']
% Plotting
nn = 1:1:ii;
hold on
plot(nn,xxmat(1:ii,1),'kd--',nn,xxmat(1:ii,2),'r*:','LineWidth',1.5)
```

```
set(gca,'fontsize',11,'fontweigh','bold')
legend('x_1','x_2',4)
xlabel('\bf Iteration Number n','fontsize',12)
ylabel('\bf x^{(n)}','fontsize',12,'rotation',0)
hold off
saveas(gcf,'MultiNewtonRaphson101','jpg')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 MultiNewtonRaphson101.m을 실행한 결과의 일부분은다음과 같다.

ii	x(1)	x(2)	f(x(1),x(2))
27	1.99997359858131	3.00002640141869	1.45757299352879e-18
28	1.9999823990542	3.0000176009458	2.87915653042723e-19
29	1.99998826603614	3.00001173396386	5.68722277615256e-20
30	1.99999217735742	3.00000782264258	1.1234020298573e-20
31	1.99999478490495	3.00000521509505	2.21906573798972e-21
32	1.99999652326997	3.00000347673003	4.38333972936241e-22
33	1.99999768217998	3.00000231782002	8.65844884812328e-23
34	1.99999845478665	3.00000154521335	1.71031088357991e-23

이 결과물에서 알 수 있듯이, 제34번째 단계에서 $x_1 = 2.0000$과 $x_2 = 3.0000$에서 극소값이 0임을 알 수 있다. 또한, 이 MATLAB프로그램을 실행하면, 그림 6.3.6이 그려진다. 그림 6.3.6에서 볼 수 있듯이, 이 문제에서는 다변수 Newton-Raphson알고리즘이 해석해로 빨리 수렴함을 알 수 있다. ■

예제 6.3.8 Python을 사용해서 예제 6.3.7을 다시 다루기 위해서, 다음 Python프로그램 MultiNewtonRaphson101.Py를 실행해보라.

```
"""
Created on Thu Jan 12 20:13:46 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from sympy import Symbol, Matrix
# from sympy.matrices import Matrix

# Rosenbrock funtion and its gradient
x1 = Symbol('x1');   x2 = Symbol('x2')
fcn = (x1-2)**4+(x1-2)**2*(x2-3)**2+(x2-3)**4
grad = Matrix([ [ diff(fcn,x1), diff(fcn,x2)] ]);
H = Matrix([ [ diff(diff(fcn,x1),x1), diff(diff(fcn,x1),x2) ], \
             [ diff(diff(fcn,x2),x1), diff(diff(fcn,x2),x2)  ]  ])
```

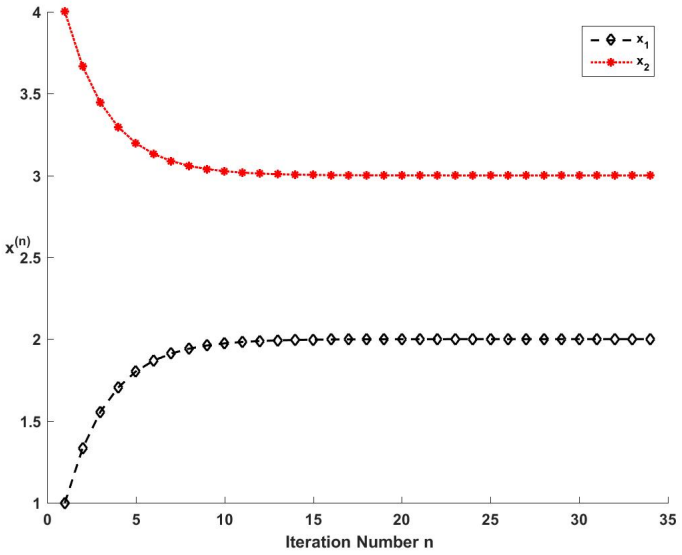


그림 6.3.6. 다변수 Newton-Raphson법

```
17
18 # Initialization
19 iiMax = 100;
20 xx = np.array([ [1],[4] ]);
21 ff = fcn.subs([ (x1, xx[0]), (x2, xx[1]) ]);
22 xxmat = np.zeros((iiMax,2));
23 ffvec = np.zeros((iiMax,1));
24 xxmat[0,:] = np.transpose(xx);
25 ffvec[0] = ff;
26
27 # Iteration for ii = 1,2, ...
28 for ii in range(0,iiMax):
29     Delfcn2 = grad.subs( [ (x1,xx[0]), (x2,xx[1]) ]);
30     Delfcn1 = np.squeeze(np.asarray(Delfcn2));
31     Delfcn = np.array(list(Delfcn1[:]), dtype=np.float)
32     Hfcn2 = H.subs( [ (x1,xx[0]), (x2,xx[1]) ]);
33     Hfcn1 = np.squeeze(np.asarray(Hfcn2));
34     Hfcn = np.array(list(Hfcn1[:,:]), dtype=np.float)
35     dum11 = np.matmul(np.linalg.inv(Hfcn), Delfcn).reshape((2,1));
36     xxN = xx - dum11;
37     ff = fcn.subs( [ (x1,xx[0]), (x2,xx[1]) ]);
38     if np.linalg.norm(xx-xxN) < 1.0e-6:
39         break
40     xxmat[ii+1,:] = np.transpose(xxN)
41     ffvec[ii+1] = ff;
42     xx = xxN;
43 print(xxmat[0:ii,:]); print(ffvec[0:ii])
44
45
46 # Plotting
47 fig = plt.figure()
48 nn = np.arange(0,ii,1);
49 plt.plot(nn,xxmat[0:ii,0],'kd--',lw=2,label=r'$x_1$')
50 plt.plot(nn,xxmat[0:ii,1],'r*:',lw=2,label=r'$x_2$')
51 plt.legend(loc='upper right', numpoints=1)
52 plt.xlabel('Iteration Number n', fontsize=12)
53 plt.ylabel(r'$x^{(n)}$', fontsize=12)
```

```
plt.show()
fig.savefig('MultiNewtonRaphson101Py.png')

# End of program
```

■

함수 $f(\boldsymbol{x})$가 점 $\boldsymbol{x}^*$에서 극소값을 갖고 초기점 $\boldsymbol{x}^{(0)}$가 점 $\boldsymbol{x}^*$에 충분히 가까운 점이라고 하자. Newton-Raphson알고리즘에 의해 생성된 벡터열 $\{\boldsymbol{x}^{(n)}\}$에 대해 다음 식을 만족하는 양수 α가 존재한다.

$$\lim_{n\to\infty} \frac{\|\boldsymbol{x}^{(n+1)} - \boldsymbol{x}^*\|}{\|\boldsymbol{x}^{(n)} - \boldsymbol{x}^*\|^2} = \alpha \tag{6.3.12}$$

같은 조건 하에서, 2차함수근사탐색법이나 뒤에서 다루게 될 준Newton법(quasi-Newton method)에 의해 생성된 점열 $\{\boldsymbol{x}^{(n)}\}$에 대해 다음 식을 만족하는 양수 α와 양수 $r(\in (1,2])$이 존재한다.

$$\lim_{n\to\infty} \frac{\|\boldsymbol{x}^{(n+1)} - \boldsymbol{x}^*\|}{\|\boldsymbol{x}^{(n)} - \boldsymbol{x}^*\|^r} = \alpha(>0) \tag{6.3.13}$$

만약 좀 더 많은 정보를 이용할 수 있다면, 식 (6.3.13)의 지수 r이 2보다 큰 알고리즘을 만들 수도 있다. 그러나, 일반적으로 r이 구간 $(1,2)$에 속하면 만족스러운 것이다.

6.3.3 Davidon-Fletcher-Powell법

Newton-Raphson법에서는 각 단계에서 Hessian행렬의 역행렬 $H^{-1}(\boldsymbol{x}^{(n)})$을 구해야 한다. 그러나, 역행렬 $H^{-1}(\boldsymbol{x})$를 수학적으로 구하는 것이 곤란하거나, 또는 그 계산이 아주 복잡한 경우가 많다. 따라서, $H^{-1}(\boldsymbol{x})$를 직접 구하지 않는 계산법이 연구되었고, 이를 준Newton법(quasi-Newton method, psuedo-gradient method)이라 부른다. 그 중에서 가장 널리 알려진 Davidon-Fletcher-Powell법이다. 이 방법은 급하강법과 Newton-Raphson법을 혼용해서 극소점을 찾는 것이다.

알고리즘 6.3.3: Davidon-Fletcher-Powell법

(1단계) 알고리즘의 최대 반복횟수 N 그리고 오차허용값들 ϵ_1, ϵ_2 그리고 ϵ_3를 선택하고, $n = 1$이라 한다. 초기벡터 $\boldsymbol{x}^{(1)}$에서 함수값 $f(\boldsymbol{x}^{(1)})$을 구하고, Hessian행렬의 역행렬의 초기행렬로 양정치행렬 $G^{(1)}$을 선택한다.

(2단계) 다음 벡터들을 계산한다.

$$\boldsymbol{s}^{(n)} = -G^{(n)}\nabla f(\boldsymbol{x}^{(n)})$$
$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \alpha^{(n)}\boldsymbol{s}^{(n)}$$
$$\Delta\boldsymbol{x}^{(n)} = \alpha^{(n)}\boldsymbol{s}^{(n)}$$

여기서 $\alpha^{(n)}$은 제n단계의 탐색방향에서 $f\left(\boldsymbol{x}^{(n+1)}\right)$의 극소값을 부여하는 상수이고 $G^{(n)}$은 Hessian행렬의 역행렬이다.

(3단계) 만약 식 $\nabla f(\boldsymbol{x}^{(n)})^t\nabla f(\boldsymbol{x}^{(n)}) \leq \epsilon_1$이 성립하면, 이 알고리즘을 멈춘다. 만약 식 $\|\nabla f(\boldsymbol{x}^{(n)}) - \nabla f(\boldsymbol{x}^{(n+1)})\| \leq \epsilon_2$가 성립하면, 함수가 감소하지 않으므로 이 알고리즘을 멈춘다. 만약 식 $[\Delta\boldsymbol{x}^{(n)}]^t\Delta\boldsymbol{x}^{(n)} \leq \epsilon_3$가 성립하면, 디자인이 바뀌지 않고 따라서 이 알고리즘을 멈춘다. 만약 $n+1 = N$이면, 반복횟수의 제한에 의해서 이 알고리즘을 멈춘다. 위 조건들에 해당사항이 없으면, 제4단계로 넘어간다.

(4단계) 다음 벡터들과 행렬들을 계산한다.

$$\boldsymbol{y}^{(n)} = \nabla f(\boldsymbol{x}^{(n+1)}) - \nabla f(\boldsymbol{x}^{(n)})$$
$$\boldsymbol{z}^{(n)} = G^{(n)}\boldsymbol{y}^{(n)}$$
$$B^{(n)} = \frac{1}{[\Delta\boldsymbol{x}^{(n)}]^t\boldsymbol{y}^{(n)}}\Delta\boldsymbol{x}^{(n)}[\Delta\boldsymbol{x}^{(n)}]^t$$
$$C^{(n)} = -\frac{1}{[\boldsymbol{y}^{(n)}]^t\boldsymbol{z}^{(n)}}\boldsymbol{z}^{(n)}[\boldsymbol{z}^{(n)}]^t$$
$$G^{(n+1)} = G^{(n)} + B^{(n)} + C^{(n)}$$

또한, n을 $n+1$로 증가시킨 다음 제2단계로 넘어간다.

웹에서 알고리즘 6.3.3를 구현하는 간단한 프로그램 하나를 다운받아서, 이 알고리즘을 실행해보자. 다음 예제에서 사용하는 프로그램은 Namir Shammas가 작성한 것이다. 그의 웹사이트 www.namirshammas.com에 가면, 다양한 프로그래밍언어로 작성된 프로그램들을 다운받을 수 있다.

예제 6.3.9 Davidon-Fletcher-Powell법을 사용해서, 다음 함수의 극소값을 구해보자.

$$f(\boldsymbol{x}) = [x_1 - 3]^2 + [x_2 - 5]^2 + \exp([x_1 - 3][x_2 - 5]) \tag{1}$$

이 목적함수 $f(\boldsymbol{x})$의 극소값을 구하기 위해서, 다음 MATLAB프로그램 DFP_NS.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: DFP_NS.m
% Multivariate optimization using
% Davidon-Fletcher-Powell method
% by Namir Shammas @ www.namirshammas.com
% ------------------------------------------------------------------------
function DFP_NS
% Input
N = 2;                    % number of variables
X = [ 0 0 ];              % array of initial guesses
gradToler = 1e-7;         % tolerance for the norm of the slopes
fxToler = 1e-7;           % tolerance for function
DxToler = [1e-5 1e-5];    % array of delta X tolerances
MaxIter = 1000;            % maximum number of iterations
myFx = 'fx1';             % name of the optimized function
% initials
B = eye(N,N);
bGoOn = true;
Iters = 0;
% calculate initial gradient
grad1 =  FirstDerivatives(X, N, myFx);
grad1 = grad1';
while bGoOn
  Iters = Iters + 1;
  if Iters > MaxIter
    break;
  end
  S = -1 * B * grad1;
  S = S' / norm(S); % normalize vector S
  lambda = 1;
  lambda = linsearch(X, N, lambda, S, myFx);
  % calculate optimum X() with the given Lambda
  d = lambda * S;
  X = X + d;
  % get new gradient
  grad2 =  FirstDerivatives(X, N, myFx);
  grad2 = grad2';
  g = grad2 - grad1;
  grad1 = grad2;
  % test for convergence
  for i = 1:N
    if abs(d(i)) > DxToler(i)
      break
    end
  end
  if norm(grad1) < gradToler
    break
  end
%  B = B + lambda * (S * S') / (S' * g) - ...
%      (B * g) * (B * g') / (g' * B * g);
  x1 = (S * S');
  x2 = (S * g);
  B = B + lambda * x1 * 1 / x2;
  x3 = B * g;
  x4 = B' * g;
  x5 = g' * B * g;
  B = B - x3 * x4' / x5;
```

```
end
F = feval(myFx, X, N);
% Output
X         % array of optimized variables
F         % function value at optimum
Iters     % number of iterations
end
% ---------------------------------------------------------------------
function y = myFxEx(N, X, DeltaX, lambda, myFx)
  X = X + lambda * DeltaX;
  y = feval(myFx, X, N);
end
% ---------------------------------------------------------------------
function FirstDerivX = FirstDerivatives(X, N, myFx)
 for iVar=1:N
  xt = X(iVar);
  h = 0.01 * (1 + abs(xt));
  X(iVar) = xt + h;
  fp = feval(myFx, X, N);
  X(iVar) = xt - h;
  fm = feval(myFx, X, N);
  X(iVar) = xt;
  FirstDerivX(iVar) = (fp - fm) / 2 / h;
 end
end
% ---------------------------------------------------------------------
function lambda = linsearch(X, N, lambda, D, myFx)
  MaxIt = 100;
  Toler = 0.000001;
  iter = 0;
  bGoOn = true;
  while bGoOn
    iter = iter + 1;
    if iter > MaxIt
      lambda = 0;
      break
    end
    h = 0.01 * (1 + abs(lambda));
    f0 = myFxEx(N, X, D, lambda, myFx);
    fp = myFxEx(N, X, D, lambda+h, myFx);
    fm = myFxEx(N, X, D, lambda-h, myFx);
    deriv1 = (fp - fm) / 2 / h;
    deriv2 = (fp - 2 * f0 + fm) / h ^ 2;
    diff = deriv1 / deriv2;
    lambda = lambda - diff;
    if abs(diff) < Toler
      bGoOn = false;
    end
  end

end
% ---------------------------------------------------------------------
function y=fx1(X, N)
  y = pi + (X(1) - 3)^2 + (X(2) - 5)^2 + exp((X(1)-3)*(X(2)-5));
end
% ---------------------------------------------------------------------
```

이 MATLAB프로그램 DFP_NS.m을 실행하면, 제5번째 단계에서 이 알고리즘이 멈추며 또한 점 $(3.0000, 5.0000)$에서 최소값 4.1416을 얻는다. ■

Venkataraman [63, 제6.3.3소절])에는 Davidon-Fletcher-Powell법이 자세히 소개되어 있다. 또한, 이 방법을 구현하기 위한 MATLAB프로그램 DFP.m이 제공되어 있다. 다음 예제는 이 MATLAB프로그램의 입출력 부분을 약간 변형한 MATLAB프로그램을 사용해서 Davidon-Fletcher-Powell법을 적용하는 예이다.

예제 6.3.10 Venkataraman [63])에 수록된 MATLAB프로그램 DFP.m을 사용하기 위해서는 MATLAB프로그램들 UpperBound_nVar.m, gradfunction.m과 GoldSection_nVar.m을 필요로 한다. 또한, 극소값을 구하고자 하는 목적함수를 지정하는 MATLAB프로그램이 필요하다. 여기서는 다음 2차함수의 극소값을 구해보자.

$$f(x) = [x_1 - 2]^2 + [x_1 - 2][x_2 - 3] + [x_2 - 3]^2 \tag{1}$$

이 함수를 MATLAB프로그램 SDMfunction.m에 기록한다.

MATLAB프로그램 DFP.m의 사용법은 다음과 같다.

```
>> DFP('SDMfunction',[2.5 0.2],20,1e-8,0,1,20)
```

첫 번째 입력변수에는 'SDMfunction'를 지정하였다. 이는 MATLAB프로그램 SDMfunction.m을 의미하는 것으로, 이 MATLAB프로그램과 MATLAB프로그램 DFP.m는 동일한 디렉토리 안에 저장되어야 한다. 두 번째 입력변수 [2.5 0.2]는 초기벡터 $\boldsymbol{x}^{(1)}$이다. 세 번째 입력변수 20은 Davidon-Fletcher-Powell법의 반복횟수를 나타내며, 네 번째 입력변수 1e-08은 황금섹션탐색법의 오차허용값을 나타낸다. 다섯 번째 입력변수 0와 여섯 번째 입력변수 1은 탐색하는 상계(upper bound)의 초기길이(initial stepsize)와 탐색간격(step interval)을 나타낸다. 또한, 일곱 번째 입력변수는 상계를 탐색하는 단계들의 개수를 나타낸다.

이 MATLAB명령문을 실행하면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

Iter	x(1)	x(2)	f(x)
0	2.5000e+000	2.0000e-001	6.6900e+000
1	3.1848e+000	2.1411e+000	1.1239e+000
2	1.9998e+000	2.9996e+000	2.8083e-007
3	1.9998e+000	2.9996e+000	2.8083e-007

이 결과물에서 알 수 있듯이, 제3번째 단계에서 알고리즘을 멈추며, 이때 $x_1 = 1.9998$과 $x_2 = 2.9996$에서 극소값이 2.8083×10^{-7}임을 알 수 있다. 또한, 이 MATLAB명령분이 실행되면, 그림 6.3.7처럼 함수 $f(x)$의 레벨함수(contour map)와 극소점을 찾아가는 경로가 출력된다. 결과물과 그림 6.3.7에서 알 수 있듯이, 급경사법보다 Davidon-Fletcher-Powell법이 훨씬 더 빨리 극점을 찾아간다. ■

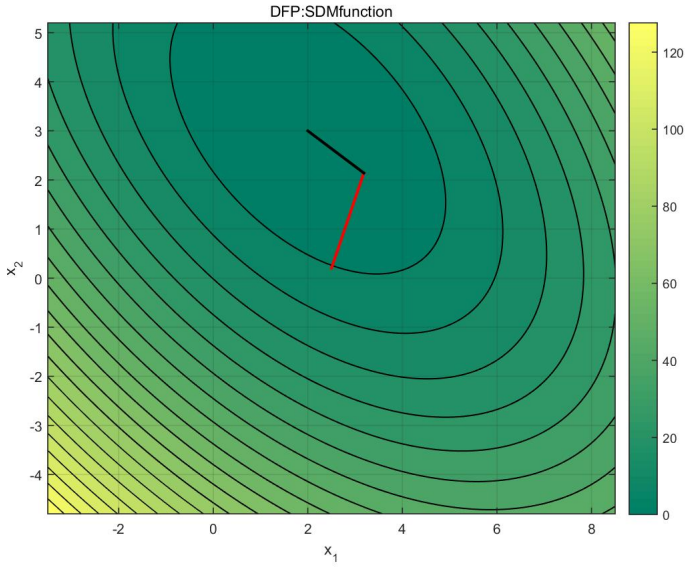


그림 6.3.7. DFP 법

6.3.4 Broydon-Fletcher-Goldfarb-Shanno 법

Broydon-Fletcher-Goldfarb-Shanno 법 (BFGS 법) 도 널리 사용되는 준Newton 법 중 하나이다. Davidon-Fletcher-Powell 법에서는 Hessian 행렬의 역행렬을 반복해서 갱신하는 반면에 BFGS 법에서는 역행렬 그 자체를 반복해서 갱신한다.

알고리즘 6.3.4: BFGS 법

(1단계) 이 알고리즘의 최대 반복횟수 N 그리고 오차허용값들 ϵ_1, ϵ_2 그리고 ϵ_3 를 선택하고, $n = 1$ 이라 한다. 초기벡터 $\boldsymbol{x}^{(1)}$ 에서 함수값 $f(\boldsymbol{x}^{(1)})$ 을 구하고, Hessian 행렬의 초기행렬로 양정치행렬 $H^{(1)}$ 을 선택한다.

(2단계) 다음 식을 만족하는 벡터들 $H^{(n)}$, $\boldsymbol{x}^{(n+1)}$ 그리고 $\Delta\boldsymbol{x}^{(n)}$ 을 계산한다.

$$
\begin{aligned}
H^{(n)}\boldsymbol{s}^{(n)} &= -\nabla f(\boldsymbol{x}^{(n)}) \\
\boldsymbol{x}^{(n+1)} &= \boldsymbol{x}^{(n)} + \alpha^{(n)}\boldsymbol{s}^{(n)} \\
\Delta\boldsymbol{x}^{(n)} &= \alpha^{(n)}\boldsymbol{s}^{(n)}
\end{aligned}
$$

여기서 $\alpha^{(n)}$ 은 제 n 단계의 탐색방향에서 $f(\boldsymbol{x}^{(n+1)})$ 의 극소값을 부여하는 상수이고 $H^{(n)}$ 은 Hessian 행렬이다.

(3단계) 만약 식 $\nabla f(\boldsymbol{x}^{(n)})^t \nabla f(\boldsymbol{x}^{(n)}) \leq \epsilon_1$ 이 성립하면, 이 알고리즘을 멈춘다. 만약

식 $\|\nabla f(\boldsymbol{x}^{(n)}) - \nabla f(\boldsymbol{x}^{(n)})\| \leq \epsilon_2$가 성립하면, 목적함수가 감소하지 않으므로 이 알고리즘을 멈춘다. 만약 식 $[\Delta\boldsymbol{x}^{(n)}]^t\Delta\boldsymbol{x}^{(n)} \leq \epsilon_3$가 성립하면, 디자인이 바뀌지 않으므로 이 알고리즘을 멈춘다. 만약 $n+1=N$이면, 반복횟수의 제한에 의해서 이 알고리즘을 멈춘다. 위 조건에 해당사항이 없으면, 제4단계로 넘어간다.

(4단계) 다음 벡터들과 행렬들을 계산한다.

$$\begin{aligned}
\boldsymbol{y}^{(n)} &= \nabla f(\boldsymbol{x}^{(n+1)}) - \nabla f(\boldsymbol{x}^{(n)}) \\
B^{(n)} &= \frac{1}{[\boldsymbol{y}^{(n)}]^t\Delta\boldsymbol{x}^{(n)}}\boldsymbol{y}^{(n)}[\boldsymbol{y}^{(n)}]^t \\
C^{(n)} &= -\frac{1}{[\nabla f(\boldsymbol{x}^{(n)})]^t\boldsymbol{s}^{(n)}}\nabla f(\boldsymbol{x}^{(n)})[\nabla f(\boldsymbol{x}^{(n)})]^t \\
H^{(n+1)} &= H^{(n)} + B^{(n)} + C^{(n)}
\end{aligned}$$

또한, n을 $n+1$로 증가시킨 다음 제2단계로 넘어간다.

알고리즘 6.3.4를 구현하는 많은 프로그램들이 있다. 이 중에서 간단한 프로그램을 하나 사용해서 이 알고리즘을 실행해보자. 다음 예제에서 사용하는 프로그램은 Namir Shammas가 작성한 것이다.

예제 6.3.11 다변수 BFGS법을 사용해서, 다음 함수의 극소값을 구해보자.

$$f(\boldsymbol{x}) = [x_1 - 3]^2 + [x_2 - 5]^2 + \exp([x_1 - 3][x_2 - 5]) \tag{1}$$

이 목적함수 $f(\boldsymbol{x})$의 극소값을 구하기 위해서, 다음 MATLAB프로그램 BFGS_NS.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename: BFGS_NS.m
% Multivariate optimization using
% Broyden-Fletcher-Goldfarb-Shanno method
% by Namir Shammas @ www.namirshammas.com
% ------------------------------------------------------------------------
function BFGS_NS
% Function bfgs performs multivariate optimization using the
% Broyden-Fletcher-Goldfarb-Shanno method.
%
% Input
N = 2;                  % number of variables
X = [ 0 0 ];            % array of initial guesses
gradToler = 1e-7;       % tolerance for the norm of the slopes
fxToler = 1e-7;         % tolerance for function
DxToler = [1e-5 1e-5];  % array of delta X tolerances
MaxIter = 100;          % maximum number of iterations
```

```
myFx = 'fx1';            % name of the optimized function
       % myFx = @(x) 10 + (X(1) - 2)^2 + (X(2) + 5)^2
% initials
B = eye(N,N);
bGoOn = true;
Iters = 0;
% calculate initial gradient
grad1 =  FirstDerivatives(X, N, myFx);
grad1 = grad1';
while bGoOn
  Iters = Iters + 1;
  if Iters > MaxIter
    break;
  end
  S = -1 * B * grad1;
  S = S' / norm(S); % normalize vector S
  lambda = 1;
  lambda = linsearch(X, N, lambda, S, myFx);
  % calculate optimum X() with the given Lambda
  d = lambda * S;
  X = X + d;
  % get new gradient
  grad2 =  FirstDerivatives(X, N, myFx);
  grad2 = grad2';
  g = grad2 - grad1;
  grad1 = grad2;
  % test for convergence
  for i = 1:N
    if abs(d(i)) > DxToler(i)
      break
    end
  end
  if norm(grad1) < gradToler
    break
  end
  d = d';
  x1 = d * d';
  x2 = d' * g;
  x3 = d * g';
  x4 = g * d';
  x5 = g' * B * g;
  x6 = d * g' * B;
  x7 = B * g * d';
  B = B + (1 + x5 / x2) * x1 / x2 - x6 / x2 - x7 / x2;
  % break
end
F = feval(myFx, X, N);
% Output
X        % array of optimized variables
F        % function value at optimum
Iters    % number of iterations
end
% ------------------------------------------------------------------------
function FirstDerivX = FirstDerivatives(X, N, myFx)
  for iVar=1:N
     xt = X(iVar);
     h = 0.01 * (1 + abs(xt));
     X(iVar) = xt + h;
     fp = feval(myFx, X, N);
     X(iVar) = xt - h;
     fm = feval(myFx, X, N);
```

```
79      X(iVar) = xt;
80      FirstDerivX(iVar) = (fp - fm) / 2 / h;
81    end
82  end
83  % --------------------------------------------------------------------
84  function lambda = linsearch(X, N, lambda, D, myFx)
85    MaxIt = 100;
86    Toler = 0.000001;
87
88    iter = 0;
89    bGoOn = true;
90    while bGoOn
91      iter = iter + 1;
92      if iter > MaxIt
93        lambda = 0;
94        break
95      end
96      h = 0.01 * (1 + abs(lambda));
97      f0 = myFxEx(N, X, D, lambda, myFx);
98      fp = myFxEx(N, X, D, lambda+h, myFx);
99      fm = myFxEx(N, X, D, lambda-h, myFx);
100     deriv1 = (fp - fm) / 2 / h;
101     deriv2 = (fp - 2 * f0 + fm) / h ^ 2;
102     if deriv2 == 0
103       break
104     end
105     diff = deriv1 / deriv2;
106     lambda = lambda - diff;
107     if abs(diff) < Toler
108       bGoOn = false;
109     end
110   end
111 end
112 % --------------------------------------------------------------------
113 function y = myFxEx(N, X, DeltaX, lambda, myFx)
114   X = X + lambda * DeltaX;
115   y = feval(myFx, X, N);
116 end
117 % --------------------------------------------------------------------
118 function y=fx1(X, N)
119   y = pi + (X(1) - 3)^2 + (X(2) - 5)^2 + exp((X(1)-3)*(X(2)-5));
120 end
121 % --------------------------------------------------------------------
```

이 MATLAB프로그램 BFGS_NS.m을 실행하면, 제6번째 단계에서 이 알고리즘이 멈추며 또한 점 $(3.0000, 5.0000)$에서 최소값 4.1416을 얻는다. ■

Venkataraman [63, 제6.3.4소절]에는 BFGS법이 자세히 소개되어 있다. 또한, 이 방법을 구현하기 위한 MATLAB프로그램 BFGS.m이 제공되어 있다. 다음 예제는 이 MATLAB 프로그램의 입출력 부분을 약간 변형한 MATLAB프로그램을 사용하는 예이다.

예제 6.3.12 MATLAB프로그램 BFGS.m을 사용하기 위해서는 MATLAB프로그램 UpperBound_nVar.m, gradfunction.m과 GoldSection_nVar.m을 필요로 한다. 또한, 극소값을

구하고자 하는 목적함수를 지정하는 MATLAB프로그램이 필요하다. 여기서는 다음 2차함수의 극소값을 구해보자.

$$f(\boldsymbol{x}) = [x_1 - 2]^2 + [x_1 - 2][x_2 - 3] + [x_2 - 3]^2 \tag{1}$$

이 함수를 MATLAB프로그램 SDMfunction.m에 기록한다.

MATLAB프로그램 BFGS.m의 사용법은 다음과 같다.

```
>> BFGS('SDMfunction',[2.5 0.2],20,1e-8,0,1,20)
```

여기서 첫 번째 변수에는 'SDMfunction'을 지정하였다. 이는 MATLAB프로그램 SDMfunction.m을 의미하는 것으로, 이 MATLAB프로그램과 MATLAB프로그램 BFGS.m은 동일한 디렉토리 안에 저장되어야 한다. 두 번째 입력변수 [2.5 0.2]는 초기벡터 $\boldsymbol{x}^{(1)}$이다. 세 번째 입력변수 20은 BFGS법의 반복횟수를 나타내며, 네 번째 입력변수 1e-08은 황금섹션탐색법과 이 알고리즘의 오차허용값을 나타낸다. 다섯 번째 입력변수 0와 여섯 번째 입력변수 1은 탐색하는 상계(upper bound)의 초기길이(initial stepsize)와 탐색간격(step interval)을 나타낸다. 또한, 일곱 번째 입력변수는 상계를 탐색하는 단계들의 개수를 나타낸다.

이 MATLAB명령문이 실행되면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

```
The design vector,function value and KT value
during the iterations
  2.5000e+000  2.0000e-001  6.6900e+000            0  2.9236e+001
  3.1848e+000  2.1411e+000  1.1239e+000  3.8068e-001  2.5684e+000
  1.9999e+000  2.9996e+000  2.3699e-007  8.7571e-001  1.0395e-007
```

이 결과물에서 알 수 있듯이 제2번째 단계에서 이 알고리즘을 멈추며, 이때 $x_1 = 1.9999$와 $x_2 = 2.9996$에서 극소값이 2.3699×10^{-7}임을 알 수 있다. 또한, 이 MATLAB명령분이 실행되면, 그림 6.3.8처럼 함수 $f(\boldsymbol{x})$의 레벨함수(contour map)와 극소점을 찾아가는 경로가 출력된다. 그림 6.3.8을 그림 6.3.7와 비교하면, 이 예제에서는 Davidon-Fletcher-Powell법보다 BFGS법이 더 빨리 극소점을 찾아감을 알 수 있다. ■

예제 6.3.13 Python함수 scipy.optimize.fmin_bfgs를 사용해서 BFGS법을 적용하는 예로서 다음 Python프로그램 BFGS101.Py를 실행해 보자.

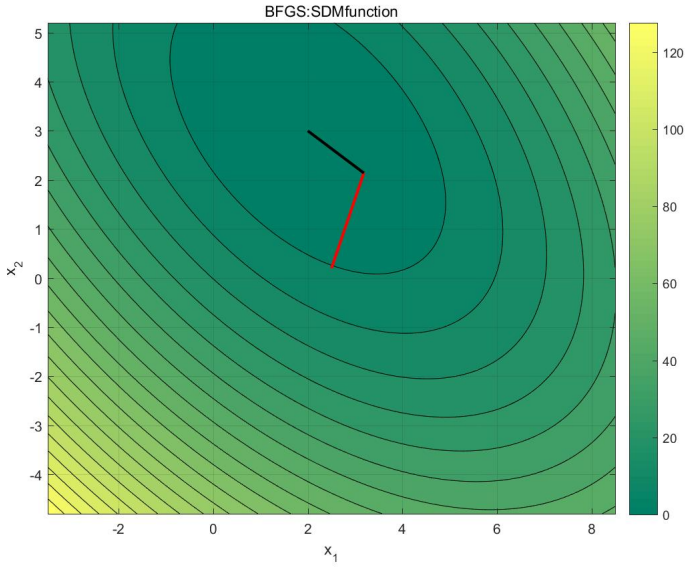


그림 6.3.8. BFGS법

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy import optimize

def f(x):   # The rosenbrock function
    return 0.5*(1 - x[0])**2 + (x[1] - x[0]**2)**2
def df(x):
    return np.array((-2*.5*(1 - x[0]) - 4*x[0]*(x[1] - x[0]**2), \
                     2*(x[1] - x[0]**2)))
xopt1 = optimize.fmin_bfgs(f, [2, 2], fprime=df)
print('xopt1 = ', xopt1, ',  fval = ', f(xopt1))

# End of program
```

이 예제에서는 다음과 같은 함수의 극소값을 구하고자 한다.

$$f(x,y) = 0.5[1-x]^2 + \left[y - x^2\right]^2 \tag{1}$$

함수 f의 편도함수들은 다음과 같다.

$$f_x = [1-x] - 2x\left[y - x^2\right], \quad f_y = 2\left[y - x^2\right] \tag{2}$$

식 (2)에서 알 수 있듯이, 점 $[1,1]$에서 극소값 0을 갖는다.

이 Python프로그램을 수행한 결과는 다음과 같다.

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 8
         Function evaluations: 9
         Gradient evaluations: 9

xopt1 =  [ 1.00000582  1.00001285] ,  fval =  1.84140934073e-11
```

■

예제 6.3.14 Python을 사용해서 예제 6.3.13을 다시 다루기 위해서, 다음 Python프로그램 BFGS102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy import optimize

def f(x):   # The rosenbrock function
    return 0.5*(1 - x[0])**2 + (x[1] - x[0]**2)**2
def df(x):
    return np.array((-2*.5*(1 - x[0]) - 4*x[0]*(x[1] - x[0]**2), \
                     2*(x[1] - x[0]**2)))
xopt1 = optimize.fmin_l_bfgs_b(f, [2, 2], fprime=df)
print(xopt1)

# End of program
```

이 Python프로그램에서는 Python함수 scipy.optimize.fmin_l_bfgs_b을 사용하였다. 이 Python함수는 메모리를 적게 사용하는 L-BFGS법(limited-memory BFGS method)를 사용한다. 이 방법은 계산적 측면에서 볼 때 BFGS법과 다음 소절에서 다루는 공액방향법의 중간 정도에 위치하는 것으로 차수가 250 이상인 경우 주로 사용한다. 이 방법에서는 Hessian 행렬을 사용하지 않는다.

이 Python프로그램을 수행한 결과는 다음과 같다.

```
(array([ 1.00000005,  1.00000009]),
 1.4417677473011859e-15,
 {'funcalls': 17,
  'grad': array([  1.02331202e-07,  -2.59299369e-08]),
  'nit': 16,
  'task': b'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL',
  'warnflag': 0})
```

■

6.3.5 공액방향법

공액방향법(conjugate directional method)을 사용해서, 다음과 같은 벡터 $\boldsymbol{x}(\in R^N)$에 관한 2차 목적함수 $f(\boldsymbol{x})$의 최소값을 구하는 문제를 살펴보자.

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \min_{\boldsymbol{x}} \left[\frac{1}{2}\boldsymbol{x}^t H\boldsymbol{x} + c^t\boldsymbol{x}\right] \tag{6.3.14}$$

여기서 H는 양정치행렬이다.

공액방향법을 설명하기 위해서, 먼저 주어진 행렬 H에 대해서 공액벡터(conjugate vector)를 정의하자. 다음 식이 만족되면, 벡터들 $\boldsymbol{y}_m$과 $\boldsymbol{y}_n$은 H-공액(H-conjugate) 또는 H-직교(H-orthogonal)하다고 한다.

$$\boldsymbol{y}_m^t H\boldsymbol{y}_n = 0, \quad (n \neq m) \tag{6.3.15}$$

벡터 $\boldsymbol{x}(\in R^N)$에 관한 목적함수 $f(\boldsymbol{x})$의 최소점을 찾아가기 위해서, 먼저 초기점 $\boldsymbol{x}_0$와 N개 H-공액벡터들 $\boldsymbol{u}_0, \boldsymbol{u}_1, \cdots, \boldsymbol{u}_{N-1}$을 선택하고, 벡터 $\boldsymbol{u}_0$ 방향으로 목적함수를 최대한 감소시키는 점 $\boldsymbol{x}_1^* = \boldsymbol{x}_0 + \alpha_0\boldsymbol{u}_0$을 찾고, 벡터 $\boldsymbol{u}_1$ 방향으로 목적함수를 최대한 감소시키는 점 $\boldsymbol{x}_2^* = \boldsymbol{x}_1^* + \alpha_1\boldsymbol{u}_1$을 찾고, 이러한 방법을 반복해서 벡터 $\boldsymbol{u}_{N-1}$ 방향으로 목적함수를 최대한 감소시키는 점 $\boldsymbol{x}_N^* = \boldsymbol{x}_{N-1}^* + \alpha_{N-1}\boldsymbol{u}_{N-1}$을 찾는다. 다음 식들이 성립함은 자명하다.

$$\boldsymbol{x}_n^* = \boldsymbol{x}_0 + \sum_{j=0}^{n-1} \alpha_j\boldsymbol{u}_j, \quad (n = 0, 1, \cdots, N-1) \tag{6.3.16}$$

단, $\boldsymbol{x}_0^* = \boldsymbol{x}_0$라고 가정한다. 식 (6.3.14)에서 알 수 있듯이, 각 n에 대해서 그래디언트벡터 $\boldsymbol{g}_n \doteq \left.\frac{df}{d\boldsymbol{x}}\right|_{\boldsymbol{x}_n}$은 다음 식을 만족한다.

$$\boldsymbol{g}_n = H\boldsymbol{x}_n + \boldsymbol{c} \tag{6.3.17}$$

정의에서 알 수 있듯이, α_n은 α의 함수 $f(\boldsymbol{x}_n^* + \alpha u_n)$을 최소로 하는 상수이다. 즉, 다음 식이 성립한다.

$$\alpha_n = \arg\min_{\alpha} f\left(\boldsymbol{x}_n^* + \alpha\boldsymbol{u}_n\right) \tag{6.3.18}$$

다음 식들이 성립한다.

$$\begin{aligned}
&\frac{\partial}{\partial\alpha} f(\boldsymbol{x}_n^* + \alpha\boldsymbol{u}_n) \\
&= \frac{\partial}{\partial\alpha}\left\{\frac{1}{2}[\boldsymbol{x}_n^* + \alpha\boldsymbol{u}_n]^t H\,[\boldsymbol{x}_n^* + \alpha\boldsymbol{u}_n] + c^t[\boldsymbol{x}_n^* + \alpha\boldsymbol{u}_n]\right\} \\
&= \boldsymbol{u}_n^t\,\{H[\boldsymbol{x}_n^* + \alpha\boldsymbol{u}_n] + \boldsymbol{c}\}
\end{aligned} \tag{6.3.19}$$

식 (6.3.18)과 식 (6.3.19)에서 알 수 있듯이, 다음 식이 성립한다.

$$\boldsymbol{u}_n^t\,\{H[\boldsymbol{x}_n^* + \alpha\boldsymbol{u}_n] + \boldsymbol{c}\} = 0 \tag{6.3.20}$$

식 (6.3.20)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\alpha_n = -\frac{\boldsymbol{u}_n^t[H\boldsymbol{x}_n^* + \boldsymbol{c}]}{\boldsymbol{u}_n^t H\boldsymbol{u}_n} = -\frac{\boldsymbol{u}_n^t\boldsymbol{g}_n}{\boldsymbol{u}_n^t H\boldsymbol{u}_n} \tag{6.3.21}$$

그래디언트벡터의 성질에서 알 수 있듯이, 목적함수 $f(\boldsymbol{x})$는 다음 식을 만족하는 점 $\boldsymbol{x}^*$에서 최소값을 갖는다.

$$\boldsymbol{g}(\boldsymbol{x}^*) \doteq \left.\frac{df}{d\boldsymbol{x}}\right|_{\boldsymbol{x}^*} = H\boldsymbol{x}^* + \boldsymbol{c} = \boldsymbol{0} \tag{6.3.22}$$

지금부터는 목적함수 $f(\boldsymbol{x})$가 점 $\boldsymbol{x}_N^*$에서 최소값을 갖음을 증명하기로 하자. 즉, 식 $\boldsymbol{g}(\boldsymbol{x}_N^*) = \boldsymbol{0}$가 성립함을 증명하기로 하자. 다음 식들이 성립한다.

$$\begin{aligned}
\boldsymbol{u}_m^t H[\boldsymbol{x}_0 - \boldsymbol{x}_m^*] &= -\boldsymbol{u}_m^t H\left[\sum_{j=0}^{m-1}\alpha_j\boldsymbol{u}_j\right] \\
&= -\sum_{j=0}^{m-1}\alpha_j\boldsymbol{u}_m^t H\boldsymbol{u}_j = 0, \quad (m = 0, 1, \cdots, N-1)
\end{aligned} \tag{6.3.23}$$

여기서 첫 번째 등호는 식 (6.3.16)에 의해서, 세 번째 등호는 식 (6.3.15)에 의해서 성립한다. 식 (6.3.17)과 식 (6.3.23)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\boldsymbol{u}_m^t[H\boldsymbol{x}_0 + \boldsymbol{c} - \boldsymbol{g}_m^*] = 0, \quad (m = 0, 1, \cdots, N-1) \tag{6.3.24}$$

식 (6.3.21)과 식 (6.3.24)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\boldsymbol{u}_m^t[H\boldsymbol{x}_0 + \boldsymbol{c} + \alpha_m H\boldsymbol{u}_m] = 0, \quad (m = 0, 1, \cdots, N-1) \tag{6.3.25}$$

식 (6.3.15)와 식 (6.3.25)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\boldsymbol{u}_m^t[H\boldsymbol{x}_0+\boldsymbol{c}+\sum_{j=0}^{N-1}\alpha_j H\boldsymbol{u}_j]=0,\quad (m=0,1,\cdots,N-1) \tag{6.3.26}$$

벡터들 $\boldsymbol{u}_0,\boldsymbol{u}_1,\cdots,\boldsymbol{u}_{N-1}$은 기저를 이루므로, 다음 식들이 성립한다.

$$H\boldsymbol{x}_N^*+\boldsymbol{c}=H\left[\boldsymbol{x}_0+\sum_{j=0}^{N-1}\alpha_j\boldsymbol{u}_j\right]+\boldsymbol{c}=H\boldsymbol{x}_0+\boldsymbol{c}+\sum_{j=0}^{N-1}\alpha_j H\boldsymbol{u}_j=\boldsymbol{0} \tag{6.3.27}$$

여기서 첫 번째 등호는 식 (6.3.16)에 의해서, 그리고 세 번째 등호는 식 (6.3.26)에 의해서 성립한다.

지금까지 내용에서 알 수 있듯이, 각 $n(=0,1,\cdots,N-1)$에 대해서 그래디언트벡터 $\boldsymbol{g}_n$을 구하고 이를 사용해서 $f(\boldsymbol{x}_{n+1})$을 최소로 하는 α_n을 계산하는 과정을 반복하면, 목적함수 $f(\boldsymbol{x})$의 최소값에 이른다. 이러한 최적화법을 Powell의 공액방향법이라 부르기도 한다.

이제 남은 문제는 H-공액벡터들 $\boldsymbol{u}_0,\boldsymbol{u}_1,\cdots,\boldsymbol{u}_{N-1}$을 구하는 것이다. 행렬 H가 양정치 행렬이므로 이들의 고유벡터들 $\boldsymbol{v}_0,\boldsymbol{v}_1,\cdots,\boldsymbol{v}_{N-1}$ 은 서로 수직이다. 고유벡터 $\boldsymbol{v}_j$에 해당하는 고유값을 λ_j라고 하면, 서로 다른 m과 n에 대해서 다음 식들이 성립한다.

$$\boldsymbol{v}_m^t H\boldsymbol{v}_n=\boldsymbol{v}_m^t[\lambda_n\boldsymbol{v}_n]=\lambda_n\boldsymbol{v}_m^t\boldsymbol{v}_n=0 \tag{6.3.28}$$

즉, $\boldsymbol{v}_0,\boldsymbol{v}_1,\cdots,\boldsymbol{v}_{N-1}$은 H-공액벡터들이다. 공액벡터들을 구하는 다른 방법은 서로 다른 두 점들 $\boldsymbol{x}$와 $\boldsymbol{y}$에서 어떤 벡터 $\boldsymbol{u}$의 방향으로 목적함수를 최소화하는 점들 $\hat{\boldsymbol{x}}$와 $\hat{\boldsymbol{y}}$를 찾아서 그 차이를 구하는 것이다. 식 (6.3.20)에서 알 수 있듯이, 다음 식들을 만족하는 α_x와 α_y가 존재한다.

$$\boldsymbol{u}^t H\hat{\boldsymbol{x}}+\boldsymbol{u}^t\boldsymbol{c}=\boldsymbol{u}^t H[\boldsymbol{x}+\alpha_x\boldsymbol{u}]+\boldsymbol{u}^t\boldsymbol{c}=0 \tag{6.3.29}$$

$$\boldsymbol{u}^t H\hat{\boldsymbol{y}}+\boldsymbol{u}^t\boldsymbol{c}=\boldsymbol{u}^t H[\boldsymbol{y}+\alpha_y\boldsymbol{u}]+\boldsymbol{u}^t\boldsymbol{c}=0 \tag{6.3.30}$$

식 (6.3.29)와 식 (6.3.30)에서 알 수 있듯이, 다음 식이 성립한다.

$$\boldsymbol{u}^t H[\hat{\boldsymbol{x}}-\hat{\boldsymbol{y}}]=0 \tag{6.3.31}$$

식 (6.3.31)에서 알 수 있듯이, 벡터들 $\boldsymbol{u}$와 $\hat{\boldsymbol{x}}-\hat{\boldsymbol{y}}$는 서로 H-공액이다. 따라서, 식 (6.3.31)을 이용해서, H-공액벡터들을 구할 수 있다.

예제 6.3.15 다음 2차함수를 살펴보자.

$$f(x_1, x_2) = x_1^2 + x_2^2 - x_1 x_2 - 4x_1 - x_2 \tag{1}$$

이 함수는 점 $(3, 2)$에서 최소값 -7을 갖음은 확실하다. Powell의 공액방향법을 적용해서 최소값을 구하기 위해서, 다음 MATLAB 프로그램 ConjugateDirectionalMethod101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: ConjugateDirectionalMethod101.m
%  Powell's Conjugate Directional Method
%  Find the minimum of a multivariable function
%  Programmed by CBS
% ------------------------------------------------------------------------
close all, clear all
imax = 30; kmax = 30; nmax1 = 30; warning = 0; alpha0 = 10;
N = 2; u = eye(N);
myfun = @(x) x(1)*(x(1)-4-x(2)) + x(2)*(x(2)-1)
% myfun = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2 % banana function
xmat = zeros(imax,N); fvec = zeros(1,imax); fdiff = zeros(1,imax);
fvec(1) = myfun(xmat(1,:)); fdiff(1) = fvec(1);
    for ii = 2:1:imax
        xx0 = xmat(ii-1,:); xx = xx0;
        ff0 = myfun(xx0); ff = ff0;
        for kk = 1:1:kmax
            for nn = 0:1:N
                m = mod(nn-1,N) + 1;
                alpha = alpha0;
                ff1 = myfun(xx+alpha*2*u(m,:));
            for nn1 = 1:1:nmax1
                ff2 = ff1;
                ff1 = myfun(xx+alpha*u(m,:));
                if (ff0>ff1+1.0e-8)&(ff1<ff2-1.0e-8)
                    deno = 4*ff1-2*ff0-2*ff2;
                    nume = deno-ff0+ff2;
                    alpha = alpha*nume/deno;
                    xx = xx + alpha*u(m,:);
                    ff = myfun(xx);
                    break
                elseif nn1 == nmax1/2
                    alpha = -alpha0;
                    ff1 = myfun(xx+alpha*2*u(m,:));
                else
                    alpha = alpha/2;
                end
            end
            xx0 = xx;
            ff0 = ff;
            if nn1 >= nmax1
                warning = warning +1;
            else
                warning = 0;
            end
            if nn < 0
                xx00 = xx;
            end
```

```
         end
            if (warning>=2)|(norm(xx-xx0)<1.0e-8)&(abs(ff-ff0)<1.0e-8)
                break;
            end
            uN = xx-xx00;
            u = [ u(2:N,:) ; uN/(norm(uNew)+1.0e-8) ];
        end
        xmat(ii,:) = xx;
        fvec(ii) = ff;
        fdiff(ii) = fvec(ii)-fvec(ii-1);
    if abs(fdiff(ii)) < 1.0e-8
        break
    end
end
ii
format short e
fdiff(ii) = 1.0e-08;
xfout = [ xmat(1:ii,:), fvec(1:ii)', fdiff(1:ii)' ]
% Plotting
xp = [ xmat(1,1) xmat(2,1) xmat(2,1) xmat(3,1) xmat(3,1) xmat(4,1) ...
       xmat(4,1) xmat(5,1) xmat(5,1) ];
yp = [ xmat(1,2) xmat(1,2) xmat(2,2) xmat(2,2) xmat(3,2) xmat(3,2) ...
       xmat(4,2) xmat(4,2) xmat(5,2) ];
plot(xp,yp,'r-','LineWidth',2.5)
set(gca,'fontsize',12,'fontweigh','bold')
axis([ -1 4 -1 3 ])
grid on
xlabel('\bf \it x','fontsize',12)
ylabel('\bf \it y','fontsize',12,'rotation',0)
saveas(gcf,'ConjugateDirectionalMethod101','jpg')
save('ConjugateDirectionalMethod101.txt','xfout','-ascii')
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 ConjugateDirectinalMethod101.m을 실행하면, 다음 결과물이 출력된다.

```
ii = 9


     x(1)         x(2)         f(x)    Delta f(x)
       0            0            0            0
2.2500e+000  1.6250e+000 -6.5781e+000 -6.5781e+000
2.8125e+000  1.9063e+000 -6.9736e+000 -3.9551e-001
2.9531e+000  1.9766e+000 -6.9984e+000 -2.4719e-002
2.9883e+000  1.9941e+000 -6.9999e+000 -1.5450e-003
2.9971e+000  1.9985e+000 -7.0000e+000 -9.6560e-005
2.9993e+000  1.9996e+000 -7.0000e+000 -6.0350e-006
2.9998e+000  1.9996e+000 -7.0000e+000 -3.0175e-007
2.9998e+000  1.9996e+000 -7.0000e+000  1.0000e-008
```

이 결과물에서 알 수 있듯이, 제8번째 반복에서 다음 식이 성립한다.

$$|f(\boldsymbol{x}_8) - f(\boldsymbol{x}_7)| < 10^{-8} \tag{1}$$

여기서는 H-공액벡터들로 $u_1 = [1, 0]$과 $u_2 = [0, 1]$를 사용한다. 따라서, 초기점 $[0, 0]$을 출발해서 X축에 평행하게 이동하여 점 $[2.25, 0]$에 도달하고, 그 점에서 방향을 좌측으로 90도 회전해서 Y축에 평행하게 이동하여 점 $[2.23, 1.63]$에 도달하고, 그 점에서 우측으로 90도 회전해서 X축에 평행하게 이동하여, 점 $[2.81, 1.63]$에 도달하고, 그 점에서 방향을 좌측으로 90도 회전해서 Y축에 평행하게 이동하여 점 $[2.81, 1.91]$에 도달한다. 같은 과정을 반복해서 점 $[3.00, 2.00]$에 가까이 도달하면, 알고리즘의 실행을 멈춘다. 이 MATLAB프로그램을 수행하면, 이 점들의 움직임을 그린 그림 6.3.9을 출력한다. ■

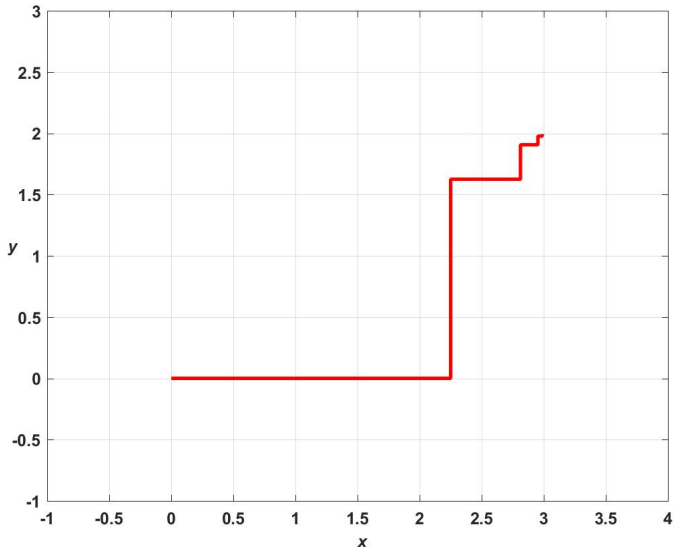


그림 6.3.9. 공액방향법

예제 6.3.16 Python을 사용해서 예제 6.3.15를 다시 다루기 위해서, 다음 Python프로그램 ConjugateDirectionalMethod101.Py를 실행해 보자.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Mar 10 05:10:50 2017

@author: CBS
"""

import numpy as np
```

```
import matplotlib.pyplot as plt

imax = 30; kmax = 30; nmax1 = 30; warning = 0; alpha0 = 10;
N = 2; u = np.identity(N);
myfun = lambda x: x[0]*(x[0]-4.-x[1]) + x[1]*(x[1]-1.)
xmat = np.zeros((imax,N)); fvec = np.zeros(imax); fdiff = np.zeros(imax);
fvec[0] = myfun(xmat[0,:]); fdiff[0] = fvec[0];

for ii in range(1,imax):     # ii
    xx0 = xmat[ii-1,:]; xx = xx0;
    ff0 = myfun(xx0);   ff = ff0;
    for kk in range(0,kmax):     # kk
        for nn in range(-1,N):     # nn
            m = np.mod(nn,N);
            alpha = alpha0;
            ff1 = myfun(xx+alpha*2*u[m,:]);
            for nn1 in range(0,nmax1):     # nn1
                ff2 = ff1;
                ff1 = myfun(xx+alpha*u[m,:]);
                if (ff0>ff1+1.0e-8)&(ff1<ff2-1.0e-8): # i101
                    deno = 4*ff1-2*ff0-2*ff2;
                    nume = deno-ff0+ff2;
                    alpha = alpha*nume/deno;
                    xx = xx + alpha*u[m,:];
                    ff = myfun(xx);
                    break
                elif nn1 == nmax1/2:
                    alpha = -alpha0;
                    ff1 = myfun(xx+alpha*2*u[m,:]);
                else:
                    alpha = alpha/2;
                # end i101F
            # end  nn1
            xx0 = xx;
            ff0 = ff;
            if nn1 >= nmax1:  # i102
               warning = warning +1;
            else:
               warning = 0;
            # end     i102F
            if nn < 0:   # i103
               xx00 = xx;
            # end     i103F
        # end nn
        if (warning>=2)|(np.linalg.norm(xx-xx0)<1.0e-8)&(abs(ff-ff0)<1.0e-8)
    : # i104
            break;
        # end i 104F
        uN = xx-xx00;
        u = np.append( u[1:(N-1),:],uN/(np.linalg.norm(uNew)+1.0e-8) );
    # end kk
    xmat[ii,:] = xx;
    fvec[ii] = ff;
    fdiff[ii] = fvec[ii]-fvec[ii-1];
    if abs(fdiff[ii]) < 1.0e-8:  # i105
       break
    # end     i105F
# end  ii
print(ii)
fdiff[ii] = 1.0e-08;
print(xmat[0:ii+1,:]), print(fvec[0:ii+1]), print(fdiff[0:ii+1])
```

```
69
70 # Plotting
71 fig = plt.figure()
72 xp = [ xmat[0,0], xmat[1,0], xmat[1,0], xmat[2,0], xmat[2,0], xmat[3,0], \
73        xmat[3,0], xmat[4,0], xmat[4,0] ];
74 yp = [ xmat[0,1], xmat[0,1], xmat[1,1], xmat[1,1], xmat[2,1], xmat[2,1], \
75        xmat[3,1], xmat[3,1], xmat[4,1] ];
76 plt.plot(xp,yp,'r-',lw=2.5)
77 plt.axis([ -1, 4, -1, 3 ])
78 plt.grid()
79 plt.xlabel('x',fontsize=12); plt.ylabel('y',fontsize=12)
80
81 plt.show()
82 fig.savefig('ConjugateDirectionalMethod101Py.png')
83
84 # End of program
```

이 Python프로그램을 수행한 결과는 예제 6.3.15의 결과와 같다. ■

예제 6.3.17 Python을 사용해서 예제 6.3.15를 다시 다루기 위해서, 다음 Python프로그램 ConjugateDirectionalMethod102.Py를 실행해 보자.

```
1 from scipy import optimize
2
3 myfun = lambda x: x[0]*(x[0]-4.-x[1]) + x[1]*(x[1]-1.)
4 xopt = optimize.fmin_cg(myfun, [0,0])
5 print('xopt = ', xopt, ',  fval = ', myfun(xopt))
6
7 # End of program
```

이 Python프로그램을 수행한 결과는 다음과 같다.

```
Optimization terminated successfully.
         Current function value: -7.000000
         Iterations: 2
         Function evaluations: 20
         Gradient evaluations: 5


xopt =  [ 2.99999996  1.99999995] ,  fval =  -7.0
```

■

예제 6.3.18 Python을 사용해서 예제 6.3.15를 다시 다루기 위해서, 다음 Python프로그램 ConjugateDirectionalMethod103.Py를 실행해 보자.

```
import numpy as np
from scipy import optimize

def f(x):
    return x[0]*(x[0]-4.0-x[1]) + x[1]*(x[1]-1.0)
def df(x):
    return np.array( (2.0*x[0]-4-x[1], -x[0]+2.0*x[1]-1) )

xopt = optimize.fmin_cg(f, [2, 2], fprime=df)
print('xopt = ', xopt, ',  fval = ', f(xopt))

# End of program
```

이 Python프로그램에서는 Python함수 scipy.optimize.fmin_cg에 그래디언트벡트를 지정하였다. 이 Python프로그램을 수행한 결과는 다음과 같다.

```
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: -6.999996
         Iterations: 4
         Function evaluations: 19
         Gradient evaluations: 7


xopt =  [ 3.00219878  2.00186552] ,  fval =  -6.99999578707
```

■

제6.4절 부등식제약조건 하에서 최적화

이 절에서는 제약조건이 등식 뿐 아니라 부등식도 포함하는 경우를 살펴보자. 이러한 부등식제약조건 하에서 최적화문제의 해가 만족해야 하는 필요조건을 제시한 것은 1951년 Kuhn & Tucker [35]이다. 그러나, 이 필요조건은 1939년 Karush [33]가 석사논문에서 제시한 것임이 훗날 알려졌다. 따라서, 최근에는 이 필요조건을 Karush-Kuhn-Tucker조건 또는 KKT조건이라 부른다.

다음과 같은 최적화문제를 살펴보자.

$$\text{Minimize } f(\boldsymbol{x}) \tag{6.4.1}$$

subject to

$$g_i(\boldsymbol{x}) \le 0, \quad (i = 1, 2, \cdots, l), \tag{6.4.2}$$

$$g_j(\boldsymbol{x}) = 0, \quad (j = l+1, l+2, \cdots, m). \tag{6.4.3}$$

만약 $\boldsymbol{x}$가 제약조건들 (6.4.2)와 (6.4.3)을 만족하면, 점 $\boldsymbol{x}$가 실현가능(feasible)이라 한다. 만약 점 $\boldsymbol{x}$가 실현가능하고 또한 $g_i(\boldsymbol{x}) = 0$라면, 제i번째 부등식제약조건 $g_i(\boldsymbol{x}) \le 0$는 점 $\boldsymbol{x}$에서 유효하다고(active) 한다. 만약 점 $\boldsymbol{x}$가 실현가능하고 또한 $g_i(\boldsymbol{x}) < 0$라면, 제i번째 부등식제약조건 $g_i(\boldsymbol{x}) \le 0$는 점 $\boldsymbol{x}$에서 유효하지않다고(inactive) 한다. 제약조건들 (6.4.2)와 (6.4.3)는 상충되지 않아야 한다. 즉, 제약조건들에는 자격이(qualifications) 필요하다. 이를 정칙조건들(regularity)이라 부른다. 다양한 정칙조건들이 있는데, 이 절에서는 Mangasarian–Fromovitz 제약조건(MFCQ) 만을 다루기로 하자. 점 $\boldsymbol{x}$에서 다음 집합들을 정의하자.

$$A(\boldsymbol{x}) \doteq \{i \,|\, g_i(\boldsymbol{x}) = 0,\ i = 1, 2, \cdots, l\} \tag{6.4.4}$$

$$A^C(\boldsymbol{x}) \doteq \{1, 2, \cdots, l\} \setminus A(\boldsymbol{x}) \tag{6.4.5}$$

$$B(\boldsymbol{x}) \doteq \left\{ \boldsymbol{y} \in \mathbb{R}^k \,\middle|\, \nabla g_i(\boldsymbol{x})^t \boldsymbol{y} = 0,\ i \in A(\boldsymbol{x}) \cup \{l+1, l+2, \cdots, m\} \right\} \tag{6.4.6}$$

만약 $\left\{\nabla g_i(\boldsymbol{x}) \,\middle|\, i \in A(\boldsymbol{x})\right\} \bigcup \left\{\nabla g_j(\boldsymbol{x}) \,\middle|\, j = l+1, l+2, \cdots, m\right\}$가 선형독립이면, 점 $\boldsymbol{x}$는 Mangasarian–Fromovitz제약조건을 만족한다고 한다. 이 제약조건이 만족되면, 점 $\boldsymbol{x}$가 정칙이라고(regular) 한다.

다음 정리는 식 (6.4.1)∼ 식 (6.4.3)의 최적화문제의 해에 관한 것이다. 여기서는 이 정리의 증명을 하지 않는다. 이 정리의 증명과 더불어 좀 더 자세한 내용을 알고자 하는 독자는 **IM&F9** [4]의 제2.4절을 참조하라.

정리 6.4.1: KKT조건

함수들 $f(\boldsymbol{x})$와 $g_i(\boldsymbol{x})$, $(i = 1, 2, \cdots, m)$가 2회연속미분가능(twice continuously differentiable)하다고 가정하자. 정칙인 실행가능점 $\boldsymbol{x}^*$에서 국소최적해를 갖기 위한 조건은 다음과 같다.

(1차 필요조건) 다음 식들을 만족하는 Lagrange승수벡터 $\boldsymbol{\lambda}^* \doteq [\lambda_1^*, \lambda_2^*, \cdots, \lambda_m^*]^t$가 존재한다.

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\boldsymbol{x}^*) = \boldsymbol{0}$$

$$\lambda_i^* \ge 0, \quad (i = 1, 2, \cdots, l)$$

$$\lambda_i^* = 0, \quad \left(i \in A^C(\boldsymbol{x}^*)\right)$$

(2차 필요조건) 임의의 $\boldsymbol{y} \in B(\boldsymbol{x}^*)$에 대해서, 다음 부등식이 성립한다.

$$\boldsymbol{y}^t \left[\nabla^2 f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla^2 g_i(\boldsymbol{x}^*) \right] \boldsymbol{y} \geq 0$$

즉, 행렬 $\nabla^2 f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla^2 g_i(\boldsymbol{x}^*)$는 벡터집합 $B(\boldsymbol{x}^*)$ 상에서 반양정치이다.

정리 6.4.1의 1차 필요조건의 제3번째 조건을 다음과 같이 쓸 수 있다.

$$\lambda_i^* g_i(\boldsymbol{x}^*) = 0, \quad (i = 1, 2, \cdots, l)) \tag{6.4.7}$$

식 (6.4.7)을 상보성조건(complementarity condition)이라 부른다.

예제 6.4.1 다음과 같은 최적화문제를 살펴보자.

$$\text{Minimize} f(\boldsymbol{x}) = 2[x_1 - 1]^2 + [x_1 - 1][x_2 - 1] + [x_2 - 1]^2 \tag{1}$$

subject to

$$g_1(\boldsymbol{x}) = x_1^2 + x_2^2 - 1 \leq 0, \tag{2}$$

$$g_2(\boldsymbol{x}) = -x_1 \leq 0, \tag{3}$$

$$g_3(\boldsymbol{x}) = -x_2 \leq 0. \tag{4}$$

만약 제약조건이 없다면, 목적함수 $f(\boldsymbol{x})$는 점 $(1, 1)$에서 최소값 0를 갖음이 확실하다. 그러나, 이 점은 실현가능영역에 속하지 않는다.

이 최적화문제를 풀기 위해서, 다음 MATLAB프로그램 KKT101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: KKT101.m
%  Graph 1 for Karush-Kuhn-Tucker condition
%  Optimization with inequality constraints
% ------------------------------------------------------------------
clear all, close all
% Feasible set
x1 = linspace(0,1,201);
x2 = sqrt(1-x1.^2);
x1a = [ 0 x1 ];
x2a = [ 0 x2 ];
% Object function
tt = (0:1:360)'*(2*pi/360);
for kdum=1:1:3
    k =  -0.1 + 0.2*kdum;
    yf1(:,kdum) = sqrt(k)*2/sqrt(7)*cos(tt)+1;
```

```
    yf2(:,kdum) = sqrt(k)*sin(tt) - 1/2*(yf1(:,kdum)-1)+1;
end
yf1(end,:), yf2(end,:)
% Plotting
fill(x1a,x2a,'g')
set(gca,'fontsize',11,'fontweigh','bold')
hold on
plot(yf1(:,1),yf2(:,1),'k:',yf1(:,2),yf2(:,2),'r--', ...
     yf1(:,3),yf2(:,3),'b-.','LineWidth',2)
legend('\bf Feasible','\bf k = 0.1','\bf k = 0.3', ...
       '\bf k = 0.5','location','NE')
plot(x1a,x2a,'k-','LineWidth',2)
plot([0 0], [ -3 3 ],'k','LineWidth',2);
plot([-3 3 ], [0 0 ],'k','LineWidth',2);
axis equal
axis([-0.3 2.2 -0.3 2.2])
hold off
text(0.8-0.02,0.7-0.02,'\bf P')
xlabel('\bf x_1')
ylabel('\bf x_2','rotation',0)
saveas(gcf,'KKT101','jpg')
% KKT condition
format long
KKTcondi = @(x) ((4*x(1)+x(2)-5)*x(2) - (x(1)+2*x(2)-3)*x(1))^2 ...
                 + (x(1)^2+x(2)^2-1)^2;
[xstar fval2 exitflag] = fminsearch(KKTcondi,[1,1])
lambdastar1 = -(4*xstar(1)+xstar(2)-5)/(2*xstar(1))
g1 = xstar(1)^2+xstar(2)^2-1
H = [ 4 1 ; 1 2 ] + lambdastar1*[ 2 0 ; 0 2 ]
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 KKT101.m을 실행하면, 그림 6.4.1을 출력한다. 그림 6.4.1에서 녹색으로 칠해진 부채꼴이 실현가능영역이다. 흑색 점선으로 그려진 타원은 식 $f(\boldsymbol{x}) = 0.1$을 나타내고, 적색 긴점선으로 그려진 타원은 식 $f(\boldsymbol{x}) = 0.3$를 나타내고, 청색 반점선으로 그려진 타원은 식 $f(\boldsymbol{x}) = 0.5$를 나타낸다. 따라서, 이 최적화문제의 해 $\boldsymbol{x}^*$는 실현가능영역과 곡선 $f(\boldsymbol{x}) = 0.3$이 만나는 점 $\mathbf{P}$이다.

상보성조건에 의해서 다음 식들이 성립함을 알 수 있다.

$$\lambda_2^* = 0, \quad \lambda_3^* = 0 \tag{5}$$

정리 6.4.1에서 기술한 KKT의 1차 필요조건에서 알 수 있듯이, 다음 식들이 성립한다.

$$\nabla f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\boldsymbol{x}^*) = \begin{bmatrix} 4x_1 + x_2 - 5 \\ x_1 + 2x_2 - 3 \end{bmatrix} + \lambda_1 \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} = \mathbf{0} \tag{6}$$

$$x_1^2 + x_2^2 - 1 = 0 \tag{7}$$

연립방정식 (6)과 (7)을 풀면, 다음 해를 얻는다.

$$x_1^* = 0.785407, \quad x_2^* = 0.618927, \quad \lambda_1^* = 0.789046 \tag{8}$$

다음 식들이 성립함을 확인할 수 있다.

$$g_1(\boldsymbol{x}^*) = -6.491108 \times 10^{-5} \approx 0 \tag{9}$$

따라서, 상보성조건 (6.4.7)이 성립함을 확인할 수 있다. 또한, 다음 식들이 성립한다.

$$\begin{aligned} &\nabla^2 f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla^2 g_i(\boldsymbol{x}^*) \\ &= \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix} + \lambda_1^* \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 5.578092 & 1 \\ 1 & 3.578092 \end{bmatrix} \end{aligned} \tag{10}$$

식 (10)에서 계산된 행렬이 양정치임을 쉽게 알 수 있다. ■

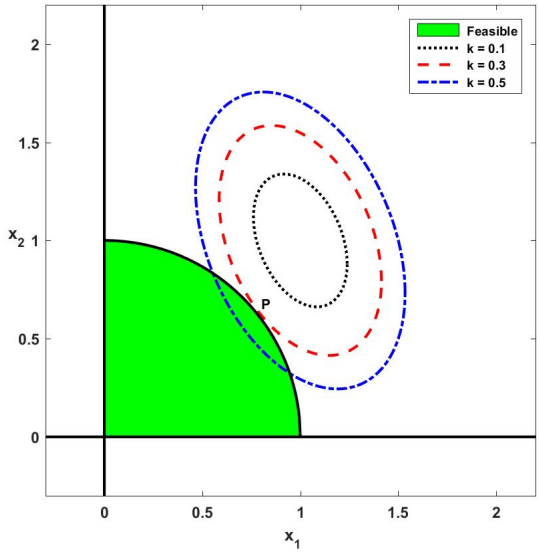


그림 6.4.1. 부등식제약조건 I

예제 6.4.2 Python을 사용해서 예제 6.4.1을 다시 다루기 위해서, 다음 Python프로그램 KKT101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize
# Feasible set
x1 = np.linspace(0,1,201);
x2 = np.sqrt(1-x1**2);
x1a = np.append(0,x1);
x2a = np.append(0,x2);
# Object function
tt = (2*np.pi/360)*np.arange(361);
yf1 = np.zeros((361,3));
yf2 = np.zeros((361,3));
print(yf2)
for kdum in range(0,3):
    k =  -0.1 + 0.2*(kdum+1);
    yf1[:,kdum] = np.sqrt(k)*2/np.sqrt(7)*np.cos(tt)+1;
    yf2[:,kdum] = np.sqrt(k)*np.sin(tt) - 1/2*(yf1[:,kdum]-1.0)+1.0;

# Plotting
fig = plt.figure()
plt.fill_between(x1a, 0, x2a, facecolor='green', label='Feasible')
plt.plot(yf1[:,0],yf2[:,0],'k:',lw=2.0,label='k=0.1')
plt.plot(yf1[:,1],yf2[:,1],'r--',lw=2.0,label='k=0.2')
plt.plot(yf1[:,2],yf2[:,2],'b-.',lw=2.0,label='k=0.3')
plt.plot(x1a,x2a,'k-',lw=2)
plt.plot([0, 0], [ -3, 3 ],'k',lw=2);
plt.plot([-3, 3 ], [0, 0 ],'k',lw=2);
plt.legend(loc='upper right', numpoints=1)
plt.axis('equal')
plt.axis([-0.3, 2.2, -0.3, 2.2])
plt.text(0.8-0.02,0.7-0.02,'P')
plt.xlabel('\bf x_1')
plt.ylabel('\bf x_2')
plt.show()
fig.savefig('KKT101Py.png')

# KKT condition
KKTcondi = lambda x: ((4*x[0]+x[1]-5)*x[1] - (x[0]+2*x[1]-3)*x[0])**2 \
                    + (x[0]**2+x[1]**2-1)**2;
xstar = scipy.optimize.fmin(func=KKTcondi,x0=[1,1])
print('xstar = ', xstar)
fval2 = KKTcondi(xstar)
print('fmin =', fval2)
lambdastar1 = -(4*xstar[0]+xstar[1]-5.0)/(2*xstar[0])
print('lambda = ', lambdastar1)
g1 = xstar[0]**2+xstar[1]**2-1.0
print('g1 = ', g1)
H = np.array([[ 4, 1], [1, 2 ]]) + lambdastar1*np.array([[ 2, 0 ], [0, 2 ]])
print('H = ', H)
# End of program
```

이 Python프로그램을 수행한 결과는 예제 6.4.1의 결과와 같다. ■

예제 6.4.3 다음과 같은 최적화문제를 살펴보자.

$$\text{Minimize} f(\boldsymbol{x}) = 2[x_1+1]^2 + [x_1+1][x_2-2] + [x_2-2]^2 \tag{1}$$

subject to

$$g_1(\boldsymbol{x}) = x_1^2 + x_2^2 - 1 \le 0, \tag{2}$$

$$g_2(\boldsymbol{x}) = -x_1 \le 0, \tag{3}$$

$$g_3(\boldsymbol{x}) = -x_2 \le 0. \tag{4}$$

만약 제약조건이 없다면, 목적함수 $f(\boldsymbol{x})$는 점 $(1,1)$에서 최소값 0를 갖음이 확실하다. 그러나, 이 점은 실현가능영역에 속하지 않는다.

이 최적화문제를 풀기 위해서, 다음 MATLAB프로그램 KKT102.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: KKT102.m
%  Graph 2 for Karush-Kuhn-Tucker condition
%  Optimization with inequality constraints
%  Object: f = 2(x_1+1)^2 +(x_2-2)^2 + (x_1+1)(x_2-2)
% ------------------------------------------------------------------------
clear all, close all
% Feasible set
x1 = linspace(0,1,201);
x2 = sqrt(1-x1.^2);
x1a = [ 0 x1 ];
x2a = [ 0 x2 ];
% Object function
tt = (0:1:360)'*(2*pi/360);
for kdum=1:1:3
    k = kdum;
    yf1(:,kdum) = sqrt(k)*2/sqrt(7)*cos(tt)-1;
    yf2(:,kdum) = sqrt(k)*sin(tt) - 1/2*(yf1(:,kdum)+1)+2;
end
% Plotting
fill(x1a,x2a,'g')
set(gca,'fontsize',11,'fontweigh','bold')
hold on
plot(yf1(:,1),yf2(:,1),'k:',yf1(:,2),yf2(:,2),'r--', ...
     yf1(:,3),yf2(:,3),'b-.','LineWidth',2)
legend('\bf Feasible','\bf k = 1','\bf k = 2', ...
       '\bf k = 3','location','NE')
plot(x1a,x2a,'k-','LineWidth',2)
plot([0 0], [ -5 5 ],'k','LineWidth',2);
plot([ -5 5 ], [0 0 ],'k','LineWidth',2);
axis equal
axis([-3 3 -1 5 ])
hold off
text(0.8-0.02,0.7-0.02,'\bf P')
xlabel('\bf x_1')
ylabel('\bf x_2','rotation',0)
saveas(gcf,'KKT102','jpg')
% KKT condition
```

```
format long
KKTcondi = @(x) ((4*x(1)+x(2)-5)*x(2) - (x(1)+2*x(2)-3)*x(1))^2 ...
                  + (x(1)^2+x(2)^2-1)^2;
[xstar fval2 exitflag] = fminsearch(KKTcondi,[1,1])
lambdastar1 = -(4*xstar(1)+xstar(2)-5)/(2*xstar(1))
g1 = xstar(1)^2+xstar(2)^2-1
H = [ 4 1 ; 1 2 ] + lambdastar1*[ 2 0 ; 0 2 ]
% End of program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 KKT102.m을 실행하면, 그림 6.4.2를 출력한다. 그림 6.4.2에서 녹색으로 칠해진 부채꼴이 실현가능영역이다. 흑색 점선으로 그려진 타원은 식 $f(\boldsymbol{x}) = 1$을 나타내고, 적색 긴점선으로 그려진 타원은 식 $f(\boldsymbol{x}) = 2$를 나타내고, 청색 반점선으로 그려진 타원은 식 $f(\boldsymbol{x}) = 3$을 나타낸다. 따라서, 이 최적화문제의 해 $\boldsymbol{x}^*$는 실현가능영역과 곡선 $f(\boldsymbol{x}) = 2$가 만나는 점 $\mathbf{P}$이다.

상보성조건에 의해서 다음 식이 성립함을 알 수 있다.

$$\lambda_3^* = 0 \tag{5}$$

정리 6.4.1에서 기술한 KKT의 1차 필요조건에서 알 수 있듯이, 다음 식들이 성립한다.

$$\nabla f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\boldsymbol{x}^*) = \begin{bmatrix} 4x_1 + x_2 + 2 \\ x_1 + 2x_2 - 3 \end{bmatrix} + \lambda_1 \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} + \lambda_2 \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \mathbf{0} \tag{6}$$

$$x_1^2 + x_2^2 - 1 = 0 \tag{7}$$

$$x_1 = 0 \tag{8}$$

식 (6)∼식 (8)을 풀면, 다음 해를 얻는다.

$$x_2^* = 1, \quad \lambda_1^* = 0.5, \quad \lambda_2^* = 3 \tag{9}$$

다음 식들이 성립함을 확인할 수 있다.

$$g_1(\boldsymbol{x}^*) = 0, \quad g_2(\boldsymbol{x}^*) = 0 \tag{10}$$

따라서, 상보성조건 (6.4.7)이 성립함을 확인할 수 있다. 또한, 다음 식들이 성립한다.

$$\nabla^2 f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla^2 g_i(\boldsymbol{x}^*)$$
$$= \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix} + \lambda_1^* \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} + \lambda_2^* \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 5 & 1 \\ 1 & 3 \end{bmatrix} \tag{11}$$

식 (11)에서 계산된 행렬이 양정치임을 쉽게 알 수 있다. ■

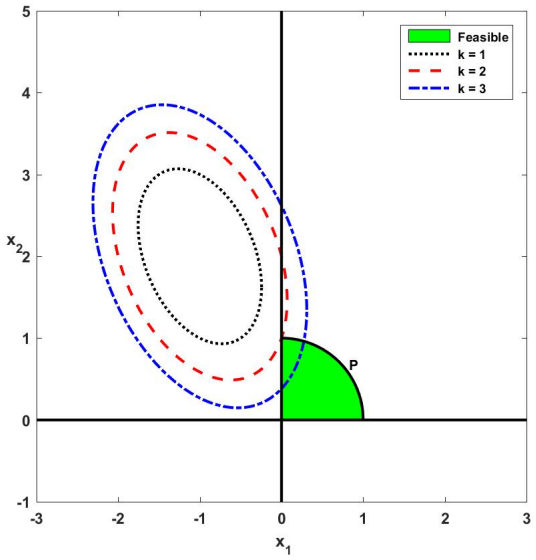


그림 6.4.2. 부등식제약조건 II

예제 6.4.4 Python을 사용해서 예제 6.4.3를 다시 다루기 위해서, 다음 Python프로그램 KKT102.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize
# Feasible set
x1 = np.linspace(0,1,201);
x2 = np.sqrt(1-x1**2);
x1a = np.append(0,x1);
x2a = np.append(0,x2);
# Object function
tt = (2*np.pi/360)*np.arange(361);
yf1 = np.zeros((361,3));
```

```
yf2 = np.zeros((361,3));
print(yf2)
for kdum in range(0,3):
    k =  kdum+1;
    yf1[:,kdum] = np.sqrt(k)*2/np.sqrt(7)*np.cos(tt)-1.0;
    yf2[:,kdum] = np.sqrt(k)*np.sin(tt) - 1/2*(yf1[:,kdum]+1.0)+2.0;

# Plotting
fig = plt.figure()
plt.fill_between(x1a, 0, x2a, facecolor='green', label='Feasible')
plt.plot(yf1[:,0],yf2[:,0],'k:',lw=2.0,label='k=0.1')
plt.plot(yf1[:,1],yf2[:,1],'r--',lw=2.0,label='k=0.2')
plt.plot(yf1[:,2],yf2[:,2],'b-.',lw=2.0,label='k=0.3')
plt.plot(x1a,x2a,'k-',lw=2)
plt.plot([0, 0], [ -5, 5 ],'k',lw=2);
plt.plot([-5, 5 ], [0, 0 ],'k',lw=2);
plt.legend(loc='upper right', numpoints=1)
plt.axis('equal')
plt.axis([-3.0, 3.0, -1.0, 5.0 ])
plt.text(0.8-0.02,0.7-0.02,'P')
plt.xlabel('\bf x_1')
plt.ylabel('\bf x_2')
plt.show()
fig.savefig('KKT102Py.png')

# KKT condition
KKTcondi = lambda x: ((4*x[0]+x[1]-5)*x[1] - (x[0]+2*x[1]-3)*x[0])**2 \
                     + (x[0]**2+x[1]**2-1)**2;
xstar = scipy.optimize.fmin(func=KKTcondi,x0=[1,1])
print('xstar = ', xstar)
fval2 = KKTcondi(xstar)
print('fmin =', fval2)
lambdastar1 = -(4*xstar[0]+xstar[1]-5.0)/(2*xstar[0])
print('lambda = ', lambdastar1)
g1 = xstar[0]**2+xstar[1]**2-1.0
print('g1 = ', g1)
H = np.array([[ 4, 1], [1, 2 ]]) + lambdastar1*np.array([[ 2, 0 ], [0, 2 ]])
print('H = ', H)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 6.4.3의 결과와 같다. ■

예제 6.4.5 다음과 같은 최적화문제를 살펴보자.

$$\text{Minimize} f(\boldsymbol{x}) = 2[x_1 - 1]^2 + [x_1 - 1][x_2 + 0.5] + [x_2 + 0.5]^2 \tag{1}$$

subject to

$$g_1(\boldsymbol{x}) = x_1^2 + x_2^2 - 1 \leq 0, \tag{2}$$

$$g_2(\boldsymbol{x}) = -x_1 \leq 0, \tag{3}$$

$$g_3(\boldsymbol{x}) = -x_2 \leq 0. \tag{4}$$

만약 제약조건이 없다면, 목적함수 $f(\boldsymbol{x})$는 점 $(1, -0.5)$에서 최소값 0를 갖음이 확실하다. 그러나, 이 점은 실현가능영역에 속하지 않는다.

이 최적화문제를 풀기 위해서, 다음 MATLAB프로그램 KKT103.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: KKT103.m
%  Graph 3 for Karush-Kuhn-Tucker condition
%  Optimization with inequality constraints
%  Object: f = 2(x_1-1)^2 +(x_2+0.5)^2 + (x_1-1)(x_2+0.5)
% ------------------------------------------------------------------------
clear all, close all
% Feasible set
x1 = linspace(0,1,201);
x2 = sqrt(1-x1.^2);
x1a = [ 0 x1 ];
x2a = [ 0 x2 ];
% Object function
tt = (0:1:360)'*(2*pi/360);
for kdum=1:1:3
    k = 0.105*kdum;
    yf1(:,kdum) = sqrt(k)*2/sqrt(7)*cos(tt)+1;
    yf2(:,kdum) = sqrt(k)*sin(tt) - 1/2*(yf1(:,kdum)-1)-0.5;
end
% Plotting
fill(x1a,x2a,'g')
set(gca,'fontsize',11,'fontweigh','bold')
hold on
plot(yf1(:,1),yf2(:,1),'k:',yf1(:,2),yf2(:,2),'r--', ...
     yf1(:,3),yf2(:,3),'b-.','LineWidth',2)
legend('\bf Feasible','\bf k = 0.105','\bf k = 0.210', ...
       '\bf k = 0.315','location','SW')
plot(x1a,x2a,'k-','LineWidth',2)
plot([0 0], [ -5 5 ],'k','LineWidth',2);
plot([ -5 5 ], [0 0 ],'k','LineWidth',2);
axis equal
axis([-1 2 -1.5 1.5 ])
hold off
text(0.8-0.02,0.7-0.02,'\bf P')
xlabel('\bf x_1')
ylabel('\bf x_2','rotation',0)
saveas(gcf,'KKT103','jpg')
% KKT condition
format long
KKTcondi = @(x) ((4*x(1)+x(2)-5)*x(2) - (x(1)+2*x(2)-3)*x(1))^2 ...
                + (x(1)^2+x(2)^2-1)^2;
[xstar fval2 exitflag] = fminsearch(KKTcondi,[1,1])
lambdastar1 = -(4*xstar(1)+xstar(2)-5)/(2*xstar(1))
g1 = xstar(1)^2+xstar(2)^2-1
H = [ 4 1 ; 1 2 ] + lambdastar1*[ 2 0 ; 0 2 ]
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 KKT103.m을 실행하면, 그림 6.4.3을 출력한다. 그림 6.4.3에서 녹색으로 칠해진 부채꼴이 실현가능영역이다. 흑색 점선으로 그려진 타원은 식 $f(\boldsymbol{x}) = 0.105$을 나타내고, 적색 긴점선으로 그려진 타원은 식 $f(\boldsymbol{x}) = 0.210$를 나타내고, 청색 반점선으로

그려진 타원은 식 $f(\boldsymbol{x}) = 0.315$를 나타낸다. 따라서, 이 최적화문제의 해 $\boldsymbol{x}^*$는 실현가능영역과 곡선 $f(\boldsymbol{x}) = 0.210$가 만나는 점 $\mathbf{P}$이다.

상보성조건에 의해서 다음 식이 성립함을 알 수 있다.

$$\lambda_1^* = 0, \quad \lambda_2^* = 0 \tag{5}$$

정리 6.4.1에서 기술한 KKT의 1차 필요조건에서 알 수 있듯이, 다음 식들이 성립한다.

$$\nabla f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\boldsymbol{x}^*) = \begin{bmatrix} 4x_1 + x_2 - 4.5 \\ x_1 + 2x_2 \end{bmatrix} + \lambda_3 \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \mathbf{0} \tag{6}$$

$$x_2 = 0 \tag{7}$$

식 (6)과 식 (7)을 풀면, 다음 해를 얻는다.

$$x_1^* = 0.875, \quad \lambda_3^* = 0.875 \tag{8}$$

다음 식들이 성립함을 확인할 수 있다.

$$g_1(\boldsymbol{x}^*) = -\frac{15}{64}, \quad g_2(\boldsymbol{x}^*) = -\frac{7}{8}, \tag{9}$$

따라서, 상보성조건 (6.4.7)이 성립함을 확인할 수 있다. 또한, 다음 식들이 성립한다.

$$\begin{aligned} &\nabla^2 f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla^2 g_i(\boldsymbol{x}^*) \\ &= \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix} + \lambda_3^* \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix} \end{aligned} \tag{10}$$

식 (11)에서 계산된 행렬이 양정치임을 쉽게 알 수 있다. ■

예제 6.4.6 Python을 사용해서 예제 6.4.5를 다시 다루기 위해서, 다음 Python프로그램 KKT103.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
```

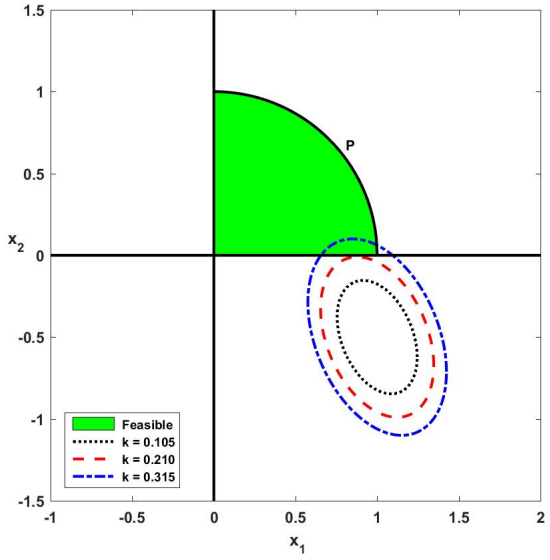


그림 6.4.3. 부등식제약조건 III

```
"""

import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize
# Feasible set
x1 = np.linspace(0,1,201);
x2 = np.sqrt(1-x1**2);
x1a = np.append(0,x1);
x2a = np.append(0,x2);
# Object function
tt = (2*np.pi/360)*np.arange(361);
yf1 = np.zeros((361,3));
yf2 = np.zeros((361,3));
print(yf2)
for kdum in range(0,3):
    k =  0.105*(kdum+1);
    yf1[:,kdum] = np.sqrt(k)*2/np.sqrt(7)*np.cos(tt)+1.0;
    yf2[:,kdum] = np.sqrt(k)*np.sin(tt) - 1/2*(yf1[:,kdum]-1.0) - 0.5;

# Plotting
fig = plt.figure()
plt.fill_between(x1a, 0, x2a, facecolor='green', label='Feasible')
plt.plot(yf1[:,0],yf2[:,0],'k:',lw=2.0,label='k=0.1')
plt.plot(yf1[:,1],yf2[:,1],'r--',lw=2.0,label='k=0.2')
plt.plot(yf1[:,2],yf2[:,2],'b-.',lw=2.0,label='k=0.3')
plt.plot(x1a,x2a,'k-',lw=2)
plt.plot([0, 0], [ -5, 5 ],'k',lw=2);
plt.plot([-5, 5 ], [0, 0 ],'k',lw=2);
plt.legend(loc='lower left', numpoints=1)
plt.axis('equal')
plt.axis([ -1.0, 2.0, -1.5, 1.5 ])
plt.text(0.8-0.02,0.7-0.02,'P')
plt.xlabel('\bf x_1')
plt.ylabel('\bf x_2')
plt.show()
fig.savefig('KKT103Py.png')
```

```
41
42 # KKT condition
43 KKTcondi = lambda x: ((4*x[0]+x[1]-5)*x[1] - (x[0]+2*x[1]-3)*x[0])**2 \
44                     + (x[0]**2+x[1]**2-1)**2;
45 xstar = scipy.optimize.fmin(func=KKTcondi,x0=[1,1])
46 print('xstar = ', xstar)
47 fval2 = KKTcondi(xstar)
48 print('fmin =', fval2)
49 lambdastar1 = -(4*xstar[0]+xstar[1]-5.0)/(2*xstar[0])
50 print('lambda = ', lambdastar1)
51 g1 = xstar[0]**2+xstar[1]**2-1.0
52 print('g1 = ', g1)
53 H = np.array([[ 4, 1], [1, 2 ]]) + lambdastar1*np.array([[ 2, 0 ], [0, 2 ]])
54 print('H = ', H)
55
56 # End of program
```

이 Python프로그램을 수행한 결과는 예제 6.4.5의 결과와 같다. ■

정리 6.4.2: 부등식제약문제의 충분조건

함수들 $f(\boldsymbol{x})$와 $g_i(\boldsymbol{x}),\ (i = 1, 2, \cdots, m)$가 2회연속미분가능(twice continuously differentiable)하다고 가정하자. 정칙인 실행가능점 $\boldsymbol{x}^*$와 Lagrange승수벡터 $\boldsymbol{\lambda}^*$가 다음 조건들을 만족한다고 하자.

$$g_i(\boldsymbol{x}^*) \le 0,\ (i = 1, 2, \cdots, l)$$

$$g_j(\boldsymbol{x}^*) = 0,\ (j = l+1, l+2, \cdots, m)$$

$$\nabla f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(\boldsymbol{x}^*) = \boldsymbol{0}$$

$$\lambda_i^* > 0,\quad (i \in A(\boldsymbol{x}^*))$$

$$\lambda_i^* = 0,\quad \left(i \in A^C(\boldsymbol{x}^*)\right)$$

$$\boldsymbol{y}^t \left[\nabla^2 f(\boldsymbol{x}^*) + \sum_{i=1}^{m} \lambda_i^* \nabla^2 g_i(\boldsymbol{x}^*) \right] \boldsymbol{y} > 0 \ \ \forall\, \boldsymbol{y}(\neq \boldsymbol{0}) \in B(\boldsymbol{x}^*)$$

이러한 조건 하에서 $\boldsymbol{x}^*$는 국소최적해(local optimal solution)이다.

제6.5절 MATLAB과 Python의 최적화함수들

이 절에서는 MATLAB이 제공하는 최적화함수들에 대한 사용법과 이에 해당하는 Python 함수들을 살펴보자.

6.5.1 Python함수 brute

무차별대입법(brute force method)은 주어진 격자의 각 점에서 함수값을 계산한 다음 그 중 최소값을 갖는 점을 선택하는 최적화 방법이다.

예제 6.5.1 Python scipy.optimize패키지의 함수 brute를 사용해서 무차별대입법을 적용하기 위해서, 다음 Python프로그램 UseBruteForce101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
Author: CBS
"""
import numpy as np
from scipy import optimize

params = (1,1,-4, 0.2, 0.2)
def ftn1(z, *params):
    x, y = z
    a, b, c, d, g = params
    return ( a*x**2+b*y**2+c*x )
def ftn2(z, *params):
    x, y = z
    a, b, c, d, g = params
    return ( (d*np.sin(y)+g*np.cos(y)) )
def ftn(z, *params):
     x, y = z
     a, b, c, d, g = params
     return ftn1(z, *params) + ftn2(z, *params)

rranges = ( slice(-np.pi, np.pi, 0.1), slice(-np.pi, np.pi, 0.1) )
resultt = optimize.brute(ftn, rranges, args=params, full_output=True, \
                         finish=optimize.fmin)
print('Global Minimum = ', resultt[0])
print('at ', resultt[1])
# End of program
```

이 Python프로그램 UseBruteForce1.Py는 무차별대입법을 써서 정의역인 $[-\pi,\pi]\times[-\pi,\pi]$에서 다음 함수의 최소값을 구하는 것이다.

$$f(x,y) = x^2 + y^2 - 4x + 0.2\sin y + 0.2\cos y \tag{1}$$

이 Python프로그램에서는 격자점들의 집합 $\{[-\pi+0.1*k, -\pi+0.1*l]\,|\,k,l=0,1,\cdots 62\}$를 변수 rranges에 저장한다. 무차별대입법은 이 집합의 각 점에서 함수값을 계산해서 그

함수값들 중에서 최소값을 갖는 점 $[x^*, y^*]$을 찾는다. 다음 단계로 점 $[x^*, y^*]$를 초기점으로 해서 Python함수 scipy.optimize.fmin를 적용해서 좀 더 정교한 최소점을 구한다. 만약 이 정교화 단계를 적용하고 싶지 않으면, 함수 brute의 옵션 finish에 None을 할당한다.

이 Python프로그램이 실행되면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다.

```
Global Minimum =  [ 2.00002912 -0.11040383]
at  -3.81106459268
```

■

6.5.2 MATLAB함수 fminbnd.m과 Python함수 fminbound

만약 목적함수가 대역적으로(globally) 아래로 볼록이 아니면, 극소점을 찾는 2차함수근사탐색법이나 Newton-Raphson법에서 하강이 아니라 상승하는 단계가 있을 수 있고, 결과적으로 발생된 벡터열이 수렴하지 않을 수 있다. 이러한 경우에는 섹션탐색법(sectionional search method) 또는 신뢰영역(trust region)을 함께 사용한다. 2차함수근사탐색법이나 Newton-Raphson법의 제n번째 단계에서 함수 $f(\boldsymbol{x})$의 근사함수를 $\tilde{f}_n(\boldsymbol{x})$라 하자. 다음 식이 성립하면, R을 신뢰영역이라 한다.

$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x}\in R} \tilde{f}_n(\boldsymbol{x}) \tag{6.5.1}$$

함수값 $f(\boldsymbol{x}_{n+1})$이 함수값 $f(\boldsymbol{x}_n)$보다 얼마나 작아지느냐에 따라 신뢰영역 R을 조절한다.

MATLAB함수 fminbnd.m의 옛날 이름은 fmin.m이다. 이 MATLAB함수 fminbnd.m은 단변수함수의 최소값을 구하기 위한 것으로 표준적 사용법은 다음과 같다.

```
>> [x,fval,exitflag] = fminbnd(fun,x1,x2,options)
```

여기서 첫 번째 입력변수 fun은 목적함수이고, 두 번째와 세 번째 입력변수들은 최소값의 탐색구간 [x1,x2]를 나타내고, 네 번째 입력변수 options는 최적화의 옵션들로서 MATLAB함수 optimoptions.m으로 지정할 수 있다. 또한, 첫 번째 출력변수 x는 최소값을 갖는 점을 나타내고, 두 번째 출력변수 fval은 그 최소값을 나타내고, 세 번째 출력변수 exitflag는 fminbnd.m의 출구조건(exit condition)을 나타낸다. 만약 이 출력변수값이 1이면, 이 fminbnd.m이 제공하는 값은 해로 수렴한다. 앞에서도 언급했듯이, MATLAB함수 fminbnd.m은 Brent법을 사용한다, 즉, 2차함수근사탐색법과 황금섹션탐색법을 바탕으로 한다. MATLAB함수 fminbnd.m에 대한 좀 더 자세한 내용을 알고자하면, MATLAB커맨드행에 'doc fminbnd'를 입력해보라.

예제 6.5.2 MATLAB함수 fminbnd.m의 다양한 사용법을 살펴보기 위해서, 다음 MATLAB프로그램 USEfminbnd101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: USEfminbnd101.m
%  Examples of MATLAB function fminbnd.m
%  To find a solution of nonlinear equation
% ------------------------------------------------------------------------
function USEfminbnd101
diary USEfminbnd101.txt
clear all, close all, format long
disp('(Example 1)')
x1 = fminbnd('sin',-2,2)
disp('(Example 2)')
[x2 fval2] = fminbnd(@(x) sin(x),-2,2)
disp('(Example 3)')
opts3 = optimset('TolX',1.0e-10,'Display','iter','MaxIter',10);
[x3 fval3] = fminbnd(@(x) sin(x),-2,2,opts3)
disp('(Example 4)')
myFMBfun4 = @(x) exp(-x)*(x-5)^2
[x4 fval4 exitflag] = fminbnd(myFMBfun4,-1,8)
disp('(Example 5)')
myFMBfun5 = inline('exp(-x)*(x-5)^2','x')
[x5 fval5] = fminbnd(myFMBfun5,-1,8)
disp('(Example 6)')
opts6 = optimset('TolX',1.0e-8,'Display','iter','MaxFunEvals',8);
[x6 fval6]= fminbnd(@(x) myFMBfun6(x),-1,9,opts6)
   % The m-file myFMBfun6.m shouuld be in the same directory.
diary off
end
% ------------------------------------------------------------------------
function ff6 = myFMBfun6(x,c)
ff6 = exp(-x)*(x-5)^2;
end
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 USEfminbnd101.m이 실행되면, 다음과 같은 결과가 OUTPUT 윈도우에 출력된다.

```
>> USEfminbnd101
(Example 1)
x1 = -1.570796126558752


(Example 2)
x2 = -1.570796126558752
fval2 = -0.999999999999980


(Example 3)
 Func-count     x          f(x)         Procedure
    1       -0.472136     -0.45479        initial
    2        0.472136      0.45479        golden
```

```
     3       -1.05573    -0.870259        golden
     4       -1.41641    -0.988106        golden
     5       -1.63656    -0.997838        parabolic
     6       -1.57193    -0.999999        parabolic
     7       -1.57076           -1        parabolic
     8        -1.5708           -1        parabolic
     9        -1.5708           -1        parabolic
    10        -1.5708           -1        parabolic
    11        -1.5708           -1        parabolic

종료 중: 최대 반복 횟수를 초과함
         - MaxIter 옵션 값을 늘리십시오.
         현재 함수 값: -1.000000

x3 = -1.570796326794297
fval3 = -1

(Example 4)
myFMBfun4 = @(x)exp(-x)*(x-4)^2
x4 = 4.000004137651406
fval4 = 3.135653553452158e-13
exitflag = 1

(Example 5)
myFMBfun5 =
     인라인 함수: myFMBfun5(x) = exp(-x)*(x-5)^2

x5 = 5.000012906285948
fval5 = 1.122340284053279e-12

(Example 6)
 Func-count     x          f(x)         Procedure
    1        2.81966     0.283456        initial
    2        5.18034  0.000182974        golden
    3        6.63932   0.00351486        golden
    4        5.87416   0.00214817        parabolic
    5        4.27864   0.00721296        golden
    6        4.83592  0.000213742        golden
```

```
    7          5.024  3.78986e-06        parabolic
    8        5.01493  1.47973e-06        parabolic

종료 중: 함수 실행의 최대 횟수를 초과함
         - MaxFunEvals 옵션 값을 늘리십시오.
         현재 함수 값: 0.000001


x6 = 5.014930322558419
fval6 = 1.479727737212582e-06
```

■

예제 6.5.3 Python을 사용해서 예제 6.5.2를 다시 다루기 위해서, 다음 Python프로그램 USEfminbound101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

import numpy as np
from scipy import optimize

x1 = optimize.fminbound(np.sin, -2, 2)
print('x1 = ', x1, ',  favl1 = ', np.sin(x1))

x2 = optimize.fminbound(lambda x:np.sin(x), -2, 2)
print('x2 = ', x2, ',  favl2 = ', np.sin(x2))

x3 = optimize.brent(np.sin, brack=(-2,2))
print('x3 = ', x3, ',  favl3 = ', np.sin(x3))

myFMBfun4 = lambda x: np.exp(-x)*(x-4)**2
x4 = optimize.fminbound(myFMBfun4, -1, 8)
print('x4 = ', x4, ',  favl4 = ', myFMBfun4(x4))

def myFMBfun5(x):
    return np.exp(-x)*(x-5)**2;
x5 = optimize.fminbound(myFMBfun5, -1, 8)
print('x5 = ', x5, ',  favl4 = ', myFMBfun4(x5))

x6 = optimize.brent(lambda x:np.exp(-x)*(x-5)**2, brack=(-1, 8))
print('x6 = ', x6, ',  favl6 = ', np.exp(-x6)*(x6-5)**2)

# End of program
```

MATLAB함수의 fminbnd.m에 해당하는 Python의 함수들로는 scipy.optimize.fminbound와 scipy.optimize.brent 등이 있다. 이 Python프로그램을 수행한 결과는 예제 6.5.2의 결과와 비슷하다. 그러나, Python의 함수 scipy.optimize.fminbound를 수행한 결과는 MATLAB

함수의 fminbnd를 수행한 결과와 같으나, Python의 함수 scipy.optimize.brent를 수행한 결과는 MATLAB함수의 fminbnd를 수행한 결과와 다를 수 있다. ■

6.5.3 MATLAB함수 fminsearch.m와 Python함수 fmin

MATLAB함수 fminsearch.m의 옛날 이름은 fmins.m이다. 이 MATLAB함수는 다변수함수의 최소값을 구하기 위한 것으로 표준적 사용법은 다음과 같다.

```
>> [x,fval,exitflag] = fminsearch(fun,x0,options)
```

여기서 첫 번째 입력변수 fun은 목적함수이고, 두 번째 입력변수 x0는 초기값을 나타내고, 세 번째 입력변수 options는 최적화의 옵션들로서 MATLAB함수 optimoptions.m으로 지정할 수 있다. 또한, 첫 번째 출력변수 x는 최소값을 갖는 점을 나타내고, 두 번째 출력변수 fval은 그 최소값을 나타내며, 세 번째 출력변수 exitflag는 fminsearch.m의 출구조건을 나타낸다. 만약 이 출력변수값이 1이면, 이 fminsearch.m이 제공하는 값은 해로 수렴한다.

MATLAB함수 fminsearch.m은 그래디언트벡터를 해석적이나 수치적으로 구하지 않고, Nelder & Mead (1965)가 제시한 단체알고리즘(simplex algorithm)을 사용한다. 앞에서 사용한 황금섹션탐색법와 2차함수근사탐색법은 목적함수가 단변수인 경우에만 적용할 수 있는 반면에, Nelder-Mead법은 다변수인 경우에도 적용할 수 있다. Nelder-Mead법을 이해하기 전에, 먼저 MATLAB프로그램 NelderMeadGraph101.m을 실행하라.

예제 6.5.4 Nelder-Mead법을 설명에 필요한 그림을 그리기 위해서, 다음 MATLAB프로그램 NelderMeadGraph101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename NelderMeadGraph101.m
% Graph for Nelder-Mead Method
% Programmed by CBS
% ------------------------------------------------------------------------
clear, clf
c = [0,0];
a = [3,2]; b = [1,-1];
m = (a+b)/2;
e = m + 2*(m-c)
d = (m+e)/2
s = (c+m)/2;
n = (a+c)/2;
hold on
plot([b(1) a(1)],[b(2) a(2)],'k','LineWidth',1.5)
set(gca,'fontsize',11,'fontweigh','bold')
plot([c(1) a(1)],[c(2) a(2)],'k','LineWidth',1.5)
plot([c(1) b(1)],[c(2) b(2)],'k','LineWidth',1.5)
```

```
plot([c(1) e(1)],[c(2) e(2)],'k','LineWidth',1.5)
plot([d(1) a(1)],[d(2) a(2)],'k','LineWidth',1.5)
plot([b(1) d(1)],[b(2) d(2)],'k','LineWidth',1.5)
plot([m(1) s(1)],[m(2) s(2)],'k','LineWidth',1.5)
plot([m(1) n(1)],[m(2) n(2)],'k','LineWidth',1.5)
%
plot([c(1) a(1)],[c(2) a(2)],'rd','LineWidth',1.5)
plot([c(1) b(1)],[c(2) b(2)],'rd','LineWidth',1.5)
plot([c(1) e(1)],[c(2) e(2)],'rd','LineWidth',1.5)
plot([d(1) a(1)],[d(2) a(2)],'rd','LineWidth',1.5)
plot([b(1) d(1)],[b(2) d(2)],'rd','LineWidth',1.5)
plot([m(1) s(1)],[m(2) s(2)],'rd','LineWidth',1.5)
plot([m(1) n(1)],[m(2) n(2)],'rd','LineWidth',1.5)
hold off
%
ax = -0.16;
ay = 0.21;
text(c(1)+ax,c(2)+ay,'\bf c','fontsize',13)
text(a(1)+ax,a(2)+ay,'\bf a','fontsize',13)
text(b(1)+ax,b(2)+ay,'\bf b','fontsize',13)
text(m(1)+ax,m(2)-ay,'\bf m','fontsize',13)
text(d(1)+ax,d(2)+ay,'\bf d','fontsize',13)
text(e(1)+ax,e(2)+ay,'\bf e','fontsize',13)
text(s(1)+ax,s(2)+ay,'\bf s','fontsize',13)
text(n(1)+ax,n(2)+ay,'\bf n','fontsize',13)
axis( [ -1 7 -2 3 ])
set(gcf,'color','w')
axis off
saveas(gcf,'NelderMeadGraph101','jpg')
% end of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 NelderMeadGraph101.m이 실행하면, 그림 6.5.1을 출력한다. ■

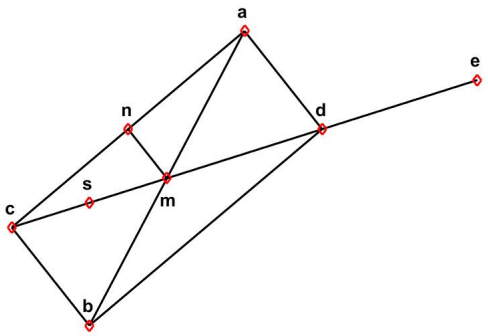


그림 6.5.1. Nelder-Mead법

예제 6.5.5 Python을 사용해서 그림 6.5.1을 그리기 위해서, 다음 Python프로그램 NelderMeadGraph101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt

c = np.array([0.0, 0.0]);
a = np.array([3.0, 2.0]); b = np.array([1.0,-1.0]);
m = (a+b)/2;
e = m + 2*(m-c);
d = (m+e)/2;
s = (c+m)/2;
n = (a+c)/2;

# plt.plotting
fig = plt.figure()
plt.plot([b[0], a[0]],[b[1], a[1]],'k',lw=2)
plt.plot([c[0], a[0]],[c[1], a[1]],'k',lw=2)
plt.plot([c[0], b[0]],[c[1], b[1]],'k',lw=2)
plt.plot([c[0], e[0]],[c[1], e[1]],'k',lw=2)
plt.plot([d[0], a[0]],[d[1], a[1]],'k',lw=2)
plt.plot([b[0], d[0]],[b[1], d[1]],'k',lw=2)
plt.plot([m[0], s[0]],[m[1], s[1]],'k',lw=2)
plt.plot([m[0], n[0]],[m[1], n[1]],'k',lw=2)

plt.plot([c[0], a[0]],[c[1], a[1]],'rd',lw=2)
plt.plot([c[0], b[0]],[c[1], b[1]],'rd',lw=2)
plt.plot([c[0], e[0]],[c[1], e[1]],'rd',lw=2)
plt.plot([d[0], a[0]],[d[1], a[1]],'rd',lw=2)
plt.plot([b[0], d[0]],[b[1], d[1]],'rd',lw=2)
plt.plot([m[0], s[0]],[m[1], s[1]],'rd',lw=2)
plt.plot([m[0], n[0]],[m[1], n[1]],'rd',lw=2)

ax = -0.16;
ay = 0.21;
plt.text(c[0]+ax,c[1]+ay,'c',fontsize=13)
plt.text(a[0]+ax,a[1]+ay,'a',fontsize=13)
plt.text(b[0]+ax,b[1]+ay,'b',fontsize=13)
plt.text(m[0]-0.5*ax,m[1]-1.2*ay,'m',fontsize=13)
plt.text(d[0]+ax,d[1]+ay,'d',fontsize=13)
plt.text(e[0]+ax,e[1]+ay,'e',fontsize=13)
plt.text(s[0]+ax,s[1]+ay,'s',fontsize=13)
plt.text(n[0]+ax,n[1]+ay,'n',fontsize=13)

plt.axis( [ -1.0, 7.0, -2.0, 3.0 ])
plt.axis('off')
plt.show()
fig.savefig('NelderMeadGraph101Py.png')

# End of program
```

이 Python프로그램을 수행하면, 그림 6.5.1이 그려진다. ■

그림 6.5.1을 바탕으로 Nelder-Mead법을 설명한 것이 다음 알고리즘이다. MATLAB함수

fminsearch.m에 대한 좀 더 자세한 내용을 알고자하면, MATLAB커맨드행에 'doc fminsearch'를 입력해보라.

알고리즘 6.5.1: Nelder-Mead법

(1단계) 목적함수 $f(x)$가 다음 식들을 만족하는 세 값들 a, b, c를 초기값들로 선택한다.

$$f(a) < f(b) < f(c)$$

(2단계) 만약 세 값들 a, b, c가 충분히 가깝거나 또는 $f(a), f(b), f(c)$가 충분히 가까우면, $(a, f(a))$를 극소점으로 하고 알고리즘을 멈춘다. 만약 그렇지 않으면, 제3단계를 넘어간다.

(3단계) 두 값들 a와 b의 평균 $m = [a+b]/2$를 구한다. 다음 식을 만족하는 값 e를 구한다.

$$e - m = 2[m - c]$$

만약 식 $f(e) < f(b)$가 성립하면, 값 e를 새로운 값 c로 선택하고 제2단계로 넘어간다. 그렇지 않으면, 제3-1단계로 넘어간다.

(3-1단계) 두 값들 m와 e의 평균 $d = [m+e]/2$를 구한다. 만약 식 $f(d) < f(c)$가 성립하면, 값 d를 새로운 값 c로 선택하고 제2단계로 돌아간다. 그렇지 않으면, 제3-2단계로 넘어간다.

(3-2단계) 만약 식 $f(d) < f(b)$가 성립하면, 제3-3단계로 넘어간다. 그렇지 않으면, 두 값들 m와 c의 평균 $s = [m+c]/2$를 구한다. 만약 식 $f(s) < f(c)$가 성립하면, 값 s를 새로운 값 c로 선택하고 제2단계로 넘어간다. 그렇지 않으면, 제3-3단계로 넘어간다.

(3-3단계) 두 값들 a와 b의 평균 $m = [a+b]/2$를 새로운 값 b로 하고 두 값들 a와 c의 평균 $n = [a+c]/2$를 새로운 값 c로 선택하고 제2단계로 돌아간다.

예제 6.5.6 MATLAB함수 fminsearch.m의 다양한 사용법을 살펴보기 위해서, 다음 MATLAB프로그램 USEfminsearch101.m을 실행해 보자.

```
% --------------------------------------------------------------------
%  Filename: USEfminsearch101.m
```

```
%  Examples of MATLAB function fminsearch.m
%  To find the minimum of a multivariable function
% ------------------------------------------------------------------------
function USEfminsearch101
clear all, close all, format long
disp('(Example 1)')
[x1 fval1] = fminsearch(@(x) 100*(x(2)-x(1)^2)^2 ...
                            + (1-x(1))^2,[-1.2,1])
disp('(Example 2)')
opts2 = optimset('TolX',1.0e-10,'Display','iter','MaxIter',10);
banana = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2
[x2 fval2 exitflag] = fminsearch(banana,[-1.2,1])
disp('(Example 3)')
opts3 = optimset('TolFun',1.0e-9,'Display','iter','MaxFunEvals',8);
[x3 fval3] = fminsearch(@(x) myFMSfun13(x),[-1.2,1],opts3)
   % The m-file myFMS13.m should be in the same directory.
end
% ------------------------------------------------------------------------
function ff13 = myFMSfun13(x)
ff13 = 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
end
%-------------------------------------------------------------------------
```

이 MATLAB프로그램 USEfminsearch101.m이 실행되면, 다음과 같은 결과가 OUTPUT 윈도우에 출력된다. 세 번째 예제에서 알 수 있듯이, MATLAB함수 fminsearch.m에서는 MATLBA함수 fminbnd.m의 옵션들과 더불어 'TolFun'이라는 옵션을 사용할 수 있다. 이 옵션에는 함수값에 대한 오차허용값을 지정한다.

```
>> USEfminsearch101
(Example 1)
x1 = 1.000022021783570   1.000042219751772
fval1 = 8.177661197416674e-10

(Example 2)
banana = @(x)100*(x(2)-x(1)^2)^2+(1-x(1))^2
x2 = 1.000022021783570   1.000042219751772
fval2 = 8.177661197416674e-10
exitflag = 1

(Example 3)
 Iteration   Func-count     min f(x)         Procedure
     0           1             24.2
     1           3            20.05         initial simplex
     2           5           5.1618         expand
     3           7           4.4978         reflect
     4           9           4.4978         contract outside
```

```
종료 중: 함수 실행의 최대 횟수를 초과함
         - MaxFunEvals 옵션 값을 늘리십시오.
         현재 함수 값: 4.497796

x3 = -1.079999999999999   1.125000000000000
fval3 = 4.497795999999987
```

■

예제 6.5.7 Python을 사용해서 예제 6.5.6을 다시 다루기 위해서, 다음 Python프로그램 USEfmin101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""

from scipy import optimize

def f(x):
    return 100*(x[1]-x[0]**2)**2 + (1-x[0])**2
x1 = optimize.fmin(f, [-1.2, 1])
print('x1 = ', x1)
print('fval = ', f(x1))

# End of program
```

Python패키지 scipy의 함수 scipy.optimize.fmin를 사용해서 Nelder-Mead법을 적용할 수 있다. 이 Python프로그램을 수행한 결과는 다음과 같다.

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 85
         Function evaluations: 159

x1 =  [ 1.00002202  1.00004222]
fval =  8.17766119742e-10
```

■

6.5.4 MATLAB함수 fminunc.m

알고리즘 6.3.1에서 설명했듯이, 방향탐색법에서는 먼저 방향벡터를 선택한 다음, 선택된 방향의 선분 상에서 목적함수값을 최소화하는 스텝크기 (step size) 를 선택하는 직선탐색 (line search) 를 한다. 반대로, 신뢰영역법 (trust region method) 에서는 스텝크기에 해당하는 신뢰영역의 크기를 먼저 결정한 다음 방향벡터를 정한다. 신뢰영역법을 제한스텝법 (restricted step method) 이라 부르기도 한다.

신뢰영역법에 의해 구해지는 벡터 $\boldsymbol{x}^{(n)}$ 에 대해서 다음과 같이 벡터 $\boldsymbol{d}^{(n)}$ 을 정의하자.

$$\boldsymbol{d}^{(n)} \doteq \arg\min_{\boldsymbol{d}\in R} \left[f(\boldsymbol{x}^{(n)}) + \nabla f(\boldsymbol{x}^{(n)})^t \boldsymbol{d} + \frac{1}{2}\boldsymbol{d}^t H^{(n)} \boldsymbol{d} \right] \tag{6.5.2}$$

다음 식이 성립하면, 신뢰영역 R을 줄여 나간다.

$$\left| \nabla f(\boldsymbol{x}^{(n)})^t \boldsymbol{d} + \frac{1}{2}[\boldsymbol{d}^{(n)}]^t H^{(n)} \boldsymbol{d}^{(n)} \right| > \epsilon \tag{6.5.3}$$

여기서 ϵ은 작은 양수로서 오차허용도 (tolerance level) 를 나타낸다. 또한, 식 (6.5.3) 이 성립하지 않으면, 신뢰영역 R을 늘여 나간다. 이렇게 정해진 신뢰영역 R에서 방향탐색법을 적용해서, 벡터 $\boldsymbol{x}^{(n+1)}$ 을 구한다.

MATLAB함수 fminunc.m은 제약조건이 없는 비선형 다변수함수의 최소값을 구하기 위한 것이다. 다음과 같은 최적화문제를 살펴보자.

$$\text{Minimize} f(\boldsymbol{x}). \tag{6.5.4}$$

여기서 $\boldsymbol{x}$는 벡터이고, $f(\cdot)$는 스칼라 함수이다.

최적화문제 (6.5.4) 을 풀기 위한 MATLAB함수 fminunc.m의 사용법은 다음과 같다.

```
x = fminunc(fun,x0)
x = fminunc(fun,x0,options)
x = fminunc(problem)
[x,fval] = fminunc(__)
[x,fval,exitflag,output] = fminunc(__)
[x,fval,exitflag,output,grad,hessian] = fminunc(__)
```

여기서 첫 번째 입력변수 fun은 목적함수이고, 두 번째 입력변수 x0는 초기값을 나타내고, 세 번째 입력변수 options는 최적화의 옵션들로서 MATLAB함수 optimoptions.m으로 지정할

수 있다. 또한, 첫 번째 출력변수 x는 최소값을 갖는 점을 나타내고, 두 번째 출력변수 fval은 그 최소값을 나타내고, 세 번째 출력변수 exitflag는 fminunc.m의 출구조건을 나타낸다. 만약 이 출력변수값이 1이면, 이 fminunc.m은 해로 수렴하는 것이다. 네 번째 출력변수 output은 출력물의 구조를 나타내고, 다섯 번째 출력변수 grad는 그래디언트벡터(gradient vector)를 나타내고, 여섯 번째 출력변수 hessian은 Hessian행렬을 나타낸다.

MATLAB함수 fminunc.m은 문제 크기에 따라 다른 알고리즘을 사용한다. 만약 문제 크기가 크지 않다면, 다음과 같은 옵션문을 사용한다.

```
>> optimoptions('LargeScale','off')
```

이 경우에 MATLAB함수 fminunc.m은 준Newton법과 직선탐색을 사용한다. 이 방법에서는 그래디언트벡터를 필요로 한다. 만약 그래디언트벡터가 입력되지 않으면, 유한차분법으로 그래디언트벡터를 계산한다. 만약 문제 크기가 크다면, 다음과 같은 옵션문을 사용한다.

```
>> optimoptions('LargeScale','on')
```

이 경우에 MATLAB함수 fminunc.m은 신뢰구간법과 공액방향법을 사용한다. 이 방법에서는 그래디언트벡터와 Hessian행렬을 필요로 한다. 만약 Hessian행렬이 입력되지 않으면, 유한차분법으로 Hessian행렬을 계산한다. 이 LargeScale옵션의 디폴트는 'on'이다.

예제 6.5.8 MATLAB함수 fminunc.m의 다양한 사용법을 살펴보기 위해서, 다음 MATLAB프로그램 USEfminunc101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: USEfminunc101.m
%  Examples of MATLAB function fminunc.m
%  To find the minimum of a multivariable function
% ------------------------------------------------------------------------
function USEfminunc101
clear all, close all, format long
disp('(Example 1)')
[x1 fval1] = fminunc(@(x) 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2,[1,1])
disp('(Example 2)')
% options = optimoptions('fminunc')
opts2 = optimoptions(@fminunc,'TolX',1.0e-8,'Display','iter');
myftn41 = @(x) 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;
[x2 fval2 exitflag] = fminunc(myftn41,[1,1])
disp('(Example 3)')
x0 = [1,1];
opts3 = optimoptions(@fminunc,'TolFun',1.0e-8,'Display','iter', ...
                 'MaxIter',999);
[x3 fval3] = fminunc(@myFMUfun33,x0,opts3)
disp('(Example 4)')
opts4 = optimoptions(@fminunc,'TolFun',1.0e-8,'Display','iter', ...
                 'GradObj','on','Algorithm','quasi-newton');
[x4 fval4] = fminunc(@myFMUfun44g,x0,opts4)
end
```

```
% --------------------------------------------------------------------
function ff33 = myFMUfun33(x)
ff33 = 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;
end
% --------------------------------------------------------------------
function [ff44,df] = myFMUfun44g(x)
ff44 = 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;
 if nargout > 1
    df(1) = 6*x(1) + 2*x(2);
    df(2) = 2*x(1) + 2*x(2);
 end
end
%--------------------------------------------------------------------
```

이 MATLAB명령문 USEfminunc101.m이 실행되면, 다음과 같은 결과가 OUTPUT윈도우에 출력된다. MATLAB함수 fminunc.m에서 디폴트 추정으로 trust-region법을 사용한다. 그러나, 그래디언트벡터가 입력되지 않으면 다른 추정법인 준Newton법을 사용한다. MATLAB함수 fminunc.m에서는 'GradObj'라는 옵션을 사용할 수 있다. 이 옵션을 'on'하면 함수의 그래디언트벡터를 입력할 수 있다. 이 MATLAB프로그램에서 목적함수를 정의하는 MATLAB파일 myFMUftn44g.m에 두 번째 출력변수로 그래디언트벡터를 지정하였다. 또한, 옵션 'Hessian'을 'on'하면, Hessian행렬을 입력할 수 있다. 만약 Hessian행렬이 제시되지 않으면, 유한차분법으로 Hessian행렬을 계산한다. 이 MATLAB프로그램을 수행한 결과는 다음과 같다.

```
(Example 1)
경고: Gradient must be provided for trust-region algorithm;
using quasi-newton algorithm instead.

Optimization completed because the size of the gradient is less than
the default value of the function tolerance.

<stopping criteria details>
x1 =
   1.0e-06 *
   0.254083674796170  -0.202933921209731

fval1 = 1.317333248878051e-13

(Example 2)
경고: Gradient must be provided for trust-region algorithm;
using quasi-newton algorithm instead.
```

```
Optimization completed because the size of the gradient is less than
the default value of the function tolerance.

<stopping criteria details>
x2 =
   1.0e-06 *
   0.254083674796170  -0.202933921209731

fval2 = 1.317333248878051e-13
exitflag = 1

(Example 3)
경고: Gradient must be provided for trust-region algorithm;
using quasi-newton algorithm instead.
```

Iteration	Func-count	f(x)	Step-size	First-order optimality
0	3	6		8
1	6	0.25	0.125	1
2	9	0.0805763	1	0.389
3	12	0.0510334	1	0.276
4	15	0.00234029	1	0.158
5	18	0.000114716	1	0.0366
6	21	3.44139e-07	1	0.00116
7	24	9.1273e-10	1	4.45e-05
8	27	1.31733e-13	1	1.16e-06
9	30	4.2407e-16	1	7.4e-08

```
Local minimum found.

Optimization completed because the size of the gradient is less than
the selected value of the function tolerance.

<stopping criteria details>
x3 =
   1.0e-07 *
  -0.125525053308671   0.229899054791367
```

```
fval3 = 4.240701020179912e-16


(Example 4)

                                                          First-order
 Iteration  Func-count       f(x)        Step-size       optimality
     0           1               6                           8
     1           2            0.25          0.125            1
     2           3       0.0805763              1        0.389
     3           4       0.0510334              1        0.276
     4           5      0.00234029              1        0.158
     5           6     0.000114717              1       0.0366
     6           7      3.4415e-07              1      0.00116
     7           8     9.13885e-10              1      4.45e-05
     8           9     1.46626e-13              1      1.16e-06
     9          10     1.99506e-17              1      1.54e-08

Local minimum found.

Optimization completed because the size of the gradient is less than
the selected value of the function tolerance.

<stopping criteria details>
x4 =
   1.0e-08 *
    0.234850356415246   0.063806985421769

fval4 = 1.995055876498883e-17
```

이 결과물에서 알 수 있듯이, 그래디언트벡터를 지정하는 네 번째 예제의 결과가 그렇지 않은 세 번째 예제의 결과보다 더 빠르고 정확하다. ■

6.5.5 MATLAB함수 lsqnonlin.m과 Python함수 leastsq

MATLAB함수 lsqnonlin.m은 목적함수 $\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}),\, f_2(\boldsymbol{x}),\, \cdots,\, f_N(\boldsymbol{x})]^t$ 의 노름을 최소화하기 위한 것이다. 즉, 다음 식을 만족하는 $\boldsymbol{x}^*$ 를 구하는 것이다.

$$\text{Minimize } \|\boldsymbol{f}(\boldsymbol{x})\|^2.$$

여기서 $\|\boldsymbol{f}(\boldsymbol{x})\|^2 \doteq \sum_{n=1}^{N} |f_n(\boldsymbol{x})|^2$ 이다. 이 MATLAB함수의 표준적 사용법은 다음과 같다.

```
>> [x,resnormSq,residual,exitflag] = lsqnonlin(fun,x0,lb,ub,options)
```

여기서 첫 번째 입력변수 fun은 목적함수이고, 두 번째 입력변수 x0는 초기값을 나타내고, 세 번째 입력변수 lb와 네 번째 입력변수 ub는 각각 최소값의 하한과 상한을 나타내며, 다섯 번째 입력변수 options는 최적화의 옵션들로서 MATLAB함수 optimoptions.m으로 지정할 수 있다. 또한, 첫 번째 출력변수 x는 최소점을 나타내고, 두 번째 출력변수 resnormSq는 최소노름제곱 $\|\boldsymbol{f}(\boldsymbol{x}^*)\|^2$를 나타내고, 세 번째 출력변수 residual은 $\boldsymbol{f}(\boldsymbol{x}^*)$를 나타내고, 네 번째 출력변수 exitflag는 MATLAB함수 lsqnonlin.m의 출구조건을 나타낸다. 만약 이 출력변수값이 1이면, MATLAB함수 lsqnonlin.m가 제공하는 값이 해로 수렴하는 것이다. MATLAB함수 lsqnonlin.m은 문제의 크기에 따라 다른 알고리즘을 사용한다. MATLAB함수 lsqnonlin.m에 대한 좀 더 자세한 내용을 알고자하면, MATLAB커맨드행에 'doc lsqnonlin'를 입력해보라.

예제 6.5.9 MATLAB함수 lsqnonlin.m의 다양한 사용법을 살펴보기 위해서, 다음 MATLAB프로그램 USElsqnonlin101.m을 실행해 보자.

```
% --------------------------------------------------------------------
%  Filename: USElsqnonlin101.m
%  Examples of MATLAB function lsqnonlin.m
%  To find least squares estimate of a nonlinear function
%  Programmed by CBS
% --------------------------------------------------------------------
function USElsqnonlin101
clear all; close all, format long
% Example 1
disp('(Example 1)')
[x1 resnormSq1] = lsqnonlin(@(x) 3*x(1)^2 + 2*x(1)*x(2) ...
                                + x(2)^2,[1,1])
% Example 2
disp('(Example 2)')
opts2 = optimset('TolX',1.0e-8,'Display','iter','MaxIter',50);
myOPfun41 = @(x) 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;
[x2 resnormSq2 residual exitflag] = lsqnonlin(myOPfun41,[1,1])
% Example 3
disp('(Example 3)')
x0 = [ 0.3 0.4 ];
[x3 resnormSq3] = lsqnonlin(@myOPfun43,x0)
% Example 4
disp('(Example 4)')
global tt Sratio
DataSet44 = [ 0.600 0.998; 0.670 0.991; 0.710 0.996; 0.750 0.995
              0.790 0.979; 0.852 0.981; 0.901 0.934; 0.952 0.861
              1.001 0.715; 1.048 0.524; 1.090 0.284; 1.161 0.131
              1.200 0.062; 1.241 0.024; 1.301 0.009; 1.356 0.003 ];
tt = DataSet44(:,1);
Sratio = DataSet44(:,2);
beta0 = [ 20.0, 1.25 ]    % Initial coefficients
lb = [ 15.0, 0.0 ]         % lower bound for beta
ub = [ 25.0, 2.0 ]         % upper bound for beta
```

```
[beta resnormSq4] = lsqnonlin(@myOPfun44,beta0,lb,ub,opts2)
% Plot the original and fitted values
ttconti = linspace( 0.95*min(tt),1.05*max(tt),21);
Sfit = 1./( 1 + exp( beta(1)*(ttconti-beta(2)) ) );
plot(ttconti,Sfit,'k-',tt,Sratio,'r+','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
legend('fitted value','original value','location','NE')
xlabel('\bf time','fontsize',12)
ylabel('Ratio','fontsize',12)
axis([ 0.95*min(tt) 1.05*max(tt), -0.05 1.05 ])
saveas(gcf,'USElsqnonlin101d','jpg')
% Example 5
disp('(Example 5)')
X = 0:.05:1;
y = 1*exp(2*X)+3*exp(4*X)+10*randn(size(X));
bb0 = [5, 1, 5, 1, 5 ];    % Initial coefficients
opts5 = optimset('Largescale','off');
[bb resnormSq5] = lsqnonlin(@myOPfun45,bb0,[],[],opts5,X,y)
% Plotting the original and fitted values
yfit = bb(1) + bb(2)*exp(bb(3).*X)+bb(4)*exp(bb(5).*X);
figure
plot(X,yfit,'k-',X,y,'r+','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
legend('fitted value','original value','location','NW')
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
axis([ -0.05 1.05 0 1.05*max(y) ])
saveas(gcf,'USElsqnonlin101e','jpg')
end
% ------------------------------------------------------------------
function ff43 = myOPfun43(x)
k = 1:1:20;
ff43 = 2 + 2*k - exp(k*x(1)) - exp(k*x(2));
end
% ------------------------------------------------------------------
function Sdiff = myOPfun44(beta)
global tt Sratio
Scalc = 1./(1+ exp( beta(1)*(tt-beta(2)) ) );
Sdiff = Scalc - Sratio;
end
%-------------------------------------------------------------------
function diff45 = myOPfun45(bb,X,Y)
% This function is called by lsqnonlin.
% bb is a vector which contains the coefficients of the equation.
% X and Y are the option data sets that were passed to lsqnonlin.
A = bb(1); B = bb(2); C = bb(3); D = bb(4); E = bb(5);
diff45 = A + B.*exp(C.*X) + D.*exp(E.*X) - Y;
end
%-------------------------------------------------------------------
```

MATLAB함수 lsqnonlin.m을 사용하는 방법은 MATLAB함수 fminunc.m을 사용하는 방법과 비슷하다. 첫 번째와 두 번째 lsqnonlin.m을 실행한 결과는 다음과 같다.

```
(Example 1)
Optimization terminated: the first-order optimality measure is
less than 1e-4 times options.TolFun.
x1 =  -0.000257114770347   0.001268690998389
```

```
resnormSq1 =  1.335185970656756e-012


(Example 2)
Optimization terminated: the first-order optimality measure is
less than 1e-4 times options.TolFun.
x2 =  -0.000257114770347   0.001268690998389
resnormSq2 =  1.335185970656756e-012
residual =  1.155502475400532e-006
exitflag =  1
```

세 번째 lsqnonlin.m은 다음 목적함수를 최소화하는 x_1과 x_2를 찾기 위한 것이다.

$$f(x_1, x_2) = \sum_{k=1}^{20} [2 + 2k - \exp(kx_1) - \exp(kx_2)]^2 \tag{1}$$

초기값으로는 $[x_1\ x_2] = [0.3\ 0.4]$를 사용한다. 이 세 번째 lsqnonlin.m을 실행한 결과는 다음과 같다.

```
(Example 3)
Optimization terminated: norm of the current step is less than OPTIONS.TolX.
x3 = 0.165190854937641   0.165190780564681
resnormSq3 = 1.449479644326904e+003
```

이 결과물에서 알 수 있듯이, 최소제곱추정벡터는 $[0.1652, 0.1652]$이고 오차제곱합은 다음과 같다.

$$\sum_{k=1}^{20} [2 + 2k - \exp(0.1652x_1) - \exp(0.1652x_2)]^2 = 1449.5 \tag{2}$$

네 번째 lsqnonlin.m은 다음 로지스틱함수(logistic function)를 DataSet44에 주어진 관찰값들에 적합시키는 것이다.

$$S_t = \frac{1}{1 + \exp(\beta_1[t - \beta_2])} \tag{3}$$

초기벡터는 $[\beta_1\ \beta_2] = [20\ 1.25]$이고, 계수들의 하한은 각각 15.0과 0.0이고, 상한은 각각 25.0와 2.0이다. 이 네 번째 lsqnonlin.m을 실행한 결과는 다음과 같다.

```
(Example 4)
beta0 =  20.000000000000000   1.250000000000000
lb =    15     0
ub =    25     2

                                        Norm of      First-order
 Iteration  Func-count     f(x)          step          optimality
     0          3         2.09019                        11.1
     1          6        0.351905        1.66009           4.7
     2          9       0.0133683        0.90751         0.279
     3         12      0.00345004       0.697284         0.026
     4         15      0.00237159       0.369029       0.00184
     5         18       0.0023219       0.101732      0.000241
     6         21      0.00232142      0.0108195      2.89e-05
Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the default value of the function tolerance.

<stopping criteria details>
beta =  18.743225327999681   1.049188377335864
resnormSq4 =   0.002321416213093
```

이 결과물에서 알 수 있듯이, 최소제곱추정벡터는 $[18.7432,\ 1.0492]$이고 오차제곱합은 다음과 같다.

$$\sum_{k=1}^{16}\left[S_{t_k}-\frac{1}{1+\exp(18.7432[t_k-1.0492])}\right]^2=0.0023 \tag{4}$$

원래의 관찰값들과 모형 (3)에 의한 적합값들이 그림 6.5.2에 그려져 있다.

다섯 번째 lsqnonlin.m은 다음 식에서 발생된 관찰값들을 적합시키는 것이다.

$$y=b_1+b_2\exp(b_3x)+b_4\exp(b_5x)+10\epsilon \tag{5}$$

여기서 $x=0.05k,\ (k=0,1,\cdots,20)$이고, ϵ은 표준정규난수(standard normal random number)이다. 이 다섯 번째 lsqnonlin.m을 실행한 결과는 다음과 같다. 이 결과물에서 알 수 있듯이, 초기벡터를 $[b_1,\ b_2,\ b_3,\ b_4,\ b_5\,]=[5,\ 1,\ 5,\ 1,\ 5\,]$라고 하면, 최소제곱추정벡터는

$[-0.0418,\ 0.0065,\ 3.9088,\ 3.4647,\ 3.8801]$이고 오차제곱합은 다음과 같다.

$$\sum_{k=0}^{20}\{y_k - [-0.0418 + 0.0065\exp(3.9088 \times 0.05k) + 3.4647\exp(3.8801 \times 0.05k)\}^2$$
$$= 1837.1 \tag{6}$$

원래의 관찰값들과 모형 (6)에 의한 적합값들이 그림 6.5.3에 그려져 있다.

```
(Example 5)
Solver stopped prematurely.

lsqnonlin stopped because it exceeded the function evaluation limit,
options.MaxFunEvals = 500 (the default value).

bb =  -0.041774471905825   0.006474918512741   3.908763977328325
       3.464712998597462   3.880054422580988
resnormSq5 =  1.837099407462912e+03
```

■

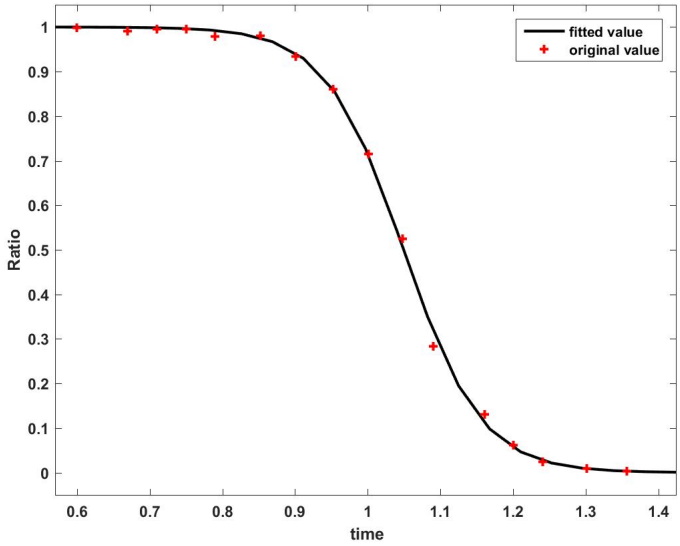


그림 6.5.2. 비선형회귀와 lsqnonlin.m I

예제 6.5.10 Python패키지 scipy의 함수 optimize.leastsq를 사용해서 최소제곱추정하는 예를 살펴보기 위해서, 다음 Python프로그램 UseLeastsq101.Py를 실행해 보자.

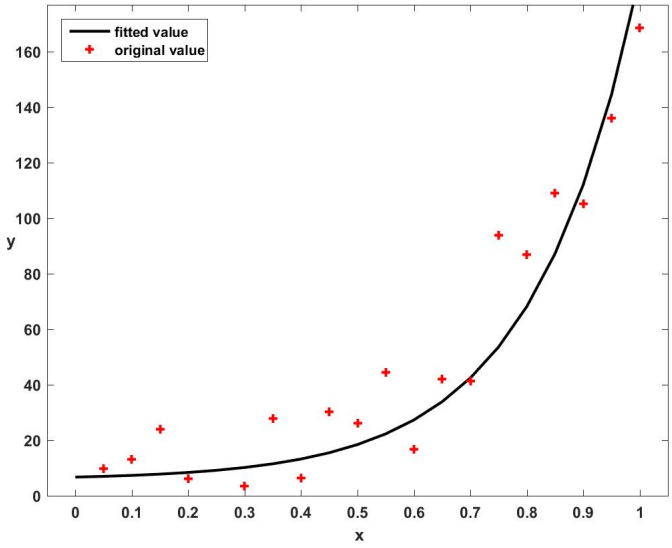


그림 6.5.3. 비선형회귀와 lsqnonlin.m II

```
"""
Created on Fri Jan 13 23:10:59 2017
Author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
import random

# Given data
tt = np.arange(-100,100); tt = tt/100.0*np.pi;
random.seed(9001)
RandTerm = np.random.normal(0,1.1,len(tt))     # sigma=1.1
SOEdata = 3.0 - 2.0*tt + 1.0*tt**2 + RandTerm
# Fitting Model
def FitModel(tt, coef):
    return coef[0] + coef[1]*tt + coef[2]*tt**2
# Initial Values
coefInit = np.array([0.0, 0.0, 0.0], dtype=float)
# SOEdata - FitModel
def residuals(coef, y, t):
    return y - FitModel(t, coef)
# Least-Square Fitting
coefFin, flag = optimize.leastsq(residuals, coefInit, args=(SOEdata, tt))
# Output
print('Coefficients = ', coefFin)
print('fflag = ', flag)
# Plotting
fig = plt.figure()
plt.plot(tt, SOEdata,'r-',lw=1.0)
plt.plot(tt, FitModel(tt,coefFin),'k-',lw=2.0)
plt.legend(['Original', 'Fitted'],loc='upper right', numpoints=1)
plt.axis([-np.pi, np.pi, -5, 25])
plt.show()
fig.savefig('UseLeastsq101Py.jpg')

```

```
# End of Program
```

이 Python프로그램을 실행하면, 다음 데이터세트가 발생한다.

$$\text{SOEdata} = \left\{3 - 2t + t^2 + \epsilon_t \mid t = -1.00, -0.99, \cdots, 0.99, 1.00\right\} \tag{1}$$

여기서 $\{\epsilon_t\}$는 정규확률분포 $\mathcal{N}\left(0, 1.1^2\right)$에서 생성된 정규난수들이다. 우리의 목적은 Python 함수 optimize.leastsq를 사용해서 2차함수 $y_t = c_0 + c_1 t + c_2 t^2$를 이 데이터세트에 적합시키는 것이다. 계수들의 초기벡터로 $[c_0,\, c_1,\, c_2\,] = [0,\, 0,\, 0\,]$를 사용한다. 이 Python프로그램을 실행한 결과 추정식은 다음과 같다.

$$\hat{y}_t = 2.9889 - 2.0837t + 0.9998t^2 \tag{2}$$

원래 생성된 관찰값들과 모형 (2)에 의한 적합값들이 그림 6.5.4에 그려져 있다. ■

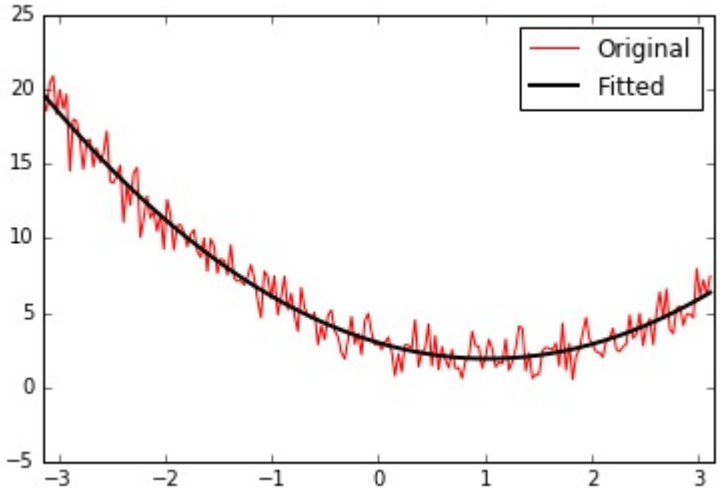


그림 6.5.4. 비선형회귀와 Python함수 leastsq

예제 6.5.11 Python패키지 scipy의 함수 optimize.curve_fit를 사용해서, 곡선을 추정하는 예를 살펴보기 위해서, 다음 Python프로그램 UseCurveFit101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
Author: CBS
"""

```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import optimize
import random

def FitModel(tt, coef0, coef1, coef2):
    return coef0 + coef1*tt + coef2*tt**2

# Given data
tt = np.arange(-100,100); tt = tt/100.0*np.pi;
random.seed(9001)
RandTerm = np.random.normal(0,1.1,len(tt))     # sigma=1.1
SOEdata = FitModel(tt, 3, -2, 1) + RandTerm
# Curve Fitting
Out1 = optimize.curve_fit(FitModel,tt,SOEdata)
CF = np.array(Out1[0])
# Output
print('Estimated Coefficients = ', CF)

# Plotting
fig = plt.figure()
plt.plot(tt, SOEdata,'r-',lw=1.0,label='Original')
plt.plot(tt, FitModel(tt,CF[0],CF[1],CF[2]),'k-', \
                      lw=2.0,label='Curve Fitted')
plt.legend(['Original', 'Fitted'],loc='upper right', numpoints=1)
plt.axis([-np.pi, np.pi, -5, 25])
plt.show()
fig.savefig('UseCurveFit101Py.jpg')

# End of Program
```

이 Python프로그램을 실행하면, 다음 데이터세트가 발생한다.

$$\text{SOEdata} = \left\{3 - 2t + t^2 + \epsilon_t \mid t = -1.00, -0.99, \cdots, 0.99, 1.00\right\} \tag{1}$$

여기서 $\{\epsilon_t\}$는 정규확률분포 $\mathcal{N}\left(0, 1.1^2\right)$에서 생성된 정규난수들이다. 우리의 목적은 Python 함수 optimize.curve_fit를 사용해서 2차함수 $y_t = c_0 + c_1 t + c_2 t^2$를 이 데이터세트에 적합시키는 것이다. 이 Python프로그램을 실행한 결과 추정식은 다음과 같다.

$$\hat{y}_t = 3.0908t - 1.9183 + 0.9772t^2 \tag{2}$$

원래 생성된 관찰값들과 모형 (2)에 의한 적합값들이 그림 6.5.5에 그려져 있다. ■

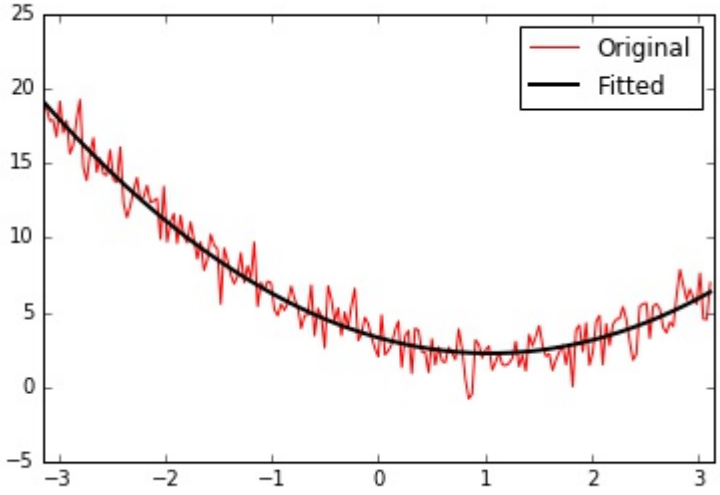


그림 6.5.5. 비선형회귀와 Python함수 curve_fit

6.5.6 MATLAB함수 lsqnonneg.m과 Python함수 nnls

다음과 같은 벡터 $\boldsymbol{x}(\in R^N)$에 관한 목적함수 $f(\boldsymbol{x}) \doteq \|C\boldsymbol{x} - \boldsymbol{d}\|$의 최소값을 구하는 문제를 살펴보자.

$$\text{Minimize} \|C\boldsymbol{x} - \boldsymbol{d}\| \quad \text{subject to} \quad \boldsymbol{x} \geq \mathbf{0}. \tag{6.5.5}$$

여기서 C는 실행렬(real matrix)이고 $\boldsymbol{d}$는 실벡터(real vector)이다. MATLAB함수 lsqnonneg.m는 이 최적화문제를 풀기 위한 것으로 표준적 사용법은 다음과 같다.

```
>> [x,resnormSq,residual,exitflag] = lsqnonneg(C,d,x0,options)
```

여기서 첫 번째 입력변수는 행렬 C이고, 두 번째 입력변수는 벡터 $\boldsymbol{d}$이고, 세 번째 입력변수 x0는 초기벡터이고, 네 번째 입력변수 options는 최적화의 옵션들로서 MATLAB함수 optimoptions.m으로 지정할 수 있다. 또한, 첫 번째 출력변수 x는 최소점 $\boldsymbol{x}^*$를 나타내고, 두 번째 출력변수 resnormSq는 최소노름제곱 $\|C\boldsymbol{x}^* - \boldsymbol{d}\|^2$를 나타내고, 세 번째 출력변수 residual은 함수값 $\boldsymbol{d} - C\boldsymbol{x}^*$를 나타내고, 네 번째 출력변수 exitflag는 lsqnonneg.m의 출구조건을 나타낸다. 만약 이 출력변수값이 1이면, 이 lsqnonneg.m은 해로 수렴하는 것이다. MATLAB함수 lsqnonneg.m에 대한 좀 더 자세한 내용을 알고자하면, MATLAB커맨드행에 'doc lsqnonneg'를 입력해보라.

예제 6.5.12 MATLAB함수 lsqnonneg.m의 사용법을 살펴보기 위해서, 다음 MATLAB프로그램 USElsqnonneg101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: USElsqnonneg101.m
%  Examples of MATLAB function lsqnonneg.m
%  Solve nonnegative least-squares constraints problem
% ------------------------------------------------------------------------
clear all, close all
C = magic(10); C = C(:,1:3)
beta = [ 2 3 4 ]'
d = C*beta + randn(10,1)  % Add normal random numbers
x0 = [ 0.1 0.1 0.1 ]'   % initial vector
options = optimset('MaxIter',50)
[x resnormSq1, residual, exitflag ] = lsqnonneg(C,d,x0,options)
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB명령문이 실행되면, 행렬 C, 벡터 $\boldsymbol{d}$ 그리고 초기벡터 $\boldsymbol{x}_0$가 다음과 같음을 알 수 있다.

$$C = \begin{bmatrix} 92 & 99 & 1 \\ 98 & 82 & 7 \\ 4 & 81 & 88 \\ 85 & 87 & 19 \\ 86 & 93 & 25 \\ 17 & 24 & 76 \\ 23 & 5 & 82 \\ 79 & 6 & 13 \\ 10 & 12 & 94 \\ 11 & 18 & 100 \end{bmatrix}, \quad \boldsymbol{d} = \begin{bmatrix} 484.5766 \\ 464.3616 \\ 602.6481 \\ 507.2695 \\ 548.4356 \\ 410.4659 \\ 390.8536 \\ 229.0393 \\ 432.9109 \\ 475.7603 \end{bmatrix}, \quad \boldsymbol{x}_0 = \begin{bmatrix} 0.1 \\ 0.1 \\ 0.1 \end{bmatrix} \tag{1}$$

이 행렬 C와 벡터 d는 다음과 같은 관계를 만족한다.

$$\boldsymbol{d} = C \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} + \boldsymbol{\epsilon} \tag{2}$$

여기서 $\boldsymbol{\epsilon}$은 정규난수벡터이다. MATLAB함수 lsqnonneg.m을 수행하면, 추정벡터가 다음과 같음을 알 수 있다.

$$\boldsymbol{x} = [2.0140,\ 2.9785,\ 4.0095\,]^t \tag{3}$$

또한, 오차제곱합은 다음과 같다.

$$\|C\boldsymbol{x}^* - \boldsymbol{d}\|^2 = 7.1230 \tag{4}$$

■

예제 6.5.13 Python을 사용해서 예제 6.5.12을 다시 다루기 위해서, 다음 Python프로그램 USEnnls101.Py를 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
Author: CBS
"""

import numpy as np
# import matplotlib.pyplot as plt
from scipy import optimize

# Define minimisation function
def ftn(x, A, b):
    return np.sum(A*x,1) - b

#Define problem
C = np.array([ [ 92.0 , 99.0 , 1.0 ], [ 98.0 , 82.0 , 7.0 ],
  [ 4.0 , 81.0 , 88.0 ], [ 85.0 , 87.0 , 19.0 ], [86.0 , 93.0 , 25.0 ],
  [ 17.0 , 24.0 , 76.0 ], [ 23.0 , 5.0 , 82.0 ], [ 79.0 , 6.0 , 13.0 ],
  [ 10.0 , 12.0 , 94.0 ],  [ 11.0 , 18.0 , 100.0 ] ])
d = np.array([ 484.5766, 464.3616, 602.6481, 507.2695, 548.4356,
               410.4659, 390.8536, 229.0393, 432.9109, 475.7603 ])
Out1 = optimize.nnls(C,d)
print('x =', Out1[0])
print('Resid Norm =', Out1[1])

# End of program
```

이 Python프로그램을 수행하면, 추정벡터가 다음과 같음을 알 수 있다.

$$\boldsymbol{x} = [1.9996,\ 2.9772,\ 4.0131\,]^t \tag{1}$$

또한, 오차제곱합은 다음과 같다.

$$\|C\boldsymbol{x}^* - \boldsymbol{d}\|^2 = 5.1038^2 = 26.0494 \tag{2}$$

■

6.5.7 MATLAB함수 quadprog.m

목적함수가 2차함수이고 제약식이 1차인 최적화문제를 2차프로그래밍문제(quadratic programming problem: QP problem)라고 부른다. 제약식이 있는 최적화문제를 푸는 일반적인 방법을 사용해서 QP문제를 풀 수 있다. 그러나, QP문제를 풀 때는 선형프로그래밍문제(linear

programming problem: LP problem)를 푸는 알고리즘을 바탕으로 한 방법이 더 간단하고 빠르기 때문에, 이러한 방법을 사용하는 것이 보편적이다.

MATLAB에서 QP문제를 푸는 도구(solver)는 MATLAB함수 quadprog.m이다. 이 MATLAB함수를 적용하기 위해서, 먼저 QP문제를 다음과 같은 표준형으로 나타내자.

$$\text{Minimize} \left[\frac{1}{2}\boldsymbol{x}^t H\boldsymbol{x} + \boldsymbol{c}^t\boldsymbol{x}\right] \tag{6.5.6}$$

$$\text{subject to } A\boldsymbol{x} \preceq \boldsymbol{b}, \quad A_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq}, \quad \boldsymbol{l}_b \preceq \boldsymbol{x} \preceq \boldsymbol{u}_b\,. \tag{6.5.7}$$

여기서 식 $\boldsymbol{y} \preceq \boldsymbol{z}$ 는 각 i에 대해서 벡터 $\boldsymbol{y}$의 제i번째 원소가 벡터 $\boldsymbol{z}$의 제i번째 원소보다 작거나 같음을 의미한다.

예제 6.5.14 다음과 같은 QP문제를 풀어보자.

$$\text{Minimize} \quad \left[4.x_1^2 + 5x_2^2 + 6x_3^2 - 2x_1x_2 + 4x_1x_3 + 2x_2x_3\right] \tag{1}$$

$$\text{subject to} \quad 1x_1 + 2x_2 - 3x_3 \le 2, \tag{2}$$

$$3x_1 - 1x_2 + 1x_3 = 1, \tag{3}$$

$$x_1 \ge 0, \;\; x_2 \ge 0, \;\; x_3 \ge 0. \tag{4}$$

다음과 같은 벡터들과 행렬들을 사용하면, 이 QP문제를 표준형으로 바꿀 수 있다.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad H = \begin{bmatrix} 4 & -1 & 2 \\ -1 & 5 & 1 \\ 2 & 1 & 6 \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} -2 \\ -2 \\ -1 \end{bmatrix} \tag{5}$$

이 QP문제를 풀기 위해서, 다음 MATLAB프로그램 USEquadprog101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename USEquadprog101.m
% Quadratic Problem Example 1
% Programmed by CBS
% ------------------------------------------------------------------------
clear, close all
starttime = cputime;  % for calculating CPU time
% QP problem
H = [4 -1  2; -1 5 1;2 1 6];
c = [-2 -2 -1];
A = [1 2 -3];
b = [2];
Aeq =[3 -1 1];
beq = [1];
LB=[0 0 0];
```

```
UB = [];
% Call MATLAB function linprog
[x,fval,exitflag,OUTPUT,lambda] = quadprog(H,c,A,b,Aeq,beq,LB,UB)
% Print the result
fprintf('Inequality constraint = '), disp(A*x)
fprintf('Equality constraint = '), disp(Aeq*x)
totaltime = cputime - starttime;
fprintf('\nTotal time (seconds)= %8.5f \n\n',totaltime)
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 USEquadprog101.m을 실행하면, 다음과 같은 결과물을 얻는다.

```
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.

<stopping criteria details>
x =
   0.511627906931786
   0.534883720795375
   0.000000000000019

fval =  -1.127906976744173
exitflag =    1

OUTPUT =

             message: 'Minimum found that satisfies the constraints.…'
          algorithm: 'interior-point-convex'
      firstorderopt: 1.168635136032059e-10
    constrviolation: 2.220446049250313e-16
         iterations: 5
       cgiterations: []

lambda =
    ineqlin: 2.791739489533158e-10
      eqlin: 0.162790697599749
      lower: [3x1 double]
      upper: [3x1 double]

Inequality constraint =    1.581395348522479
```

```
Equality constraint =    1.000000000000000
Total time (seconds)=  0.37440
```

즉, 함수 $f(x)$는 점 $x^* = [0.5116, 0.5349, 0]$에서 극소값 -1.1279를 갖는다. 또한, 이 극소점에서 다음 식들이 성립한다.

$$1x_1 + 2x_2 - 3x_3 = 1.581 \tag{6}$$

$$3x_1 - 1x_2 + 1x_3 = 1 \tag{7}$$

$$x_1 > 0, \quad x_2 > 0, \quad x_3 = 0 \tag{8}$$

■

6.5.8 MATLAB함수들 gradient.m과 jacobian.m

MATLAB에는 그래디언트벡터와 Jacobian행렬을 구하기 위한 함수들 gradient.m과 jacobian.m이 수록되어 있다. MATLAB함수 quiver.m과 더불어 이들을 사용해서 그래디언트벡터장(gradient vector field)을 그릴 수 있다. 이 그래디언트벡터장을 바탕으로 극대점 또는 극소점이 어디에 위치하는가를 직관적으로 알 수 있다. 또한, 그래디언트벡터장은 미분방정식을 푸는데도 유용하다.

예제 6.5.15 MATLAB함수 gradient.m은 그래디언트벡터를 구하기 위한 것이다. 이 함수의 사용법을 살펴보기 위해서, 다음 MATLAB프로그램 USEgradient101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: USEgradient101.m
%  Calculating ing gradient vector using gradient.m
%  Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
dum = -3:0.6:3;
[ x,y ] = meshgrid(dum);
F = exp( - (x.^2+y.^2)/2 );
[Fx,Fy] = gradient(F,0.1,0.1);
contour(dum,dum,F,20)
set(gca,'fontsize',11,'fontweigh','bold')
hold on
quiver(dum,dum,Fx,Fy)
axis square
hold off
```

```
saveas(gcf,'USEgradient101','jpg')
% End of program
% --------------------------------------------------------------------------
```

MATLAB함수 gradient.m의 표준적 사용법은 다음과 같다.

```
>> [Fx,Fy] = gradient(F,dx,dy)
>> [Fx,Fy,Fz] = gradient(F,dx,dy,dz)
```

첫 번째 사용법은 2변수 함수에 관한 것이고, 두 번째 사용법은 3변수 함수에 관한 것이다. 첫 번째 입력변수 F는 함수값 F이고, 두 번째 입력변수들 dx, dy 그리고 dz는 각 축 방향에서 증분이고, 출력변수들 Fx, Fy 그리고 Fz는 그래디언트벡터의 각 원소이다. 또한, MATLAB 함수 quiver.m은 각 점에서 그래디언트벡터를 나타내는 그래디언트벡터장을 그리기 위한 것으로, 표준적 사용법은 다음과 같다.

```
>> quiver(x,y,Fx,Fy)
```

첫 번째와 두 번째 입력변수들 [x,y]는 그래디언트벡터가 계산되는 점이고, 세 번째와 네 번째 입력변수들 [Fx,Fy]는 그래디언트벡터이다.

MATLAB프로그램 USEgradient101.m을 실행하면, 다음 함수의 그래디언트벡터를 계산하고 그래디언트벡터장을 그린다. 이 그래디언트벡터장이 그림 6.5.6에 그려져 있다.

$$F(x,y) = \frac{1}{2\pi}\exp\left(-\frac{1}{2}[x^2+y^2]\right)$$

■

예제 6.5.16 Python을 사용해서 예제 6.5.15를 다시 다루기 위해서, 다음 Python프로그램 USEgradient101.Py를 실행해 보자.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 13 14:58:39 2017

@author: user
"""

import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-3,3,11);
y = np.linspace(-3,3,11);
xx, yy = np.meshgrid(x,y)
```

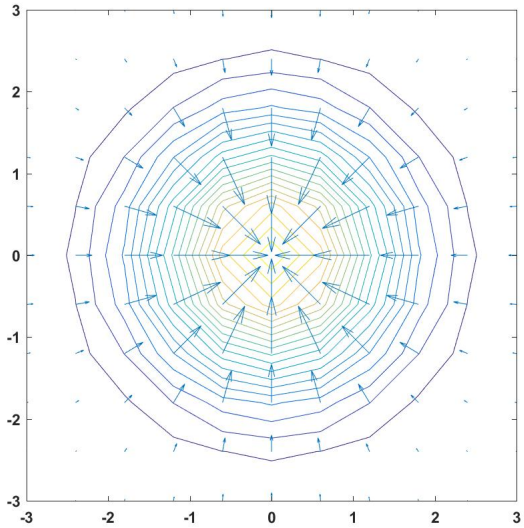


그림 6.5.6. MATLAB함수 gradient.m

```
F = np.exp( - (xx**2 + yy**2)/2 );
Fx, Fy = np.gradient(F,0.1,0.1)

# plt.plotting
fig = plt.figure()
h = plt.contour(xx,yy,F,20)
Q = plt.quiver(x,y,xx,yy, units='width')
plt.axis('square')   # Better than equal
plt.show()
fig.savefig('USEgradient101Py.png')

# End of program
```

이 Python프로그램을 수행한 결과는 예제 6.5.15의 결과와 같다. ■

MATLAB의 Symbolic Math Toolbox에서 제공하는 MATLAB함수 jacobian.m은 Jacobian행렬을 구하기 위한 것이다. 다음 예제는 MATLAB함수 jacobian.m의 사용법을 살펴보기 위한 것이다.

예제 6.5.17 Jacobian행렬을 구하는 예로서, 다음 MATLAB프로그램 USEjacobian101.m을 실행해 보자. 이 MATLAB프로그램은 Janathan Rosenberg가 웹사이트에 게시한 것을 출력물의 크기를 조절하기 위해 약간 변형한 것이다.

```
% --------------------------------------------------------------------
%  Filename: USEjacobian101.m
%  Gradients, Gradient Plots and Tangent Planes
%  from www.math.umd.edu/~jmr/241/gradients.html
%  Copyright by Janathan Rosenberg
```

```
% --------------------------------------------------------------------
clear all, close all
syms x y
%%% (Example 1)
f = ( (x^2-1) + (y^2-4) + (x^2-1)*(y^2-4) )/(x^2+y^2+1)^2
gradf = jacobian(f,[x,y])
[xx,yy] = meshgrid(-3:0.1:3,-3:0.1:3);
ffun = @(x,y) eval(vectorize(f));
fxfun = @(x,y) eval(vectorize(gradf(1)));
fyfun = @(x,y) eval(vectorize(gradf(2)));
subplot(2,2,1)
surf(xx,yy,ffun(xx,yy))
subplot(2,2,2)
contour(xx,yy,ffun(xx,yy),30)
hold on
[xx,yy] = meshgrid(-3:0.25:3,-3:0.25:3);
quiver(xx,yy,fxfun(xx,yy),fyfun(xx,yy),0.6)
axis equal tight
hold off
%%% (Example 2) Minimum
f1 = x^2 + y^2
gradf1 = jacobian(f1,[x,y])
[xx,yy] = meshgrid(-1:0.1:1,-1:0.1:1);
f1fun = @(x,y) eval(vectorize(f1));
f1xfun = @(x,y) eval(vectorize(gradf1(1)));
f1yfun = @(x,y) eval(vectorize(gradf1(2)));
subplot(2,2,3)
surf(xx,yy,f1fun(xx,yy))
subplot(2,2,4)
contour(xx,yy,f1fun(xx,yy),10)
set(gca,'xtick',-1:0.5:1,'ytick',-1:0.5:1)
hold on
[xx,yy] = meshgrid(-1:0.25:1,-1:0.25:1);
quiver(xx,yy,f1xfun(xx,yy),f1yfun(xx,yy),0.6)
axis equal tight
hold off
saveas(gcf,'USEjacobian101a','jpg')
%%% (Example 3) Maximum
figure
subplot(2,2,1)
surf(xx,yy,-f1fun(xx,yy))
subplot(2,2,2)
contour(xx,yy,-f1fun(xx,yy),10)
set(gca,'xtick',-1:0.5:1,'ytick',-1:0.5:1)
hold on
[xx,yy] = meshgrid(-1:0.25:1,-1:0.25:1);
quiver(xx,yy,-f1xfun(xx,yy),-f1yfun(xx,yy),0.6)
axis equal tight
hold off
%%% (Example 4) Saddle point
f2 = x^2 - y^2
gradf2 = jacobian(f2,[x,y])
[xx,yy] = meshgrid(-1:0.1:1,-1:0.1:1);
f2fun = @(x,y) eval(vectorize(f2));
f2xfun = @(x,y) eval(vectorize(gradf2(1)));
f2yfun = @(x,y) eval(vectorize(gradf2(2)));
subplot(2,2,3)
surf(xx,yy,f2fun(xx,yy))
subplot(2,2,4)
contour(xx,yy,f2fun(xx,yy),10)
set(gca,'xtick',-1:0.5:1,'ytick',-1:0.5:1)
```

```
hold on
[xx,yy] = meshgrid(-1:0.25:1,-1:0.25:1);
quiver(xx,yy,f2xfun(xx,yy),f2yfun(xx,yy),0.6)
axis equal tight
hold off
saveas(gcf,'USEjacobian101b','jpg')
% End of program
% --------------------------------------------------------------------
```

첫 번째 예제는 다음 함수의 3차원 그래프와 그래디언트벡터를 포함하는 등고선그림(contour map)를 그린다. 즉, 등고선그림 위에 그래디언트벡터장을 그린다. 이 3차원 그래프는 그림 6.5.7의 좌측상단에 그리고 등고선그림는 이 그림의 우측상단에 그려져 있다.

$$f(x,y) = \frac{x^2 - 1 + y^2 - 4 + [x^2 - 1][y^2 - 4]}{[x^2 + y^2 + 1]^2} \tag{1}$$

두 번째 예제는 다음 함수의 3차원 그래프와 등고선그림 위에 그래디언트벡터장을 그린다. 이 3차원 그래프는 그림 6.5.8의 좌측하단에 그리고 등고선그림는 이 그림의 우측하단에 그려져 있다.

$$f(x,y) = x^2 + y^2 \tag{2}$$

세 번째 예제는 다음 함수의 3차원 그래프와 등고선그림 위에 그래디언트벡터장을 그린다. 이 3차원 그래프는 그림 6.5.8의 좌측상단에 그리고 등고선그림는 이 그림의 우측상단에 그려져 있다. 이 그래프들을 그리는 목적은 극대점을 찾기 위한 것이다.

$$f(x,y) = -x^2 - y^2 \tag{3}$$

MATLAB함수 jacobian.m의 표준적 사용법은 다음과 같다.

```
>> grad = jacobian(F,[x,y])
```

이 MATLAB명령문에서 첫 번째 입력변수 F는 그래디언트벡터를 구하고자 하는 종속변수이고, 두 번째 입력변수 [x,y]는 독립변수들이다. 또한, 출력변수 grad는 그래디언트벡터이다.

네 번째 예제는 다음 함수의 3차원 그래프와 등고선그림 위에 그래디언트벡터장을 그린다. 이 3차원 그래프는 그림 6.5.8의 좌측하단에 그리고 등고선그림는 이 그림의 우측하단에 그려져 있다. 이 그래프들을 그리는 목적은 안장점(saddle point)을 찾기 위한 것이다.

$$f(x,y) = x^2 - y^2 \tag{4}$$

■

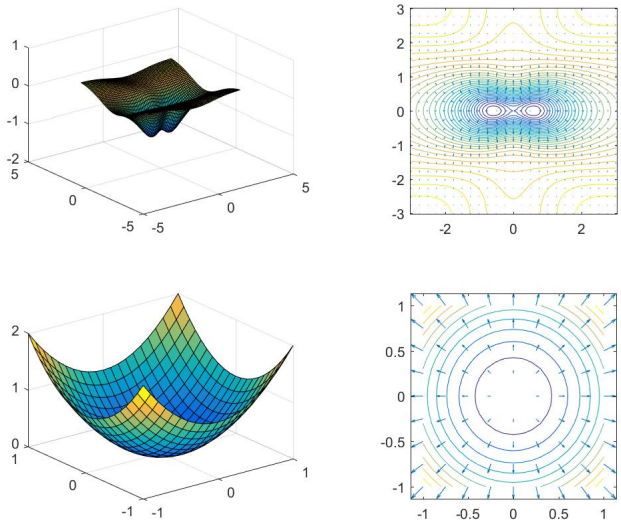


그림 6.5.7. MATLAB함수 jacobian.m과 그래디언트장 I

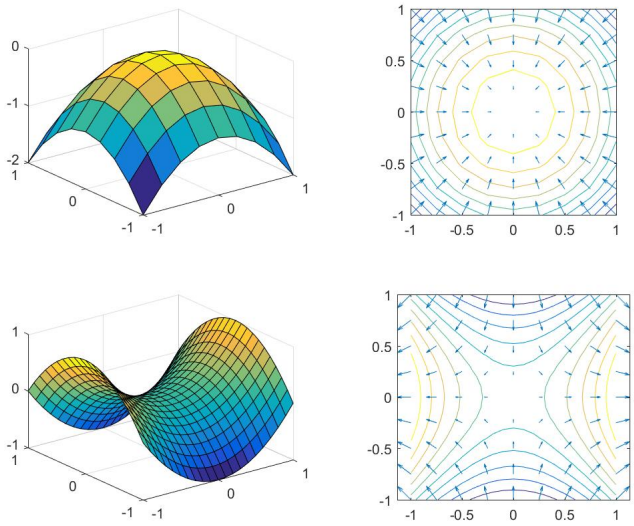


그림 6.5.8. MATLAB함수 jacobian.m과 그래디언트장 II

예제 6.5.18 Python을 사용해서 예제 6.5.17을 다시 다루기 위해서, 다음 Python프로그램 USEjacobian101.Py를 실행해 보자.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 13 01:36:05 2017
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
```

```
from matplotlib.ticker import LinearLocator, FormatStrFormatter
# from sympy import *

# Make Dataset 1
x = np.arange(-3, 3, 0.1)
y = np.arange(-3, 3, 0.1)
X, Y = np.meshgrid(x, y)
F = ( (X**2-1) + (Y**2-4) + (X**2-1)*(Y**2-4) )/(X**2+Y**2+1)**2
# Make Dataset 2
x1 = np.arange(-1, 1, 0.1)
y1 = np.arange(-1, 1, 0.1)
X1, Y1 = np.meshgrid(x1, y1)
F1 = X1**2+Y1**2

# Plot the surface
fig = plt.figure()
# subplot(2,2,1)
ax1 = fig.add_subplot(2,2,1,projection='3d')
surf = ax1.plot_surface(X,Y,F, cmap=cm.coolwarm, \
                        linewidth=0, antialiased=False)
# Customize the z axis.
ax1.set_zlim(-2.0, 1.0)
ax1.zaxis.set_major_locator(LinearLocator(3))
ax1.zaxis.set_major_formatter(FormatStrFormatter('%0.1f'))
# Add a color bar which maps values to colors.
# fig.colorbar(surf, shrink=0.5, aspect=5)
# subplot(2,2,2)
ax2 = fig.add_subplot(222)
h = ax2.contour(X,Y,F,10)
Q = plt.quiver(x,y,X,Y, units='width')
ax2.axis('square')   # Better than equal
# subplot(2,2,3)
ax3 = fig.add_subplot(2,2,3,projection='3d')
surf = ax3.plot_surface(X1,Y1,F1, cmap=cm.coolwarm, \
                        linewidth=0, antialiased=False)
ax3.set_xlim(-1.0, 1.0)
ax3.xaxis.set_major_locator(LinearLocator(3))
ax3.xaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax3.set_ylim(-1.0, 1.0)
ax3.yaxis.set_major_locator(LinearLocator(3))
ax3.yaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax3.set_zlim(0.0, 2.0)
ax3.zaxis.set_major_locator(LinearLocator(3))
ax3.zaxis.set_major_formatter(FormatStrFormatter('%0.1f'))
# subplot(2,2,4)
ax4 = fig.add_subplot(224)
h = ax4.contour(X1,Y1,F1,10)
Q = plt.quiver(x1,y1,X1,Y1, units='width')
ax4.axis('square')   # Better than equal

plt.show()
fig.savefig('USEjacobian101aPy.png')

# Make Dataset 3
x2 = np.arange(-1, 1, 0.1)
y2 = np.arange(-1, 1, 0.1)
X2, Y2 = np.meshgrid(x2, y2)
F2 = - X2**2-Y2**2
# Make Dataset 4
x3 = np.arange(-1, 1, 0.1)
y3 = np.arange(-1, 1, 0.1)
```

```
X3, Y3 = np.meshgrid(x3, y3)
F3 = X3**2-Y3**2

# Plot the surface
fig = plt.figure()
# subplot(2,2,1)
ax1 = fig.add_subplot(2,2,1,projection='3d')
surf = ax1.plot_surface(X2,Y2,F2, cmap=cm.coolwarm, \
                        linewidth=0, antialiased=False)
# Customize axes
ax1.set_xlim(-1.0, 1.0)
ax1.xaxis.set_major_locator(LinearLocator(3))
ax1.xaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax1.set_ylim(-1.0, 1.0)
ax1.yaxis.set_major_locator(LinearLocator(3))
ax1.yaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax1.set_zlim(-2.0, 0.0)
ax1.zaxis.set_major_locator(LinearLocator(3))
ax1.zaxis.set_major_formatter(FormatStrFormatter('%0.1f'))
# subplot(2,2,2)
ax2 = fig.add_subplot(222)
h = ax2.contour(X2,Y2,F2,10)
Q = plt.quiver(x2,y2,X2,Y2, units='width')
ax2.axis('square')   # Better than equal

# subplot(2,2,3)
ax3 = fig.add_subplot(2,2,3,projection='3d')
surf = ax3.plot_surface(X3,Y3,F3, cmap=cm.coolwarm, \
                        linewidth=0, antialiased=False)
ax3.set_xlim(-1.0, 1.0)
ax3.xaxis.set_major_locator(LinearLocator(3))
ax3.xaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax3.set_ylim(-1.0, 1.0)
ax3.yaxis.set_major_locator(LinearLocator(3))
ax3.yaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax3.set_zlim(-1.0, 1.0)
ax3.zaxis.set_major_locator(LinearLocator(3))
ax3.zaxis.set_major_formatter(FormatStrFormatter('%0.1f'))
# subplot(2,2,4)
ax4 = fig.add_subplot(224)
h = ax4.contour(X3,Y3,F3,10)
Q = plt.quiver(x3,y3,X3,Y3, units='width')
ax4.axis('square')   # Better than equal

plt.show()
fig.savefig('USEjacobian101bPy.png')

# End of program
```

이 Python프로그램을 수행한 결과는 예제 6.5.17의 결과와 같다. ■

6.5.9 MATLAB함수 fmincon.m와 Python함수 minimize

MATLAB함수 fmincon.m은 제약조건이 부등식인 경우에 비선형 다변수함수의 극소값을 구하기 위한 것이다. 이 MATLAB함수를 적용하기 위해서, 먼저 이 최적화문제를 다음과

같은 표준형으로 나타내자.

$$\text{Minimize } f(\boldsymbol{x}) \tag{6.5.8}$$

$$\text{subject to} \quad A\boldsymbol{x} \preceq \boldsymbol{b} \quad A_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq}, \quad \boldsymbol{l}_b \preceq \boldsymbol{x} \preceq \boldsymbol{u}_b, \tag{6.5.9}$$

$$\boldsymbol{c}(\boldsymbol{x}) \preceq \boldsymbol{0}, \quad \boldsymbol{c}_{eq}(\boldsymbol{x}) = \boldsymbol{0}. \tag{6.5.10}$$

여기서 식 $\boldsymbol{y} \preceq \boldsymbol{z}$ 는 각 i에 대해서 벡터 $\boldsymbol{y}$의 제i번째 원소가 벡터 $\boldsymbol{z}$의 제i번째 원소보다 작거나 같음을 의미한다. 또한, $\boldsymbol{x}$는 벡터이고, $f(\cdot)$는 스칼라함수이고, $\boldsymbol{c}(\cdot)$와 $\boldsymbol{c}_{eq}(\cdot)$는 벡터값 함수들이고, $\boldsymbol{b}$와 $\boldsymbol{b}_{eq}$는 벡터들이고, A와 A_{eq}는 행렬들이다. 함수들 $f(\cdot), \boldsymbol{c}(\cdot)$와 $\boldsymbol{c}_{eq}(\cdot)$는 선형함수들 또는 비선형함수들이다.

MATLAB함수 fmincon.m의 사용법은 다음과 같다.

```
x = fmincon(fun,x0,A,b)
x = fmincon(fun,x0,A,b,Aeq,beq)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon)
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
x = fmincon(problem)
[x,fval] = fmincon(__ )
[x,fval,exitflag,output] = fmincon(__ )
[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(__ )
```

이 fmincon명령문들에 사용되는 첫 번째 입력변수 fun은 목적함수이고, 두 번째 입력변수는 초기값 x0를 나타낸다. 또한, 입력변수 options는 최적화의 옵션들로서 MATLAB함수 optimoptions.m으로 지정할 수 있다. 첫 번째 출력변수 x는 극소값을 갖는 점을 나타내고, 두 번째 출력변수 fval은 그 극소값을 나타내고, 세 번째 출력변수 exitflag는 fmincon.m의 출구조건을 나타낸다. 만약 이 출력변수값이 1이면, 이 fmincon.m은 해로 수렴하는 것이다. 네 번째 출력변수 output은 출력물의 구조를 나타내고, 다섯 번째 출력변수 grad는 그래디언트벡터(gradient vector)나타내고, 여섯 번째 출력변수 hessian은 Hessian행렬을 나타낸다.

다음 예제들을 통해서 MATLAB함수 fmincon.m의 사용법을 살펴보자. 이 예제들은 MathWorks사가 제시한 것들이다.

예제 6.5.19 MATLAB함수 fmincon.m을 사용하는 첫 번째 MATLAB프로그램 USEfmincon101.m은 다음과 같다.

```
% ------------------------------------------------------------------------
%  Filename: USEfmincon101.m
%  Examples of MATLAB function fmincon.m
%  To find the minimum of a nonlinear funtion under constraints
%  From MATLAB Help
% ------------------------------------------------------------------------
function USEfmincon101
diary USEfmincon101.txt
disp('(Example 1)')
 Rosenbrock = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
 x0 = [-1,2];
 A = [1,2];
 b = 1;
 x001 = fmincon(Rosenbrock,x0,A,b)
 clear all
disp('(Example 2)')
 Rosenbrock = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
 x0 = [0.5,0];
 A = [1,2];
 b = 1;
 Aeq = [2,1];
 beq = 1;
 x002 = fmincon(Rosenbrock,x0,A,b,Aeq,beq)
 clear all
disp('(Example 3)')
 fun003 = @(x) 1+x(1)/(1+x(2)) - 3*x(1)*x(2) + x(2)*(1+x(1));
 lb = [0,0];
 ub = [1,2];
 A = [];
 b = [];
 Aeq = [];
 beq = [];
 x0 = [0.5,1];
 x003 = fmincon(fun003,x0,A,b,Aeq,beq,lb,ub)
 clear all
disp('(Example 4)')
 fun003 = @(x) 1+x(1)/(1+x(2)) - 3*x(1)*x(2) + x(2)*(1+x(1));
 lb = [0,0];
 ub = [1,2];
 A = [];
 b = [];
 Aeq = [];
 beq = [];
 x0 = [0.5,1]/5;
 x004 = fmincon(fun003,x0,A,b,Aeq,beq,lb,ub)
 clear all
disp('(Example 5) Nonlinear Constraint')
 Rosenbrock = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
 A = [];
 b = [];
 Aeq = [];
 beq = [];
 lb = [0,0.2];
 ub = [0.5,0.8];
 x0 = [1/4,1/4];
 nonlcon = @circlecon;
 x005 = fmincon(Rosenbrock,x0,A,b,Aeq,beq,lb,ub,nonlcon)
diary off
end
% ------------------------------------------------------------------------
function [c,ceq] = circlecon(x)
```

```
c = (x(1)-1/3)^2 + (x(2)-1/3)^2 - (1/3)^2;
ceq = [];
end
% ------------------------------------------------------------------------
```

첫 번째 fmincon 명령문에서는 다음과 같은 Rosenbrock 함수의 극소값을 구한다.

$$\text{Rosenbrock}(\boldsymbol{x}) = 100\left[x_2 - x_1^2\right]^2 + [1 - x_1]^2 \tag{1}$$

초기값벡터와 부등식제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \quad \begin{bmatrix} 1, & 2 \end{bmatrix} \boldsymbol{x} \leq 1 \tag{2}$$

이 fmincon 명령문을 실행한 결과는 다음과 같다.

```
Local minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.
<stopping criteria details>
x001 = 0.502202637435719   0.248898560083203
```

이 결과물에서 알 수 있듯이, 점 $(0.5022, 0.2489)$ 에서 Rosenbrock 함수가 극소값을 갖는다.

두 번째 fmincon 명령문에서도 Rosenbrock 함수의 극소값을 구한다. 또한, 초기값벡터, 부등식제약조건, 그리고 등호제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 1, & 2 \end{bmatrix} \boldsymbol{x} \leq 1, \quad \begin{bmatrix} 2, & 1 \end{bmatrix} \boldsymbol{x} = 1 \tag{3}$$

이 fmincon 명령문을 실행한 결과는 다음과 같다.

```
Local minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.
<stopping criteria details>
x002 =   0.414944316846950   0.170111366306101
```

이 결과물에서 알 수 있듯이, 점 $(0.4149, 0.1701)$ 에서 Rosenbrock 함수가 극소값을 갖는다.

세 번째 fmincon명령문에서는 다음 함수의 극소값을 구한다.

$$f_3(\boldsymbol{x}) = 1 + \frac{x_1}{1+x_2} - 3x_1x_2 + x_2[1+x_1] \tag{4}$$

초기값벡터와 부등식제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix} \preceq \boldsymbol{x} \preceq \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{5}$$

이 fmincon명령문을 실행한 결과는 다음과 같다.

```
Local minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.
<stopping criteria details>
x003 =   0.999999978181818   1.999999928000000
```

이 결과물에서 알 수 있듯이, 점 $(1.0000, 2.0000)$에서 함수 $f_3(\boldsymbol{x})$가 극소값을 갖는다.

네 번째 fmincon명령문에서도 함수 $f_3(\boldsymbol{x})$의 극소값을 구한다. 초기값벡터와 부등식제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \frac{1}{5}\begin{bmatrix} 0.5 \\ 1 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 \end{bmatrix} \preceq \boldsymbol{x} \preceq \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{6}$$

이 fmincon명령문을 실행한 결과는 다음과 같다.

```
Local minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.
<stopping criteria details>
x004 =   1.0e-06 *( 0.399999394351953   0.400000466758231 )
```

이 fmincon명령문은 세 번째 fmincon명령문에서 초기값벡터를 $[0.5, 1]^t$에서 $[0.1, 0.2]^t$로 바꾼 것이다. 이 경우에 결과물이 아주 달라져서, 함수 $f_3(\boldsymbol{x})$가 점 $(0.0000, 0.0000)$에서 극소값을 갖음을 알 수 있다.

다섯 번째 fmincon 명령문에서도 Rosenbrock 함수의 극소값을 구한다. 또한, 초기값벡터, 부등식제약조건, 그리고 비선형제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0.2 \end{bmatrix} \preceq \boldsymbol{x} \preceq \begin{bmatrix} 0.5 \\ 0.8 \end{bmatrix}, \quad \left[x_1 - \frac{1}{3}\right]^2 + \left[x_2 - \frac{1}{3}\right]^2 \leq \left[\frac{1}{3}\right]^2 \tag{7}$$

이 비선형조건을 새로운 MATLAB 함수 circlecon에 저장하였다. 이 fmincon 명령문을 실행한 결과는 다음과 같다.

```
Local minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.
<stopping criteria details>
x005 =   0.499999983949673   0.249999978109737
```

이 결과물에서 알 수 있듯이, 점 $(0.5000, 0.2500)$ 에서 Rosenbrock 함수가 극소값을 갖는다. ■

예제 6.5.20 MATLAB 함수 fmincon.m을 사용하는 두 번째 MATLAB 프로그램 USEfmincon102.m은 다음과 같다.

```
% ------------------------------------------------------------------------
%  Filename: USEfmincon102.m
%  Examples of MATLAB function fmincon.m w/ Nondefault Options
%  To find the minimum of a nonlinear funtion under constraints
%  From MATLAB Help
% ------------------------------------------------------------------------
function USEfmincon102
diary USEfmincon102.txt
 Rosenbrock = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
 A = [];
 b = [];
 Aeq = [];
 beq = [];
 lb = [];
 ub = [];
 x0 = [0,0];
 nonlcon = @unitdisk;
 options = optimoptions('fmincon','Display','iter','Algorithm','sqp');
 x201 = fmincon(Rosenbrock,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
diary off
end
% ------------------------------------------------------------------------
function [c,ceq] = unitdisk(x)
c = x(1)^2 + x(2)^2 -1;
ceq = [];
end
% ------------------------------------------------------------------------
```

이 fmincon명령문에서도 Rosenbrock함수의 극소값을 구한다. 초기값벡터와 부등식제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad x_1^2 + x_2^2 - 1 \le 0 \tag{1}$$

이 MATLAB프로그램에서는 MATLAB함수 fmincon.m에 옵션을 지정하기 위해서 MATLAB함수 optimoptions.m를 사용하였다. 이 함수에서 옵션 Display에 'iter'을 지정하였다. 이는 fmincon명령문에서 수치해가 변화하는 과정을 출력하기 위한 것이다. 또한, 옵션 Algorithm에 'sqp'를 지정하였다. 이는 디폴트인 'interior-point' 알고리즘 대신에 'sqp' 알고리즘을 사용하기 위한 것이다. 때때로 축차이차프로그래밍법(sequential quadratic programming method)이 내점법(interior-point method)보다 빠르거나 정확할 수 있다.

이 fmincon명령문을 실행한 결과는 다음과 같다.

```
                                                        Norm of  First-order
 Iter F-count            f(x) Feasibility  Steplength        step  optimality
    0       3    1.000000e+00    0.000e+00                          2.000e+00
    1      12    8.913011e-01    0.000e+00    1.176e-01    2.353e-01   1.107e+01
    2      22    8.047847e-01    0.000e+00    8.235e-02    1.900e-01   1.330e+01
    3      28    4.197517e-01    0.000e+00    3.430e-01    1.217e-01   6.153e+00
    4      31    2.733703e-01    0.000e+00    1.000e+00    5.254e-02   4.587e-01
    5      34    2.397111e-01    0.000e+00    1.000e+00    7.498e-02   3.029e+00
    6      37    2.036002e-01    0.000e+00    1.000e+00    5.960e-02   3.019e+00
    7      40    1.164353e-01    0.000e+00    1.000e+00    1.459e-01   1.058e+00
    8      43    1.161753e-01    0.000e+00    1.000e+00    1.754e-01   7.383e+00
    9      46    5.901601e-02    0.000e+00    1.000e+00    1.547e-02   7.278e-01
   10      49    4.533081e-02    2.898e-03    1.000e+00    5.393e-02   1.252e-01
   11      52    4.567454e-02    2.225e-06    1.000e+00    1.492e-03   1.679e-03
   12      55    4.567481e-02    4.406e-12    1.000e+00    2.095e-06   1.501e-05
   13      58    4.567481e-02    0.000e+00    1.000e+00    2.160e-09   1.511e-05

Local minimum possible. Constraints satisfied.
fmincon stopped because the size of the current step is less than
the default value of the step size tolerance and constraints are
satisfied to within the default value of the constraint tolerance.
<stopping criteria details>
x201 =  0.7864    0.6177
```

이 결과물에서 알 수 있듯이, Rosenbrock함수가 점 $(0.7864, 0.6177)$에서 극소값을 갖는다. ■

예제 6.5.21 MATLAB함수 fmincon.m을 사용하는 세 번째 MATLAB프로그램 USEfmincon103.m은 다음과 같다.

```
% ------------------------------------------------------------------------
%  Filename: USEfmincon103.m
%  Examples of MATLAB function fmincon.m Including Gradient
%  To find the minimum of a nonlinear funtion under constraints
%  From MATLAB Help
% ------------------------------------------------------------------------
function USEfmincon103
close all, clear all
diary USEfmincon103.txt
fun103 = @Rosenbrockwithgrad
x0 = [-1,2];
A = [];
b = [];
Aeq = [];
beq = [];
lb = [-2,-2];
ub = [2,2];
nonlcon = [];
options = optimoptions('fmincon','GradObj','on');
x103 = fmincon(fun103,x0,A,b,Aeq,beq,lb,ub,nonlcon,options)
diary off
end
% ------------------------------------------------------------------------
function [f,g] = Rosenbrockwithgrad(x)
% Calculate objective f
f = 100*(x(2) - x(1)^2)^2 + (1 -x(1))^2;
 if nargout > 1 % gradient required
   g = [-400*(x(2)-x(1)^2)*x(1)-2*(1-x(1)); 200*(x(2)-x(1)^2)];
 end
end
% ------------------------------------------------------------------------
```

이 fmincon명령문에서도 Rosenbrock함수의 극소값을 구한다. 초기값벡터와 부등식제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \quad \begin{bmatrix} -2 \\ -2 \end{bmatrix} \preceq \boldsymbol{x} \preceq \begin{bmatrix} 2 \\ 2 \end{bmatrix} \tag{1}$$

MATLAB함수 fmincon.m에 옵션을 지정하기 위해서 MATLAB함수 optimoptions.m를 사용하였고, 이 함수에서 옵션 GradObj에 'on'을 지정하였다. 이는 fmincon명령문에서 그래디언트벡터를 입력변수로 사용하기 위한 것이다. Rosenbrock함수의 그래디언트벡터는 다음과 같다.

$$\nabla \text{Rosenbrock} = \begin{bmatrix} -400\left[x_2 - x_1^2\right] x_1 - 2[1 - x_1] \\ 200\left[x_2 - x_1^2\right] \end{bmatrix}$$

이 fmincon명령문을 실행한 결과는 다음과 같다.

```
Local minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.
<stopping criteria details>
x103 =    0.999999920004335   0.999999839736623
```

이 결과물에서 알 수 있듯이, Rosenbrock함수가 점 $(1.0000, 1.0000)$에서 극소값을 갖는다. ■

예제 6.5.22 MATLAB함수 fmincon.m을 사용하는 네 번째 MATLAB프로그램 USEfmincon104.m은 다음과 같다.

```
% ------------------------------------------------------------------------
%  Filename: USEfmincon104.m
%  Examples of MATLAB function fmincon.m
%              using a Problem Structure
%  To find the minimum of a nonlinear funtion under constraints
%  From MATLAB Help
% ------------------------------------------------------------------------
function USEfmincon104
close all, clear all
diary USEfmincon104.txt
options = optimoptions('fmincon','Display','iter','Algorithm','sqp');
problem.options = options;
problem.solver = 'fmincon';
problem.objective = @(x) 100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
problem.x0 = [0,0];
problem.nonlcon = @unitdisk;
x104 = fmincon(problem)
diary off
end
% ------------------------------------------------------------------------
function [c,ceq] = unitdisk(x)
 c = x(1)^2 + x(2)^2 -1;
 ceq = [ ];
end
% ------------------------------------------------------------------------
```

이 fmincon명령문에서도 Rosenbrock함수의 극소값을 구한다. 초기값벡터와 부등식제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad x_1^2 + x_2^2 - 1 \le 0 \tag{1}$$

즉, 이 최적화문제는 예제 6.5.20의 최적화문제와 같다. 다만, 문제를 지정할 때 Problem 구조를 사용한다. 따라서, 예제 6.5.20과 동일한 결과를 출력한다. ■

예제 6.5.23 MATLAB함수 fmincon.m을 사용하는 다섯 번째 MATLAB프로그램 USEfmincon105.m은 다음과 같다.

```
% ------------------------------------------------------------------------
%  Filename: USEfmincon105.m
%  Examples of MATLAB function fmincon.m
%              obtaining the objective function value
%  To find the minimum of a nonlinear funtion under constraints
%  From MATLAB Help
% ------------------------------------------------------------------------
function USEfmincon105
close all, clear all
diary USEfmincon105.txt
fun105 = @(x)1+x(1)./(1+x(2))-3*x(1).*x(2) + x(2).*(1+x(1));
lb = [0,0];
ub = [1,2];
A = [];
b = [];
Aeq = [];
beq = [];
x0a = [0.5,1]
[x105a,fval105a] = fmincon(fun105,x0a,A,b,Aeq,beq,lb,ub)
x0b = x0a/5
[x105b,fval105b] = fmincon(fun105,x0b,A,b,Aeq,beq,lb,ub)
diary off
end
% ------------------------------------------------------------------------
```

이 MATLAB프로그램은 예제 6.5.19의 MATLAB프로그램 USEfmincon101.m에서 세 번째와 네 번째 fmincon명령문들을 다시 수행하는 것이다. 단, 극소값을 갖는 점 뿐아니라 구해진 극소값도 출력한다.

이 fmincon명령문을 실행한 결과는 당연히 예제 6.5.19의 결과와 동일하다. 첫 번째 fmincon명령문이 실행되면, 점 $(1.0000, 2.0000)$에서 목적함수가 극소값 -0.6667을 갖는다. 두 번째 fmincon명령문이 실행되면, 점 $(0.0000, 0.0000)$에서 목적함수가 극소값 1.0000을 갖는다. ■

예제 6.5.24 MATLAB함수 fmincon.m을 사용하는 여섯 번째 MATLAB프로그램 USEfmincon106.m은 다음과 같다.

```
% ------------------------------------------------------------------------
%  Filename: USEfmincon106.m
%  Examples of MATLAB function fmincon.m
%            for examing solution using extra outputs
%  To find the minimum of a nonlinear funtion under constraints
%  From MATLAB Help
% ------------------------------------------------------------------------
function USEfmincon106
close all, clear all
```

```
diary USEfmincon106.txt
Rosenbrock = @(x)100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
nonlcon = @unitdisk;
A = [];
b = [];
Aeq = [];
beq = [];
lb = [];
ub = [];
x0 = [0,0];
[x106,fval106,exitflag,output] ...
      = fmincon(Rosenbrock,x0,A,b,Aeq,beq,lb,ub,nonlcon)
diary off
end
% ------------------------------------------------------------------------
function [c,ceq] = unitdisk(x)
c = x(1)^2 + x(2)^2 -1;
ceq = [];
end
% ------------------------------------------------------------------------
```

이 fmincon명령문에서도 Rosenbrock함수의 극소값을 구한다. 초기값벡터와 부등식제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad x_1^2 + x_2^2 - 1 \le 0 \tag{1}$$

이 MATLAB프로그램은 근본적으로 예제 6.5.22와 같다. 다만, 출력변수들로 x106, fval106, exitflag, 그리고 output을 사용한다. 여기서 x106에는 극소점 벡터를 출력하고, fval106에는 이 극소점에서 극소값을 출력하고, exitflag의 값이 1이면 구한 값이 극소값임을 나타내고, output에는 여러 통계값들을 출력한다.

이 fmincon명령문을 실행한 결과는 다음과 같다.

```
Local minimum found that satisfies the constraints.
Optimization completed because the objective function is non-decreasing in
feasible directions, to within the default value of the function tolerance,
and constraints are satisfied to within the default value of the constraint tolerance.
<stopping criteria details>
x106 =   0.786415115581785   0.617698254793982
fval106 =   0.045674824758137
exitflag =     1
output =
         iterations: 24
          funcCount: 84
    constrviolation: 0
           stepsize: 6.916006580630123e-06
```

```
    algorithm: 'interior-point'
firstorderopt: 2.062365103694397e-08
 cgiterations: 4
      message: 'Local minimum found that satisfies the constraints.…'
```

■

예제 6.5.25 일곱 번째 MATLAB프로그램 USEfmincon107.m은 다음과 같다.

```
% ------------------------------------------------------------------------
%  Filename: USEfmincon107.m
%  Examples of MATLAB function fmincon.m
%           for examing solution using extra outputs
%  To find the minimum of a nonlinear funtion under constraints
%  From MATLAB Help
% ------------------------------------------------------------------------
function USEfmincon107
close all, clear all
diary USEfmincon107.txt
Rosenbrock = @(x)100*(x(2)-x(1)^2)^2 + (1-x(1))^2;
nonlcon = @unitdisk;
A = [];
b = [];
Aeq = [];
beq = [];
lb = [];
ub = [];
x0 = [0,0];
[x107,fval107,exitflag,output,lambda,grad,Hessian] ...
    = fmincon(Rosenbrock,x0,A,b,Aeq,beq,lb,ub,nonlcon)
diary off
end
% ------------------------------------------------------------------------
function [c,ceq] = unitdisk(x)
c = x(1)^2 + x(2)^2 -1;
ceq = [];
end
% ------------------------------------------------------------------------
```

이 MATLAB프로그램은 근본적으로 예제 6.5.24와 같다. 다만, 출력변수들로 lambda, grad, 그리고 Hessian을 추가했을 뿐이다. 여기서 lambda에는 Lagrange승수를 출력하고, grad에는 그래디언트벡터를, 그리고 Hessian에는 Hessian행렬을 출력한다.

이 fmincon명령문을 실행한 결과, 다음과 같이 이 새로운 변수들이 출력된다.

```
lambda =
          eqlin: [0x1 double]
       eqnonlin: [0x1 double]
        ineqlin: [0x1 double]
```

```
          lower: [2x1 double]
          upper: [2x1 double]
      ineqnonlin: 0.121494869061073


grad =
  -0.191090784035623
  -0.150094354059547


Hessian =
   1.0e+02 *
   4.972870562912335  -3.145573571860540
  -3.145573571860540   2.002378315539319
```

■

예제 6.5.26 MATLAB함수 fmincon.m을 사용하는 여덟 번째 MATLAB프로그램 USEfmincon108.m은 다음과 같다.

```
% ------------------------------------------------------------------------
%  Filename: USEfmincon108.m
%  Examples of MATLAB function fmincon.m
%  To find the minimum of a nonlinear funtion under constraints
% ------------------------------------------------------------------------
function USEfmincon108
close all, clear all
diary USEfmincon108.txt
display('(Example 1)')
A = [-1 -2 -2; 1 2 2];
b = [0; 72 ];
x0 = [10; 10; 10];
[x108a, fval108a] = fmincon(@myfun108a, x0, A, b)
clear all
display('(Example 2)')
x0 = [-1; -1];
[x108b, fval108b] ...
    = fmincon(@myfun108b,x0,[],[],[],[],[],[],@confun108b)
diary off
end
% ------------------------------------------------------------------------
function f = myfun108a(x)
  f = -x(1)*x(2)*x(3);
end
% ------------------------------------------------------------------------
function f = myfun108b(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
end
% ------------------------------------------------------------------------
function [c, ceq] = confun108b(x)
 c = [1.5+x(1)*x(2)-x(1)-x(2); -x(1)*x(2)-10];
 ceq = [];
```

```
33 end
34 % --------------------------------------------------------------------
```

첫 번째 fmincon 명령문에서 다음 함수의 극소값을 구한다.

$$f(\boldsymbol{x}) = -x_1 x_2 x_3 \tag{1}$$

초기값벡터와 부등식제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \begin{bmatrix} 10 \\ 10 \\ 10 \end{bmatrix}, \quad \begin{bmatrix} -1 & -2 & -2 \\ 1 & 2 & 2 \end{bmatrix} \boldsymbol{x} \preceq \begin{bmatrix} 0 \\ 72 \end{bmatrix} \tag{2}$$

이 fmincon 명령문을 실행한 결과는 다음과 같다.

```
x108a =
  24.000000002151218
  11.999999635045473
  12.000000362490027
fval108a =   -3.455999999599997e+03
```

이 결과물에서 알 수 있듯이, 목적함수 $f(\boldsymbol{x})$는 벡터 $[24, 12, 12]^t$ 에서 극소값 -3.4560×10^3 을 갖는다.

두 번째 fmincon 명령문에서 다음 함수의 극소값을 구한다.

$$f_2(\boldsymbol{x}) = \exp(x_1)\left[4x_1^2 + 2x_2^2 + 4x_1 x_2 + 2x_2 + 1\right] \tag{3}$$

또한, 초기값벡터와 비선형부등식제약조건은 각각 다음과 같다.

$$\boldsymbol{x}_0 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \quad \begin{bmatrix} 1.5 + x_1 x_2 - x_1 - x_2 \\ -x_1 x_2 - 10 \end{bmatrix} \preceq \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4}$$

이 fmincon 명령문을 실행한 결과는 다음과 같다.

```
x108b =
  -9.547345799620329
   1.047408300939684
fval108b =   0.023551461737953
```

이 결과물에서 알 수 있듯이, 목적함수 $f(\boldsymbol{x})$는 벡터 $[-9.5473,\ 1.0474]^t$ 에서 극소값 0.0236을 갖는다. ■

예제 6.5.27 Python패키지 scipy의 Optimize모듈의 함수 minimize를 사용해서 제약조건이 있는 최소화문제를 다루는 예로서 다음 Python프로그램 USEminimize101.PY를 실행하자.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 14 01:54:14 2017
https://docs.scipy.org/doc/scipy-0.18.1/reference/tutorial/optimize.html
"""

import numpy as np
from scipy import optimize

def func(x, sign=1.0):
    return sign*(2*x[0]*x[1] + 2*x[0] - x[0]**2 - 2*x[1]**2)
def func_deriv(x, sign=1.0):
    dfdx0 = sign*(-2*x[0] + 2*x[1] + 2)
    dfdx1 = sign*(2*x[0] - 4*x[1])
    return np.array([ dfdx0, dfdx1 ])

cons = ({'type': 'eq',
         'fun' : lambda x: np.array([x[0]**3 - x[1]]),
         'jac' : lambda x: np.array([3.0*(x[0]**2.0), -1.0])},
        {'type': 'ineq',
         'fun' : lambda x: np.array([x[1] - 1]),
         'jac' : lambda x: np.array([0.0, 1.0])})
res = optimize.minimize(func, [-1.0,1.0], args=(-1.0,), jac=func_deriv, \
                        constraints=cons, method='SLSQP', \
                        options={'disp': True})
print(res)
print(res.x)

# End of program
```

이 Python프로그램에서는 다음 함수의 극소값을 구한다.

$$f(\boldsymbol{x}) = -\left[2x_1x_2 + 2x_1 - x_1^2 - 2x_2^2\right] \tag{1}$$

등호제약조건과 부등식제약조건은 다음과 같다.

$$x_1^3 - x_2 = 0 \tag{2}$$

$$x_2 \geq 1 \tag{3}$$

또한 초기값벡터는 $[-1.0,\ 1.0]$이다.

이 Python프로그램을 실행한 결과는 다음과 같다.

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: -1.00000018311
            Iterations: 9
            Function evaluations: 14
            Gradient evaluations: 9
     fun: -1.0000001831052137
     jac: array([-1.99999982,  1.99999982,  0.        ])
 message: 'Optimization terminated successfully.'
    nfev: 14
     nit: 9
    njev: 9
  status: 0
 success: True
       x: array([ 1.00000009,  1.        ])
```

이 결과물에서 알 수 있듯이, 목적함수 $f(\boldsymbol{x})$는 점 $(1,1)$ 에서 극소값 -1.0000을 갖는다. ■

예제 6.5.28 Python패키지 scipy의 Optimize모듈의 함수 minimize를 사용해서 제약조건이 있는 최소화문제를 다루는 예로서 다음 Python프로그램 USEminimize102.PY를 실행하자.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 14 01:54:14 2017
http://stackoverflow.com/questions/25735301/looping-through-dataframe-and-
    creating-a-function-for-optimization
"""

import numpy as np
from scipy.optimize import minimize

"""define power and coefficients"""
power = 0.6
coefficient = 5.6
"""define objective function"""
def func(x,sign=1.0):
    return sign*sum(coefficient*(x[0:]**power))
""" define constraints"""
cons = ({'type': 'ineq', 'fun': lambda x:  x[0] - 0.2 * x[1] - 2},
        {'type': 'ineq', 'fun': lambda x: -x[2] - 0.2 * x[1] + 1},
        {'type': 'ineq', 'fun': lambda x: -x[0] - 0.2 * x[1] + 2})
""" define bounds"""
bnds = ((0.1, 1.3), (0.1, 1.3), (0.1,1.3))
"""initial values of the variables"""
x0 = np.array([1.1, 3.9,5.6])
"""run the optimization  algorithm"""
resul = minimize(func, x0, method='SLSQP',bounds = bnds,constraints=cons,
```

```
                  options={'disp': True})

"""print the results"""
print(resul)
print(resul.x)

# End of program
```

이 Python프로그램에서는 다음 함수의 극소값을 구한다.

$$f(\boldsymbol{x}) = 5.6\left[x_1^{0.6} + x_2^{0.6} + x_3^{0.6}\right] \tag{1}$$

부등식제약조건들은 다음과 같다.

$$+ x_1 - 0.2x_2 = 2 \tag{2}$$

$$- x_3 - 0.2x_2 = -1 \tag{3}$$

$$- x_1 - 0.2x_2 = -2 \tag{4}$$

$$0.1 \le x_1 \le 1.3, \;\; 0.1 \le x_2 \le 1.3, \;\; 0.1 \le x_3 \le 1.3 \tag{5}$$

또한 초기값벡터는 $[1.1,\, 3.9,\, 5.6]$이다.

이 Python프로그램을 실행한 결과는 다음과 같다.

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 9.36803120133
            Iterations: 12
            Function evaluations: 80
            Gradient evaluations: 8
     fun: 9.368031201334599
     jac: array([ 3.02525473,  8.43993807,  8.43993819,  0.        ])
 message: 'Optimization terminated successfully.'
    nfev: 80
     nit: 12
    njev: 8
  status: 0
 success: True
       x: array([ 1.3,  0.1,  0.1])

[ 1.3  0.1  0.1]
```

이 결과물에서 알 수 있듯이, 이 목적함수 $f(\boldsymbol{x})$ 는 점 $(1.3,\, 0.1,\, 0.1)$ 에서 극소값 9.3680 을 갖는다. ■

제6.6절 능형회귀와 LASSO

6.6.1 편의와 분산

다음과 같은 선형회귀모형을 생각해보자.

$$\boldsymbol{y} = X\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{6.6.1}$$

여기서 벡터들 $\boldsymbol{y}\,(\in \mathbb{R}^n)$, $\boldsymbol{\epsilon}\,(\in \mathbb{R}^n)$ 과 $\boldsymbol{\beta}\,(\in \mathbb{R}^p)$, 그리고 행렬 $X\,(\in \mathbb{R}^{n\times p})$ 는 각각 다음과 같다.

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}, \quad X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \tag{6.6.2}$$

오차항벡터 $\boldsymbol{\epsilon}$ 의 원소들 $\epsilon_1,\, \epsilon_2,\, \cdots,\, \epsilon_n$ 은 서로 독립이며 평균이 0 이고 분산이 σ^2 인 확률분포를 따른다고 가정하자. 또한, 행렬 X 의 제 j 번째 행을 $\boldsymbol{x}_j^t$ 로 표기하자. 즉, 다음과 같이 벡터 $\boldsymbol{x}_j$ 정의하자.

$$\boldsymbol{x}_j \doteq [x_{j1},\, x_{j2},\, \cdots,\, x_{jp}]^t \tag{6.6.3}$$

식 $(6.6.1)$ 에서 알 수 있듯이, 다음 식이 성립한다.

$$X^t\boldsymbol{y} = X^tX\boldsymbol{\beta} + X^t\boldsymbol{\epsilon} \tag{6.6.4}$$

행렬 X 와 오차항벡터 $\boldsymbol{\epsilon}$ 는 정보를 공유하지 않으므로, 다음 식이 성립한다고 가정하자.

$$X^t\boldsymbol{\epsilon} = \mathbf{0} \tag{6.6.5}$$

식 (6.6.5)가 L^2-공간에서 성립함을 증명할 수 있다. 식 (6.6.4)과 식 (6.6.5)에서 알 수 있듯이, 다음과 같은 모수벡터 $\boldsymbol{\beta}$의 추정벡터를 생각할 수 있다.

$$\hat{\boldsymbol{\beta}} = \left[X^t X\right]^- X^t \boldsymbol{y} \tag{6.6.6}$$

여기서 A^-는 행렬 A의 일반화역행렬(generalized inverse)이다. 만약 $\text{rank}(X) = p$이면, 식 $\left[X^t X\right]^- = \left[X^t X\right]^{-1}$가 성립한다. 이 경우에 $\hat{\boldsymbol{\beta}}$는 $\boldsymbol{\beta}$의 최소제곱추정량이다. 오차항들 $\epsilon_1, \epsilon_2, \cdots, \epsilon_n$이 정규확률분포를 따른다는 가정 하에서 $\hat{\boldsymbol{\beta}}$는 최우추정량이다. 또한, 다음 식들이 성립함을 쉽게 증명할 수 있다.

$$E\left(\hat{\boldsymbol{\beta}}\right) = \boldsymbol{\beta} \tag{6.6.7}$$

$$Var\left(\hat{\boldsymbol{\beta}}\right) = \sigma^2 [X^t X]^{-1} \tag{6.6.8}$$

식 (6.6.7)에서 알 수 있듯이, 최소제곱추정량 $\hat{\boldsymbol{\beta}}$는 불편추정량이다. 또한, 만약 $\text{rank}(X) = p$이면, 식 (6.6.7)과 식 (6.6.8)을 사용해서 $\hat{\boldsymbol{\beta}}$가 최량불편선형추정량(Best Unbiased Linear Estimator: BLUE)임을 증명할 수 있다. 이 성질을 Gauss-Markov성질이라 부른다. 원래는 이 추정량을 BLUE라 부르나 본저자는 BULE라고 불러야한다고 생각한다. 최량불편선형추정량이란 좋은 추정량이라고 해석할 수 있으므로, 식 $\text{rank}(X) = p$이 성립하는 경우에 최소제곱추정량이 널리 사용되고 있다. 그러나, 만약 식 $\text{rank}(X) < p$이 성립하면, 식 (6.6.8)에서 알 수 있듯이 최소제곱추정량 $\hat{\boldsymbol{\beta}}$의 분산은 유한이 아니다. 즉, $\det(X^t X)$가 0에 가깝거나 사전에 p가 정해지지 않은 경우, 최소제곱추정량이 반드시 좋은 추정량이라고 할 수 없다. 이러한 경우에는 추정량의 분산(variance)이 너무 커지므로, 추정량의 편의(bias)를 약간 허용해서 추정량의 분산을 감소시키는 새로운 추정량을 사용하는 것이 좋을 것이다. 즉, 편의와 분산을 교환하는 추정량을 사용하는 것이 좋을 것이다.

모수벡터 $\boldsymbol{\beta}$의 어떤 추정량 $\tilde{\boldsymbol{\beta}}$의 편의와 분산을 타협하는 기준은 무엇일까? 자주 이용되는 기준의 하나는 다음과 같이 정의되는 최소제곱오차(mean squared error: MSE)를 사용하는 것이다.

$$MSE(\tilde{\boldsymbol{\beta}}) \doteq E\left(\|\tilde{\boldsymbol{\beta}} - \boldsymbol{\beta}\|^2\right) = E\left(\left[\tilde{\boldsymbol{\beta}} - \boldsymbol{\beta}\right]^t \left[\tilde{\boldsymbol{\beta}} - \boldsymbol{\beta}\right]\right) \tag{6.6.9}$$

최소제곱추정량 $\hat{\boldsymbol{\beta}}$의 최소제곱오차는 다음과 같다.

$$\begin{aligned} MSE(\hat{\boldsymbol{\beta}}) &= E\left(\left[\hat{\boldsymbol{\beta}}-\boldsymbol{\beta}\right]^t\left[\hat{\boldsymbol{\beta}}-\boldsymbol{\beta}\right]\right) \\ &= E\left(\mathrm{Tr}\left(\left[\hat{\boldsymbol{\beta}}-\boldsymbol{\beta}\right]\left[\hat{\boldsymbol{\beta}}-\boldsymbol{\beta}\right]^t\right)\right) = \sigma^2\mathrm{Tr}\left(\left[X^tX\right]^{-1}\right) \end{aligned} \tag{6.6.10}$$

여기서 세 번째 등호는 식 (6.6.8)에 의해서 성립한다.

모수벡터 $\boldsymbol{\beta}$의 어떤 추정량 $\tilde{\boldsymbol{\beta}}$는 주어진 데이터세트 $\{(X,\boldsymbol{y})\}$로부터 추정된 것이다. 비록 이 추정량 $\tilde{\boldsymbol{\beta}}$가 주어진 데이터세트를 잘 나타낸다고해도, 새로운 데이터세트 $\{(X^*,\boldsymbol{y}^*)\}$를 잘 나타낸다고 단정할 수는 없다. 즉, 예측오차(prediction error: PE)의 노름 $\left\|\boldsymbol{y}^*-X^*\tilde{\boldsymbol{\beta}}\right\|$가 충분히 작은 값이라는 보장은 없다. 만약 $\tilde{\boldsymbol{\beta}}$가 좋은 추정량이라면, 평균적으로 작은 예측오차를 발생시켜야 할 것이다. 어떤 목적점(target point) $\boldsymbol{x}_0 \doteq [x_{01},\, x_{02},\, \cdots,\, x_{0p}]^t$에서 예측오차는 다음과 같다.

$$PE(\boldsymbol{x}_0;\,\tilde{\boldsymbol{\beta}}) \doteq E_{y|\,\boldsymbol{x}_0}\left(\left[y-\boldsymbol{x}_0^t\tilde{\boldsymbol{\beta}}\right]^2\right) \tag{6.6.11}$$

다음 식이 성립함을 쉽게 증명할 수 있다.

$$PE(\boldsymbol{x}_0;\,\tilde{\boldsymbol{\beta}}) = \sigma^2 + \left[Bias\left(\boldsymbol{x}_0^t\tilde{\boldsymbol{\beta}}\right)\right]^2 + Var\left(\boldsymbol{x}_0^t\tilde{\boldsymbol{\beta}}\right) \tag{6.6.12}$$

식 (6.6.12)의 분해를 편의분산교환(bias-variance tradeoff)이라 부른다. 모형에 설명변수를 추가하면, 모형은 국지구조(local structure)를 잘 나타내나 모형 전체의 분산은 커진다. 즉, 추정식 $\boldsymbol{x}^t\tilde{\boldsymbol{\beta}}$의 편의는 작아지나 분산이 커진다. 이러한 현상을 완화시키기 위해서 추정량 $\tilde{\boldsymbol{\beta}}$에 약간의 편의를 허용함으로써 분산을 현저하게 감소시킬 수 있다면, 예측오차 관점에서 이 추정량은 좋은 것이다.

6.6.2 선형회귀모형과 능형회귀

식 (6.6.1)의 선형회귀모형을 다시 생각해보자. 만약 $\det(X^tX)$가 0에 가까우면, 최소제곱추정량 $\hat{\boldsymbol{\beta}}$의 분산이 아주 커져서 이 추정량을 신뢰하기 어렵다. 이러한 경우 모수벡터 $\boldsymbol{\beta}$의 노름이 지나치게 커지는 것을 방지하기 위해서, 다음 최적화문제를 살펴보자.

$$\underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\text{Minimize}} \sum_{j=1}^{n}\left[y_j-\boldsymbol{x}_j^t\boldsymbol{\beta}\right]^2 \quad \text{subject to} \quad \sum_{i=1}^{p}\beta_i^2 \le r. \tag{6.6.13}$$

일반적으로 다음 식들이 성립한다고 가정한다.

$$E(y_j) = 0, \quad (j = 1, 2, \cdots, n) \tag{6.6.14}$$

$$\frac{1}{n}\sum_{j=1}^{n} x_{i,j} = 0, \quad \frac{1}{n-1}\sum_{j=1}^{n}\left[x_{i,j} - \frac{1}{n}\sum_{k=1}^{n} x_{i,k}\right]^2 = 1, \quad (i = 1, 2, \cdots, p) \tag{6.6.15}$$

식 (6.6.13)의 최적화문제를 풀기 위해서 다음과 같이 Lagrange함수를 정의하자.

$$PRSS_2(\boldsymbol{\beta}) \doteq \sum_{j=1}^{n}\left[y_j - \boldsymbol{x}_j^t\boldsymbol{\beta}\right]^2 + \lambda\sum_{i=1}^{p}\beta_i^2. \tag{6.6.16}$$

식 (6.6.16)의 Lagrange함수 $PRSS_2(\boldsymbol{\beta})$를 벌칙잔차제곱합(penalized residual sum of squares)이라 부르고, λ를 능형모수(ridge parameter)라 부른다. 이 $PRSS_2(\boldsymbol{\beta})$의 아래첨자 2는 l_2-공간에서 노름을 의미한다. 원래 식 (6.6.13)의 능형제약조건(ridge constraint)이 부등식이므로 Karush-Kuhn-Tucker법을 써야하나, 문제를 간단히 하기 위해서 Lagrange승수법을 적용하기로 하자. 벌칙잔차제곱합 $PRSS_2(\boldsymbol{\beta})$는 $\boldsymbol{\beta}$의 볼록함수이므로, 최적화문제 (6.6.13)은 일의적인 해를 갖는다. 식 (6.6.16)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{\partial PRSS_2(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} = 2X^t\left[\boldsymbol{y} - X\boldsymbol{\beta}\right] + 2\lambda\boldsymbol{\beta} \tag{6.6.17}$$

즉, 최적화문제 (6.6.13)의 해는 다음과 같다.

$$\hat{\boldsymbol{\beta}}_\lambda^R = \left[X^tX + \lambda I_p\right]^{-1}X^t\boldsymbol{y} \tag{6.6.18}$$

설사 X^tX가 특이행렬이라 하더라도, 행렬 $X^tX + \lambda I_p$는 정칙행렬이다. 따라서, 양수 λ에 대해서 해 $\hat{\boldsymbol{\beta}}_\lambda^R$가 항상 존재한다.

다음 식들이 성립한다.

$$\hat{\boldsymbol{\beta}}_\lambda^R = \left[X^tX + \lambda I_p\right]^{-1}X^tX\left[X^tX\right]^{-1}X^t\boldsymbol{y} = \left[X^tX + \lambda I_p\right]^{-1}X^tX\boldsymbol{\beta} \tag{6.6.19}$$

즉, 능형회귀추정량과 최소제곱추정량은 다음 식을 만족한다.

$$\hat{\boldsymbol{\beta}}_\lambda^R = \left[I_p + \lambda\left[X^tX\right]^{-1}\right]^{-1}\hat{\boldsymbol{\beta}} \tag{6.6.20}$$

식 (6.6.20)에서 알 수 있듯이, 다음 식들이 성립한다.

$$E\left(\hat{\boldsymbol{\beta}}_{\lambda}^{R}\right)=\left[I_{p}+\lambda\left[X^{t}X\right]^{-1}\right]^{-1}\boldsymbol{\beta}\neq\boldsymbol{\beta} \tag{6.6.21}$$

즉, 능형회귀추정량은 편의가 있는 추정량이다.

주어진 데이터세트에 관찰점들을 추가하는 것으로 능형회귀추정량을 설명할 수 있다. 다음 식이 성립한다.

$$PRSS_{2}(\boldsymbol{\beta})=\sum_{j=1}^{n}\left[y_{j}-\boldsymbol{x}_{j}^{t}\boldsymbol{\beta}\right]^{2}+\sum_{j=1}^{p}\left[0-\sqrt{\lambda}\beta_{i}\right]^{2}. \tag{6.6.22}$$

다음 행렬을 정의하자.

$$X_{p}\doteq\sqrt{\lambda}\,\mathrm{diag}\left(1,\,-1,\,1,-1,\cdots,[-1]^{p}\right) \tag{6.6.23}$$

여기서 1과 -1을 교대로 사용한 것은 가정 (6.6.15)를 따르기 위한 것이다. 또한 다음 행렬과 벡터를 정의하자.

$$X_{\lambda}\doteq\begin{bmatrix}X\\X_{p}\end{bmatrix},\quad\boldsymbol{y}_{\lambda}\doteq\begin{bmatrix}\boldsymbol{y}\\\mathbf{0}_{p}\end{bmatrix} \tag{6.6.24}$$

여기서 $\mathbf{0}_{p}$는 차원이 p인 영벡터이다. 이 확장된 데이터세트(augmented dataset)에 해당하는 최소제곱추정량은 다음과 같다.

$$\left[X_{\lambda}^{t}X_{\lambda}\right]^{-1}X_{\lambda}^{t}\boldsymbol{y}_{\lambda}=\left[X^{t}X+\lambda I_{p}\right]^{-1}X^{t}\boldsymbol{y} \tag{6.6.25}$$

즉, 다음 식이 성립한다.

$$\hat{\boldsymbol{\beta}}_{\lambda}^{R}=\left[X_{\lambda}^{t}X_{\lambda}\right]^{-1}X_{\lambda}^{t}\boldsymbol{y}_{\lambda} \tag{6.6.26}$$

오차항벡터 $\boldsymbol{\epsilon}$의 원소들 $\epsilon_{1},\,\epsilon_{2},\,\cdots,\,\epsilon_{n}$이 서로 독립이고 정규확률분포 $\mathcal{N}\left(0,\sigma^{2}\right)$를 따른다고 하자. 또한, 모수벡터 $(\boldsymbol{\beta},\sigma^{2})$의 사전확률분포가 다음과 같다고 하자.

$$\boldsymbol{\beta}|\sigma^{2}\overset{d}{\sim}\mathcal{N}\left(\mathbf{0},\frac{\sigma^{2}}{\lambda}I_{p}\right),\quad\sigma^{2}\overset{d}{\sim}\mathcal{IG}(\alpha_{0},\beta_{0}) \tag{6.6.27}$$

여기서 $\mathcal{IG}(\alpha,\beta)$는 모수들이 α와 β인 역감마확률분포이다. 우도함수는 다음과 같다.

$$f_{\boldsymbol{y}}\left(\boldsymbol{y}\,|\,X,\boldsymbol{\beta},\sigma^2\right)\propto\sigma^{-n}\exp\left(-\frac{1}{2\sigma^2}\left[\boldsymbol{y}-X\boldsymbol{\beta}\right]^t\left[\boldsymbol{y}-X\boldsymbol{\beta}\right]\right)\tag{6.6.28}$$

따라서 사후확률밀도함수는 다음과 같다.

$$\begin{aligned}&f_{\boldsymbol{\beta},\sigma^2}\left(\boldsymbol{\beta},\sigma^2\,|\,\boldsymbol{y},X\right)=f_{\boldsymbol{y}}\left(\boldsymbol{y}\,|\,X,\boldsymbol{\beta},\sigma^2\right)f_{\boldsymbol{\beta}}\left(\boldsymbol{\beta}\,|\,\sigma^2\right)f_{\sigma^2}\left(\sigma^2\right)\\&\propto\ \sigma^{-n}\exp\left(-\frac{1}{2\sigma^2}\left[\boldsymbol{y}-X\boldsymbol{\beta}\right]^t\left[\boldsymbol{y}-X\boldsymbol{\beta}\right]\right)\\&\times\sigma^{-p}\exp\left(-\frac{\lambda}{2\sigma^2}\boldsymbol{\beta}^t\boldsymbol{\beta}\right)\times\sigma^{-2[\alpha_0+1]}\exp\left(-\frac{\beta_0}{2\sigma^2}\right)\end{aligned}\tag{6.6.29}$$

다음 식이 성립한다.

$$\begin{aligned}&\left[\boldsymbol{y}-X\boldsymbol{\beta}\right]^t\left[\boldsymbol{y}-X\boldsymbol{\beta}\right]+\lambda\boldsymbol{\beta}^t\boldsymbol{\beta}\\&=\ \boldsymbol{y}^t\boldsymbol{y}-\boldsymbol{y}^tX\left[X^tX+\lambda I_p\right]^{-1}X^t\boldsymbol{y}+\left[\boldsymbol{\beta}-\hat{\boldsymbol{\beta}}_\lambda^R\right]^t\left[X^tX+\lambda I_p\right]\left[\boldsymbol{\beta}-\hat{\boldsymbol{\beta}}_\lambda^R\right]\end{aligned}\tag{6.6.30}$$

식 (6.6.29)과 식 (6.6.30)에서 알 수 있듯이, 다음 식이 성립한다.

$$f_{\boldsymbol{\beta},\sigma^2}\left(\boldsymbol{\beta},\sigma^2\,|\,\boldsymbol{y},X\right)\propto g_{\boldsymbol{\beta}}\left(\boldsymbol{\beta}\,|\,\sigma^2,\boldsymbol{y},X\right)\times g_{\sigma^2}\left(\sigma^2\,|\,\boldsymbol{y},X\right)\tag{6.6.31}$$

여기서 $g_{\boldsymbol{\beta}}\left(\boldsymbol{\beta}\,|\,\sigma^2,\boldsymbol{y},X\right)$는 다음과 같다.

$$g_{\boldsymbol{\beta}}\left(\boldsymbol{\beta}\,|\,\sigma^2,\boldsymbol{y},X\right)\propto\ \exp\left(-\frac{1}{2\sigma^2}\left[\boldsymbol{\beta}-\hat{\boldsymbol{\beta}}_\lambda^R\right]^t\left[X^tX+\lambda I_p\right]\left[\boldsymbol{\beta}-\hat{\boldsymbol{\beta}}_\lambda^R\right]\right)\tag{6.6.32}$$

따라서, 사후평균(posterior mean)은 다음과 같다.

$$E\left(\boldsymbol{\beta}\,|\,\sigma^2,\boldsymbol{y},X\right)=\hat{\boldsymbol{\beta}}_\lambda^R\tag{6.6.33}$$

즉, 회귀계수벡터 $\boldsymbol{\beta}$의 베이지안추정량은 능형회귀계수벡터 $\hat{\boldsymbol{\beta}}_\lambda^R$이다.

능형회귀계수벡터 $\hat{\boldsymbol{\beta}}_\lambda^R=\left[X^tX+\lambda I_p\right]^{-1}X^t\boldsymbol{y}$를 계산하기 위해서 Gauss소거법을 적용하면 많은 계산량을 필요로 한다. 좀 더 빠르게 능형회귀계수벡터를 계산하기 위해서 행렬 X에 특이값분해(singular value decomposition: SVD)를 적용하기로 하자. 다음 식들을 만족하는

$n \times n$ 직교행렬(orthogonal matrix) U, $n \times p$ 행렬 D, 그리고 $p \times p$ 직교행렬 V가 존재한다.

$$X = UDV^t \tag{6.6.34}$$

$$U^tU = I_n, \quad V^tV = I_p, \quad D \doteq \begin{bmatrix} d_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & d_2 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & d_p \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \doteq \begin{bmatrix} D_p \\ O_{[n-p]\times p} \end{bmatrix} \tag{6.6.35}$$

여기서 d_i들은 다음 식들을 만족한다고 가정하자.

$$d_1 \geq d_2 \geq \cdots \geq d_p \tag{6.6.36}$$

다음 식들이 성립한다.

$$X^tX = \left[UDV^t\right]^t \left[UDV^t\right] = VD^tDV^t \tag{6.6.37}$$

$$\left[X^tX + \lambda I_p\right]^{-1} = \left\{V\left[D^tD + \lambda I_p\right]V^t\right\}^{-1} = V\left[D^tD + \lambda I_p\right]^{-1}V^t \tag{6.6.38}$$

$$\hat{\boldsymbol{\beta}}_\lambda^R = V\left[D^tD + \lambda I_p\right]^{-1}D^tU^t\boldsymbol{y} \tag{6.6.39}$$

따라서 다음 식이 성립한다.

$$\hat{\boldsymbol{\beta}}_\lambda^R = V\left[\text{diag}\left(\frac{d_1}{d_1^2+\lambda}, \frac{d_2}{d_2^2+\lambda}, \cdots, \frac{d_p}{d_p^2+\lambda}\right), O_{p\times[n-p]}\right]U^t\boldsymbol{y} \tag{6.6.40}$$

능형회귀적합벡터 $\hat{\boldsymbol{y}}_\lambda^R$는 다음 식들을 만족한다.

$$\hat{\boldsymbol{y}}_\lambda^R = X\hat{\boldsymbol{\beta}}_\lambda^R = UD\left[\text{diag}\left(\frac{d_1}{d_1^2+\lambda}, \frac{d_2}{d_2^2+\lambda}, \cdots, \frac{d_p}{d_p^2+\lambda}\right), O_{p\times[n-p]}\right]U^t\boldsymbol{y} \tag{6.6.41}$$

즉, 다음 식이 성립한다.

$$\hat{\boldsymbol{y}}_\lambda^R = \sum_{i=1}^{p}\left[\boldsymbol{u}_i\frac{d_i^2}{d_i^2+\lambda}\boldsymbol{u}_i^t\right]\boldsymbol{y} \tag{6.6.42}$$

여기서 $\boldsymbol{u}_i$는 행렬 U의 제i번째 열벡터이다. 벡터 $X\boldsymbol{v}_i = d_i\boldsymbol{u}_i$는 제$i$번째 주성분벡터(principal component vector)이다. 식 (6.6.42)에서 알 수 있듯이, 능형회귀식은 작은 특성값 d_i에 대응하는 계수를 축소시켜서(shrink), 종속벡터 $\boldsymbol{y}$를 큰 특성값 d_i에 대응하는 특성벡터 $\boldsymbol{u}_i$로 근사 사영시킨다(project).

정의 6.6.1

벡터들 $\boldsymbol{y}$와 $\hat{\boldsymbol{y}}$에 대해서, 다음 식을 만족하는 S를 스무더행렬(smoother matrix)이라 한다.

$$\hat{\boldsymbol{y}} = S\boldsymbol{y}$$

선형회귀모형에서 모자행렬(hat matrix) $H \doteq X[X^tX]^{-1}X^t$는 스무더행렬로서, 다음 식을 만족한다.

$$\text{rank}(H) = \text{Tr}(H) \tag{6.6.43}$$

이 $\text{rank}(H)$는 모형의 자유도(degrees of freedom)이고, $\text{Tr}(H)$는 행렬 H의 트레이스이다. 따라서, $\text{Tr}(S)$를 스무더행렬 S의 효율적 자유도(effective degrees of freedom) 또는 효율적 모수 개수(effective number of parameters)라 하고 $df(S)$로 표기한다. 능형회귀적합벡터 $\hat{\boldsymbol{y}}_\lambda^R$는 다음 식들을 만족한다.

$$\hat{\boldsymbol{y}}_\lambda^R = X\hat{\boldsymbol{\beta}}_\lambda^R = X\left[X^tX + \lambda I_p\right]^{-1}X^t\boldsymbol{y} \tag{6.6.44}$$

따라서 능형회귀모형의 스무더는 다음과 같다.

$$S_\lambda \doteq X\left[X^tX + \lambda I_p\right]^{-1}X^t \tag{6.6.45}$$

또한, 능형회귀모형의 효율적 자유도는 다음과 같다.

$$df(S_\lambda) = \text{Tr}(S_\lambda) = \text{Tr}\left(X\left[X^tX + \lambda I_p\right]^{-1}X^t\right) = \sum_{i=1}^{p}\frac{d_i^2}{d_i^2+\lambda} \tag{6.6.46}$$

이 효율적 자유도 $df(S_\lambda)$는 λ의 단조감소함수이다. 능형모수 λ는 수축모수(shrinkage parameter)이므로, 효율적 자유도 $df(S_\lambda)$는 확장모수라고 할 수 있다. 다음 식들이 성립한다.

$$\lim_{\lambda \downarrow 0} = p, \quad \lim_{\lambda \uparrow \infty} = 0 \tag{6.6.47}$$

즉, λ가 0으로 수렴하면, 능형회귀추정량은 최소제곱추정량으로 수렴한다. 또한, λ가 ∞로 발산하면, 능형회귀모형은 상수모형으로 수렴한다. 효율적 자유도 $df(S_\lambda)$가 작아지면, 회귀계수들의 개수가 감소하고 정규화(regularization) 정도가 증가한다.

예제 6.6.1 MATLAB에 내장된 데이터세트 acetylene.mat는 화학실험에서 얻은 것으로, 설명변수들 x1, x2, x3와 종속변수 y로 구성되어 있다. 이 데이터세트를 능형회귀분석하기 위해서, 다음 MATLAB프로그램 RidgeRegression101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: RidgeRegression201.m
%  Ridge Regression
%  https://kr.mathworks.com/help/stats/ridge.html
%------------------------------------------------------------------------
clear all, close all, format long

% Dataset
load acetylene
X = [x1 x2 x3];
D = x2fx(X,'interaction');
          % Convert predictor matrix to design matrix
D(:,1) = [];  % No constant term
% standardize and center
Dstan = (D-repmat(mean(D,1),size(D,1),1)) ...
         ./repmat(std(D,0,1),size(D,1),1);
   % Dstan = (D-dum1*mean(D))./(dum1*std(D))
% center ys
yStan = y - mean(y);
   % yStan = y-dum1*mean(y)
whos

% Least squares method
bLS = regress(y, D)
bLSS = regress(yStan,Dstan)
bLSs = ridge(y,D,0)
check1 = norm(bLSS-bLSs)
fitLS = D*bLSs;

% Plotting Data
subplot(2,2,1)
plot(x1,x2,'r.','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('x1'); ylabel('x2'); grid on; axis square
subplot(2,2,2)
plot(x1,x3,'r.','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('x1'); ylabel('x3'); grid on; axis square
subplot(2,2,3)
```

```
plot(x2,x3,'r.','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('x2'); ylabel('x3'); grid on; axis square
subplot(2,2,4)
plot(y,fitLS,'b.','LineWidth',3)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('y'); ylabel('fits'); grid on; axis square
axis([ 0 60 -3*10^6 0 ])
saveas(gcf,'RidgeRegression101a.jpg')

% Ridge Regression
bLambda002 = ridge(y,D,0.002)

% Ridge Trace
lambdaa = 0:1e-5:5e-3;
bLambda = ridge(y,D,lambdaa);
figure
plot(lambdaa,bLambda,'LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
ylim([-100 100])
grid on
xlabel('Ridge Parameter \lambda')
ylabel('Standardized Coefficient')
title('{\bf Ridge Trace}')
legend('x1','x2','x3','x1x2','x1x3','x2x3')
saveas(gcf,'RidgeRegression101b.jpg')
% End of Program
% ------------------------------------------------------------------------
```

MATLAB함수 x2fx.m을 사용해서 데이터를 확장한다. 이 MATLAB함수 x2fx.m의 옵션 interaction을 지정하면, 교차항들 x4 = x1*x2, x5 = x1*x3, 그리고 x6 = x2*x3를 데이터세트에 추가한다. 또한, 다음과 같이 종속변수와 설명변수들을 표준화한다.

$$y_j^s = y_j - \bar{y}, \quad x_{j,k}^s = \frac{x_{j,k} - \bar{x}_k}{\sqrt{\frac{1}{n-1}\sum_{j=1}^{n}[x_{j,k} - \bar{x}_k]^2}}, \quad (k = 1, 2, \cdots, 6) \tag{1}$$

MATLAB명령문 'bLS = regress(y,D)'를 실행하면, 다음과 같은 선형회귀모형의 최소제곱추정식을 출력한다.

$$\begin{aligned} y = {} & 0.0462x_1 - 5.7811x_2 + 1267.2049x_3 + 0.00405x_1x_2 \\ & - 1.6141x_1x_3 + 32.7251x_2x_3 + \hat{\epsilon} \end{aligned} \tag{2}$$

MATLAB명령문 'bLS = regress(yStan,Dstan)'을 실행하면, 다음과 같은 선형회귀모형의 최소제곱추정식을 출력한다.

$$\begin{aligned} y^s = {} & 21.2204x_1^s + 82.3283x_2^s + 44.1077x_3^s - 81.4237x_4^s \\ & - 40.8794x_5^s - 3.5940x_6^s + \hat{\epsilon}^s \end{aligned} \tag{3}$$

MATLAB명령문 'bLS = ridge(y,D,0)'를 실행하면, 식 (3)과 동일한 결과를 출력한다.

MATLAB명령문 'bLambda002 = ridge(y,D,0.002)'를 실행하면, 다음과 같은 $\lambda = 0.002$에 대한 능형회귀추정식을 출력한다.

$$\begin{aligned} y_{0.002} =& 14.3548x_1^s + 25.3624x_2^s + 24.8958x_3^s - 26.7622x_4^s \\ & - 28.7174x_5^s + 4.3757x_6^s + \hat{\epsilon}_{0.002} \end{aligned} \tag{4}$$

MATLAB명령문 'bLambda = ridge(y,D,lambdaa)'를 실행하면, 각 $\lambda \left(\in \left\{0, 10^{-5}, 2\times 10^{-5}, \cdots, 5\times 10^{-5}\right\}\right)$에 대한 능형회귀추정값을 출력한다. 또한, λ에 대한 능형회귀추정값을 그린 능형트레이스(ridge trace)가 그려진 그림 6.6.1을 출력한다. 이 그림에서도 알 수 있듯이, 능형모수가 $\lambda = 0$인 경우 각 능형회귀추정값이 각 최소제곱추정값과 같음을 확인할 수 있다. 또한, 능형모수 λ가 증가하면 능형회귀추정값의 절대값이 축소함을(shrink) 알 수 있다. 능형모수 λ가 증가해도 능형회귀추정값의 변화가 미미하면, 그 λ값을 최적값으로 선택하기도 한다. 이러한 선택법에 대해서는 논란이 많다. ■

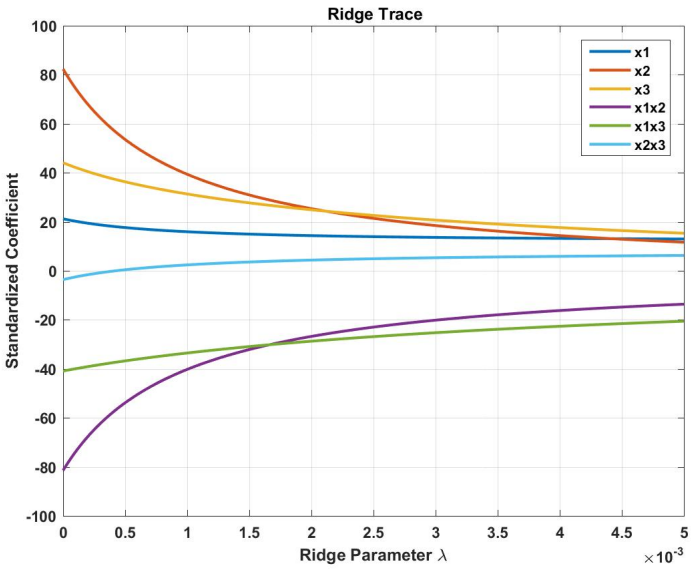


그림 6.6.1. 능형트레이스

능형회귀모형에 관한 자세한 내용은 **SAS4TSA3b**[3]의 제8.3절을 참조하라.

6.6.3 교차타당성

능형회귀모형에서 능형모수(ridge parameter) λ를 선택하는 것은 중요한 일이다. 능형모수를 선택하는 것은 모형선택(model selection)의 특수한 경우이다. 모형선택의 기준은 다양하지만, 최근에는 교차타당성기준(cross-validation criterion)이 널리 사용된다.

만약 어떤 데이터세트를 바탕으로 선택한 모형이 좋은 것이라면, 이 데이터세트에 새로운 관찰점들이 추가될 때 선택된 모형은 이 새로운 관찰점들도 잘 나타내야 할 것이다. 머신러닝의 용어를 빌어 설명하면, 통계적 모형을 선택하는데 사용되는 관찰점들의 집합을 트레이닝세트(training set)라 부르고, 이렇게 선택된 모형이 얼마나 좋은가 검정하는데 사용하는 관찰점들의 집합을 테스트세트(test set)라 부른다. 이상적으로 말하면, 관찰점들을 두 집합들로 나누어 하나는 트레이닝세트로 다른 하나는 테스트세트로 사용해야 한다.

가장 일반적인 교차타당성법은 다음과 같다.

(1단계) 트레이닝집합 T를 서로 배타적이며(separate) 같은 크기(equal size)를 갖는 K개 집합들로 나눈다. 이 집합들을 $T_1, T_2, \cdots, T_K$로 표기하자.

(2단계) 각 $k(=1,2,\cdots,K)$에 대해서 제k번째 집합(fold) T_k를 제외한 나머지 집합들을 트레이닝집합으로 해서 모형을 추정한다. 이 추정된 모형을 $\hat{f}_{-k}^{(\lambda)}$로 표기하자. 추정된 모형 $\hat{f}_{-k}^{(\lambda)}$를 사용해서 집합 T_k에 속한 관찰점의 적합값을 구한다. 다음과 같이 제k번째 집합(fold) T_k의 교차타당성오차(cross-validation error)를 계산한다.

$$(\text{CV Error})_k^{(\lambda)} = \frac{1}{|T_k|} \sum_{(\boldsymbol{x},y)\in T_k} \left[y - \hat{f}_{-k}^{(\lambda)}(\boldsymbol{x}) \right]^2$$

(3단계) 다음과 같이 총교차타당성오차(total cross-validation error)를 계산한다.

$$(\text{CV Error})^{(\lambda)} = \frac{1}{K} \sum_{1}^{K} (\text{CV Error})_k^{(\lambda)}$$

(4단계) 총교차타당성오차 $(\text{CV Error})^{(\lambda)}$를 최소화하는 λ를 선택하고, 이를 λ^*로 표기하자.

(5단계) 트레이닝집합 T에 속하는 각 $(\boldsymbol{x}, y)$에 대해서 예측값 $\hat{f}_{-k}^{(\lambda^*)}(\boldsymbol{x})$를 계산한다.

만약 관찰점들 개수가 충분히 크지 않으면, 제2단계에서 모형을 추정할 때 제k번째 집합(fold) T_k를 제외하지 않기도 한다.

트레이닝집합이 $K=1$ 인 경우, leave-one-out교차타당성법을 적용한다. 총교차타당성오차는 다음과 같다.

$$\mathrm{CV}(1)=\frac{1}{n}\sum_{j=1}^{n}\left[y-\hat{f}_{-j}(\boldsymbol{x}_j)\right]^2 \tag{6.6.48}$$

스무더행렬(smoother matrix)이 $S=[s_{ij}]$ 라고 하자. 최소제곱추정법과 같은 많은 선형적합추정법에서는 다음 식이 성립한다.

$$\mathrm{CV}(1)=\frac{1}{n}\sum_{j=1}^{n}\left[\frac{y_j-\hat{f}\left(\boldsymbol{x}_j\right)}{1-s_{jj}}\right]^2 \tag{6.6.49}$$

다음 근사식이 성립한다.

$$\mathrm{CV}(1)=\frac{1}{n}\sum_{j=1}^{n}\left[\frac{y_j-\hat{f}\left(\boldsymbol{x}_j\right)}{1-\frac{1}{n}\mathrm{Tr}(S)}\right]^2 \tag{6.6.50}$$

여기서 $\mathrm{Tr}(S)$ 가 효율적 자유도임을 상기하라. 식 (6.6.50) 의 우변을 일반화교차타당성(generalized cross-validation) 또는 GCV오차라 부른다.

6.6.4 선형회귀모형과 LASSO

식 (6.6.1) 의 선형회귀모형을 다시 생각해보자. 만약 $\det(X^tX)$ 가 0에 가까우면, 각 회귀계수의 최소제곱추정량의 분산이 아주 커져서 이 추정량을 신뢰하기 어렵다. 이러한 경우 다음과 같은 최적화문제를 살펴보자.

$$\underset{\boldsymbol{\beta}\in\mathbb{R}^p}{\text{Minimize}}\sum_{j=1}^{n}\left[y_j-\boldsymbol{x}_j^t\boldsymbol{\beta}\right]^2 \quad \text{subject to} \quad \sum_{i=1}^{p}|\beta_i|\le r. \tag{6.6.51}$$

식 (6.6.13) 에서는 제약조건이 β_j 의 2차식이었으나, 식 (6.6.51) 에서는 제약조건이 $|\beta_j|$ 의 1차식이다. 일반적으로 다음 식들이 성립한다고 가정한다.

$$E(y_j)=0,\quad (j=1,2,\cdots,n) \tag{6.6.52}$$

$$\frac{1}{n}\sum_{j=1}^{n}x_{i,j}=0,\quad \frac{1}{n-1}\sum_{j=1}^{n}\left[x_{i,j}-\frac{1}{n}\sum_{k=1}^{n}x_{i,k}\right]^2=1,\quad (i=1,2,\cdots,p) \tag{6.6.53}$$

식 (6.6.51)의 최적화문제를 풀기 위해서 다음과 같은 Lagrange함수를 정의하자.

$$PRSS_1(\boldsymbol{\beta}) \doteq \sum_{j=1}^{n} \left[y_j - \boldsymbol{x}_j^t \boldsymbol{\beta}\right]^2 + \lambda \sum_{i=1}^{p} |\beta_i|. \tag{6.6.54}$$

이 $PRSS_1(\boldsymbol{\beta})$의 아래첨자 1은 l_1-공간에서 노름을 의미한다. 함수 $y = |x|$는 점 $x = 0$에서 미분이 불가능하므로, 최적화문제 (6.6.51)을 해석적으로 풀 수는 없다. 그러나, $PRSS_1(\boldsymbol{\beta})$는 $\boldsymbol{\beta}$의 볼록함수이므로, 최적화문제 (6.6.51)은 일의적인 해를 갖는다. 이 최적화문제 (6.6.51)을 푸는 것을 LASSO(Least Absolute Shrinkage and Selection Operator)라 부르고, 이 최적화문제의 해 $\hat{\beta}_\lambda^L$를 LASSO계수라 부른다.

능형모수와 마찬가지로 LASSO모수 λ는 수축모수(shrinkage parameter)이다. 다음 식들이 성립한다.

$$\lim_{\lambda \downarrow 0} = p, \quad \lim_{\lambda \uparrow \infty} = 0 \tag{6.6.55}$$

즉, λ가 0으로 수렴하면, LASSO추정량은 최소제곱추정량으로 수렴한다. 또한, λ가 ∞로 발산하면, LASSO모형은 상수모형으로 수렴한다. 따라서, λ가 커지면, 많은 LASSO계수들이 정확히 0이 되고 또한 정규화(regularization) 정도가 증가한다.

예제 6.6.2 능형회귀추정모형보다 LASSO추정모형에서 더 많은 회귀계수들이 0이 된다. 즉 후자는 설긴 해(sparse solutions)를 제공한다. 이를 살펴보기 위해서 다음 MATLAB 프로그램 RidgeLassoGraph.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: RidgeLassoGraph.m
%  Graph for Ridge and LASSO
% Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all
% Feasible set
x1 = linspace(-1,1,1001);
Triang = 1- abs(x1);
Circl = sqrt(1-x1.^2);
% Intersection
a = 2;   % slope
xInt = 1/(a-1);
yInt = a/(a-1);
% Object function
tt = (0:1:360)'*(2*pi/360);
r(1) = sqrt(2/(a-1)^2)
r(2) = sqrt( (1+a^2)*(1/(a-1)-1/sqrt(a^2+1))^2 )
for kdum= 1:2
    yf1(:,kdum) = r(kdum)*cos(tt) + xInt;
    yf2(:,kdum) = r(kdum)*sin(tt) + yInt;
```

```
end

% Plotting
set(gca,'fontsize',11,'fontweigh','bold')
hold on
plot(x1,Triang,'b-',x1,Circl,'r-','LineWidth',2)
plot(x1,-Triang,'b-',x1,-Circl,'r-','LineWidth',2)
legend('LASSO','Ridge','location','NE')
plot(yf1(:,1),yf2(:,1),'b:',yf1(:,2),yf2(:,2),'r:','LineWidth',2)
plot([0 0], [ -5 5 ],'k','LineWidth',1.2);
plot([ -5 5 ], [0 0 ],'k','LineWidth',1.2);
plot(0,1,'kO',1/sqrt(a^2+1),a/sqrt(a^2+1),'k*')
axis equal
axis([-0.1 1.6 -0.1 1.6 ])
text(0,1.05,'\bf L')
text(0.45, 0.93,'\bf R')
xlabel('\bf \beta_{1}'), ylabel('\bf \beta_{2}')
hold off
saveas(gcf,'RidgeLassoGraph','jpg')
% End of program
% ------------------------------------------------------------------------
```

LASSO의 제약조건 R^{LASSO}과 능형회귀모형의 제약조건 R^{Ridge}이 각각 다음과 같다고 하자.

$$R^{LASSO} \doteq \{(\beta_1,\, \beta_2)\ ;\ |\beta_1| + |\beta_2| \le 1\} \tag{1}$$

$$R^{Ridge} \doteq \left\{(\beta_1,\, \beta_2)\ ;\ \beta_1^2 + \beta_2^2 \le 1\right\} \tag{2}$$

또한, 목적함수가 다음과 같다고 하자.

$$f\,(\beta_1,\, \beta_2) = \left[\beta_1 - \frac{1}{a-1}\right]^2 + \left[\beta_2 - \frac{a}{a-1}\right]^2 \tag{3}$$

여기서 a는 개구간 $(1, \infty)$에 속하는 주어진 상수이다. 이 MATLAB프로그램을 실행하면, 그림 6.6.2를 출력한다. 이 그림에서 알 수 있듯이, 이 목적함수는 능형회귀모형의 제약조건 R^{Ridge}와 점 $R\left(\frac{1}{\sqrt{1+a^2}}, \frac{a}{\sqrt{1+a^2}}\right)$에서 만나고, LASSO의 제약조건 R^{LASSO}와 점 $L(0,1)$에서 만난다. 즉, 능형회귀모형의 회귀계수들보다 LASSO모형의 회귀계수들 중에 0인 것이 더 많다. ■

앞에서도 언급했듯이 LASSO모형의 최적해를 해석적으로 구할 수 없으므로, 컴퓨터를 사용해서 구현할 수 밖에 없다. 대표적인 LASSO알고리즘으로는 LARS(least angle regression), Forward stagewise 등이 있다. 본저자는 이 문제를 푸는 핵심 알고리즘은 Karmarkar알고리즘이 아닐까 생각한다. 이 알고리즘에 대해서는 Karmarkar [32]와 Strang [58]을 참조하라.

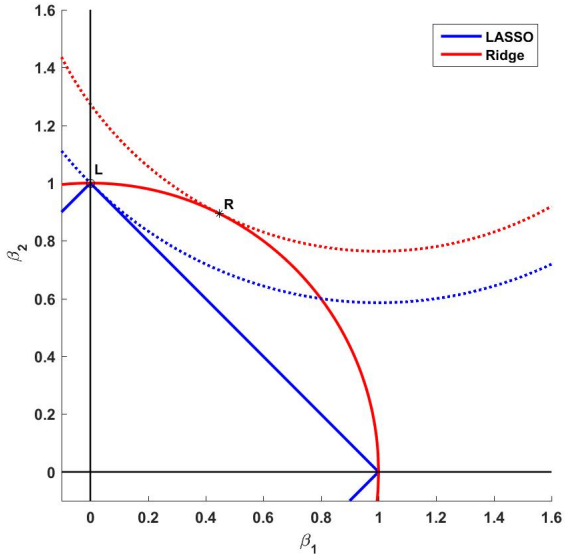


그림 6.6.2. 능형회귀모형과 LASSO

이 소절에서는 머신러닝에서 자주 사용되는 Python패키지들 statmodels와 Scikit-Learn을 사용해서 능형회귀모형, LASSO회귀모형, 그리고 Elastic Net회귀모형을 추정하기로 하자.

여기서 Elastic Net회귀모형은 능형회귀모형과 LASSO회귀모형을 일반화한 것으로서, 능형회귀모형 (6.6.16)이나 LASSO회귀모형 (6.6.52) 대신에 다음 최적화문제의 해로 정의되는 것이다.

$$PRSS_{EN}(\boldsymbol{\beta}) \doteq \sum_{j=1}^{n} \left[y_j - \boldsymbol{x}_j^t \boldsymbol{\beta}\right]^2 + \lambda_1 \sum_{i=1}^{p} |\beta_i| + \lambda_2 \sum_{i=1}^{p} \beta_i^2. \tag{6.6.56}$$

여기서 $\lambda_1 = 0$이면, 능형회귀모형이다. 또한, $\lambda_2 = 0$이면, LASSO회귀모형이다.

예제 6.6.3 Python패키지 statsmodels를 사용해서 선형회귀모형, 능형회귀모형, LASSO회귀모형, 그리고 Elastic Net회귀모형을 추정하기 위해서, 다음 Python프로그램 RegularizedRegression201.Py을 실행해 보자.

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import pandas as pd

# Plotting
def plot_MYstatsmodels(result):
    plt.scatter(X,y)
```

```
    xInt = np.linspace(0,1,1024)
    dfxInt0 = pd.DataFrame(xInt, columns=["x"])
    dfxInt =  sm.add_constant(dfxInt0)
    plt.plot(xInt, result.predict(dfxInt),'r',lw=2)

# Make dataset
np.random.seed(1)
Nsamples = 32
X = np.sort(np.random.rand(Nsamples))
y = 5*np.cos(np.pi*X)*np.exp(-X) + np.random.randn(Nsamples)

dfX0 = pd.DataFrame(X, columns=["x"])
dfX = sm.add_constant(dfX0)
dfy = pd.DataFrame(y, columns=["y"])
df = pd.concat([dfX, dfy], axis=1)

# Least Squares Method
model = sm.OLS.from_formula("y ~ x + I(x**2) + I(x**3) + I(x**4) + I(x**5)",
                            data=df)
resultOLS = model.fit()
print(resultOLS.params)
print(resultOLS.tvalues)
fig = plt.figure()
plt.subplot(2,2,1)
plt.title('Least Squares Regression')
plot_MYstatsmodels(resultOLS)

# Ridge Regression
resultRidge = model.fit_regularized(alpha=0.02, L1_wt=0)
print(resultRidge.params)
print(resultRidge.tvalues)
plt.subplot(2,2,2)
plt.title('Ridge Regression')
plot_MYstatsmodels(resultRidge)

# LASSO Regression
resultLASSO = model.fit_regularized(alpha=0.02, L1_wt=1)
print(resultLASSO.params)
print(resultLASSO.tvalues)
plt.subplot(2,2,3)
plt.title('LASSO Regression')
plot_MYstatsmodels(resultLASSO)

# Elastic Net Regression
resultEN = model.fit_regularized(alpha=0.02, L1_wt=0.56789)
print(resultEN.params)
print(resultEN.tvalues)
plt.subplot(2,2,4)
plt.title('Elastic Net Regression')
plot_MYstatsmodels(resultEN)

plt.show()
fig.savefig('RegularizedRegression201Py.png')

# End of Program
```

최소제곱추정법에 의한 추정모형은 다음과 같다.

$$y = 4.968348 + 6.858236x - 138.832301x^2 + 375.574653x^3 - 413.745775x^4 + 164.847685x^5 + \hat{\epsilon}_{OLS} \quad (1)$$

능형회귀추정법에 의한 추정모형은 다음과 같다.

$$y = 3.783023 - 4.606171x - 2.223633x^2 - 0.559235x^3 + 0.349287x^4 + 0.839145x^5 + \hat{\epsilon}_{Ridge} \quad (2)$$

LASSO추정법에 의한 추정모형은 다음과 같다.

$$y = 4.888049 - 9.207765x + 0.000000x^2 + 0.000000x^3 + 0.000000x^4 + 2.786402x^5 + \hat{\epsilon}_{LASSO} \quad (3)$$

Python패키지 statsmodels에서 Elastic Net모형은 다음 Lagrange함수를 최소화한다.

$$PRSS_{EN}(\alpha, L_l) = \frac{1}{2n} \| \boldsymbol{y} - X\boldsymbol{\beta} \|^2 + \alpha \left\{ \frac{1}{2}[1 - L_1] \sum_i \beta_i^2 + L_1 \sum_i |\beta_i| \right\} \quad (4)$$

모수들이 $\alpha = 0.02$과 $L_1 = 0.56789$인 Elastic Net추정법에 의한 추정모형은 다음과 같다.

$$y = 4.199538 - 6.254754x - 1.605844x^2 + 0.000000x^3 + 0.340139x^4 + 1.121965x^5 + \hat{\epsilon}_{EN} \quad (5)$$

이 MATLAB프로그램을 실행하면, 그림 6.6.3를 출력한다. 그림 6.6.3의 각 그래프에는 청색 점으로 관찰점들이 표시되어 있으며 적색 실선으로 회귀식이 그려져 있다. ■

예제 6.6.4 Python패키지 Scikit-Learn를 사용해서 선형회귀모형, 능형회귀모형, LASSO 회귀모형, 그리고 Elastic Net회귀모형을 추정하기 위해서, 다음 Python프로그램 RegularizedRegression202.Py을 실행해 보자.

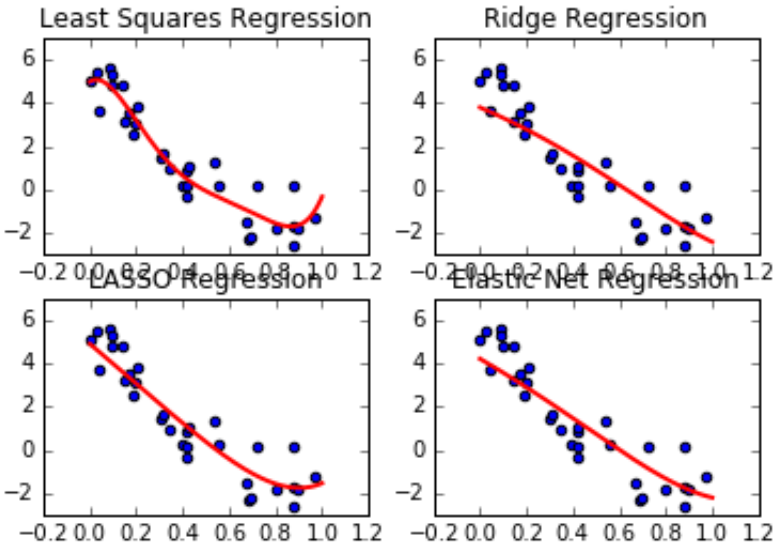


그림 6.6.3. statsmodels 회귀분석

```
"""
Created on Fri Jan 13 23:10:59 2017
@author: CBS
"""
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.pipeline import make_pipeline

# Plotting
def plot_MYsklearn(model):
    plt.scatter(X,y)
    xInt = np.linspace(0,1,1024)
    plt.plot(xInt, model.predict(xInt[:, np.newaxis]),'r',lw=2)

# Make dataset
np.random.seed(1)
Nsamples = 32
X = np.sort(np.random.rand(Nsamples))
y = 5*np.cos(np.pi*X)*np.exp(-X) + np.random.randn(Nsamples)
poly = PolynomialFeatures(5)

# Least Squares Method
# modelOLS = make_pipeline(poly, LinearRegression()).fit(X[:, np.newaxis],y)
modelOLS = make_pipeline(poly, LinearRegression())
modelOLS.fit(X[:, np.newaxis],y)
print(modelOLS.steps[1][1].coef_)
fig = plt.figure()
plt.subplot(2,2,1)
plt.title('Least Squares Regression')
plot_MYsklearn(modelOLS)

# Ridge Regression
modelRidge = make_pipeline(poly, Ridge(alpha=0.02))
modelRidge.fit(X[:, np.newaxis],y)
```

```
print(modelRidge.steps[1][1].coef_)
plt.subplot(2,2,2)
plt.title('Ridge Regression')
plot_MYsklearn(modelRidge)

# Ridge Regression
modelLASSO = make_pipeline(poly, Lasso(alpha=0.02))
modelLASSO.fit(X[:, np.newaxis],y)
print(modelLASSO.steps[1][1].coef_)
plt.subplot(2,2,3)
plt.title('LASSO Regression')
plot_MYsklearn(modelLASSO)

# Elastic Net Regression
modelEN = make_pipeline(poly,ElasticNet(alpha=0.02, l1_ratio=0.56789))
modelEN.fit(X[:, np.newaxis],y)
print(modelLASSO.steps[1][1].coef_)
plt.subplot(2,2,4)
plt.title('Elastic Net Regression')
plot_MYsklearn(modelEN)

plt.show()
fig.savefig('RegularizedRegression202Py.png')

# End of Program
```

최소제곱추정법에 의한 추정모형은 다음과 같다.

$$y = 0.000000 + 6.85823603x - 138.83230057x^2 + 375.57465303x^3 - 413.7457746x^4 + 164.84768493x^5 + \hat{\epsilon}_{OLS} \tag{1}$$

능형회귀추정법에 의한 추정모형은 다음과 같다.

$$y = 0.000000 - 10.89991402x - 1.29557393x^2 + 2.30615163x^3 + 2.32290884x^4 + 1.37867435x^5 + \hat{\epsilon}_{Ridge} \tag{2}$$

LASSO추정법에 의한 추정모형은 다음과 같다.

$$y = 0.000000 - 9.41782726x + 0.000000x^2 + 0.000000x^3 + 0.000000x^4 + 2.96973354x^5 + \hat{\epsilon}_{LASSO} \tag{3}$$

Python패키지 Scikit-Learn에서 Elastic Net모형은 다음 Lagrange함수를 최소화한다.

$$PRSS_{EN}(\alpha, L_l) = \frac{1}{2n} \| \boldsymbol{y} - X\boldsymbol{\beta} \|^2 + \alpha \left\{ \frac{1}{2}[1 - L_1] \sum_i \beta_i^2 + L_1 \sum_i |\beta_i| \right\} \tag{4}$$

모수들이 $\alpha = 0.02$과 $L1_{wt} = 0.56789$인 Elastic Net추정법에 의한 추정모형은 다음과 같다.

$$\begin{aligned} y = {} & 0.000000 - 9.41782726x + 0.000000x^2 \\ & + 0.000000x^3 + 0.000000x^4 + 2.96973354x^5 + \hat{\epsilon}_{EN} \end{aligned} \tag{5}$$

이 MATLAB프로그램을 실행하면, 그림 6.6.4를 출력한다. 그림 6.6.4의 각 그래프에는 청색 점으로 관찰점들이 표시되어 있으며 적색 실선으로 회귀식이 그려져 있다. ■

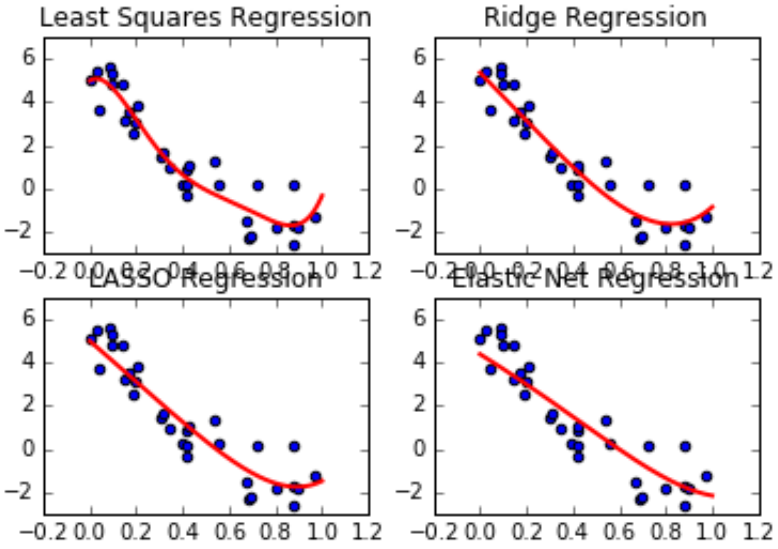


그림 6.6.4. Scikit-Learn 회귀분석

6.6.5 정규화된 회귀분석

함수 $f_i : \mathbb{R} \mapsto \mathbb{R}$의 2차 도함수가 연속이라고 하고, 다음과 같이 함수 $F : \mathbb{R}^m \mapsto \mathbb{R}$를 정의하자.

$$F(\boldsymbol{x}) = \sum_{i=1}^{m} f_i(x_i) \tag{6.6.57}$$

다음 식들이 성립한다.

$$\nabla F(\boldsymbol{x}) = \begin{bmatrix} f_1'(x_1) \\ f_2'(x_2) \\ \vdots \\ f_m'(x_m) \end{bmatrix}, \quad \nabla^2 F(\boldsymbol{x}) = \begin{bmatrix} f_1''(x_1) & 0 & \cdots & 0 \\ 0 & f_2''(x_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & f_m''(x_m) \end{bmatrix} \tag{6.6.58}$$

여기서 각 i에 대해서 식 $f_i''(x) > 0$이 성립한다고 가정하자. 즉, $f_i(x)$는 아래로 볼록인 함수이다. 다음과 같이 함수 $F : \mathbb{R}^n \mapsto \mathbb{R}$를 정의하자.

$$L(\boldsymbol{x}) = \sum_{j=1}^{m} f_j\left(\boldsymbol{a}_j^t \boldsymbol{x} + b_j\right) \tag{6.6.59}$$

여기서 식 $m \leq n$이 성립하고, $\boldsymbol{x} = [x_1, x_2, \cdots, x_n]^t$이고, 각 j에 대해서 $\boldsymbol{a}_j = [a_{1j}, a_{2j}, \cdots, a_{nj}]^t$는 알려진 벡터이고 b_j는 알려진 스칼라이다. 우리의 목적은 다음과 같은 최적화문제를 푸는 것이다.

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{Minimize}}\, L(\boldsymbol{x}). \tag{6.6.60}$$

다음 식들이 성립한다.

$$\frac{\partial \boldsymbol{a}_j^t \boldsymbol{x}}{\partial x_i} = \frac{\partial\left(\sum_{k=1}^{n}\left[a_{kj}x_k + b_k\right]\right)}{\partial x_i} = a_{ij} \tag{6.6.61}$$

즉, 각 j에 대해서 다음 식이 성립한다.

$$\nabla \boldsymbol{a}_j^t \boldsymbol{x} = \boldsymbol{a}_j \tag{6.6.62}$$

따라서, 각 i에 대해서 다음 식이 성립한다.

$$\frac{\partial L(\boldsymbol{x})}{\partial x_i} = \sum_{k=1}^{m} a_{i,k} f_k'\left(\boldsymbol{a}_k^t \boldsymbol{x} + b_k\right) \tag{6.6.63}$$

식 (6.6.63)에서 알 수 있듯이, 다음 식이 성립한다.

$$\nabla L(\boldsymbol{x}) = A^t \nabla F\left(A\boldsymbol{x} + \boldsymbol{b}\right) \tag{6.6.64}$$

식 (6.6.63)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\frac{\partial^2 L(\boldsymbol{x})}{\partial x_i\, \partial x_j} = \sum_{k=1}^{m} a_{i,k} \frac{\partial}{\partial x_j} f_k'\left(\boldsymbol{a}_k^t \boldsymbol{x} + b_k\right) = \sum_{k=1}^{m} a_{i,k} f_k''\left(\boldsymbol{a}_k^t \boldsymbol{x} + b_k\right) a_{j,k} \tag{6.6.65}$$

식 (6.6.65)에서 알 수 있듯이, 다음 식이 성립한다.

$$\nabla^2 L(\boldsymbol{x}) = A^t \nabla^2 F\left(A\boldsymbol{x} + \boldsymbol{b}\right) A \tag{6.6.66}$$

여기서 $A = [a_{ij}]$는 $n \times m$ 행렬이다.

이 최적화문제를 정규화된 회귀분석문제(regularized regression problem)로 접근하기로 하자. 함수 $\phi_i : \mathbb{R} \mapsto \mathbb{R}$의 2차 도함수가 연속이라고 하고, 다음과 같이 함수 $J : \mathbb{R}^n \mapsto \mathbb{R}$를 정의하자.

$$J(\boldsymbol{x}) = L(\boldsymbol{x}) + \lambda \Phi(\boldsymbol{x}) \tag{6.6.67}$$

여기서 정규화모수 $\lambda(>0)$는 상수이고, 함수 $\Phi(\boldsymbol{x})$는 다음과 같다.

$$\Phi(\boldsymbol{x}) \doteq \sum_{i=1}^{n} \phi_i(x_i) \tag{6.6.68}$$

다음 식들이 성립함을 쉽게 알 수 있다.

$$\nabla \Phi(\boldsymbol{x}) = \begin{bmatrix} \phi_1'(x_1) \\ \phi_2'(x_2) \\ \vdots \\ \phi_n'(x_n) \end{bmatrix}, \quad \nabla^2 \Phi(\boldsymbol{x}) = \begin{bmatrix} \phi_1''(x_1) & 0 & \cdots & 0 \\ 0 & \phi_2''(x_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \phi_n''(x_n) \end{bmatrix} \tag{6.6.69}$$

여기서 각 i에 대해서 식 $\phi_i''(x) > 0$이 성립한다고 가정하자. 즉, $\phi_i(x)$는 아래로 볼록인 함수이다. 최적화의 1차조건인 방정식 $\nabla J(\boldsymbol{x}) = 0$의 근을 구하기 위한 Newton-Raphson 알고리즘은 다음과 같다.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left[\nabla^2 J(\boldsymbol{x}_k)\right]^{-1} \nabla J(\boldsymbol{x}_k) \tag{6.6.70}$$

식 (6.6.64)와 식 (6.6.67)에서 알 수 있듯이, 다음 식이 성립한다.

$$\nabla J(\boldsymbol{x}) = A^t \nabla F(A\boldsymbol{x} + \boldsymbol{b}) + \lambda \nabla \Phi(\boldsymbol{x}) \tag{6.6.71}$$

식 (6.6.66), 식 (6.6.67), 그리고 식 (6.6.69)에서 알 수 있듯이, 다음 식이 성립한다.

$$\nabla^2 J(\boldsymbol{x}) = A^t \nabla^2 F(A\boldsymbol{x} + \boldsymbol{b}) A + \lambda \nabla^2 \Phi(\boldsymbol{x}) \tag{6.6.72}$$

식 $f_i''(x) > 0$과 식 $\phi_i''(x) > 0$이 성립한다고 가정했으므로, Hessian행렬 $\nabla^2 J(\boldsymbol{x})$은 양정치 행렬이다. 식 (6.6.71)과 식 (6.6.72)을 식 (6.6.70)에 대입하면, Newton-Raphson알고리즘이

다음과 같음을 알 수 있다.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left[A^t \nabla^2 F\left(A\boldsymbol{x}_k + \boldsymbol{b}\right) A + \lambda \nabla^2 \Phi\left(\boldsymbol{x}_k\right)\right]^{-1} \left[A^t \nabla F\left(A\boldsymbol{x}_k + \boldsymbol{b}\right) + \lambda \nabla \Phi\left(\boldsymbol{x}_k\right)\right] \tag{6.6.73}$$

예제 6.6.5 다음 최적화문제를 살펴보자.

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{Minimize}} \sum_{i=1}^{n} \left[\boldsymbol{a}_i^t \boldsymbol{x} + b_i - y_i \ln\left(\boldsymbol{a}_i^t \boldsymbol{x} + b_i\right) + \lambda x_i^2\right]. \tag{1}$$

여기서 행렬 $A = [a_{ij}]$는 주어진 것이고, $y_1(>0)$, $y_2(>0)$, $\cdots$, $y_n(>0)$, $b_1(>0)$, $b_2(>0)$, $\cdots$, $b_n(>0)$ 그리고 $\lambda(>0)$는 주어진 상수들이다. 또한, x_1, x_2, $\cdots$, x_n은 양(positive)인 변수들이다.

다음 함수들을 식 (6.6.67)에 대입하면, 식 (1)의 목적함수를 얻는다.

$$f_i(x) = x - y_i \ln x, \quad \phi(x) = x^2 \tag{2}$$

즉, 다음 목적함수를 얻는다.

$$J(\boldsymbol{x}) = \sum_{j=1}^{m} f_j\left(\boldsymbol{a}_j^t \boldsymbol{x} + b_j\right) + \lambda \sum_{i=1}^{n} \phi\left(x_i\right) \tag{3}$$

또한, 다음 식들이 성립한다.

$$F(\boldsymbol{x}) = \sum_{i=1}^{n} \left[x_i - y_i \ln x_i\right] \tag{4}$$

$$\nabla F(\boldsymbol{x}) = \begin{bmatrix} 1 - \frac{y_1}{x_1} \\ 1 - \frac{y_2}{x_2} \\ \vdots \\ 1 - \frac{y_n}{x_n} \end{bmatrix}, \quad \nabla^2 F(\boldsymbol{x}) = \begin{bmatrix} \frac{y_1}{x_1^2} & 0 & \cdots & 0 \\ 0 & \frac{y_2}{x_2^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{y_n}{x_n^2} \end{bmatrix} \tag{5}$$

가정 $y_i > 0$으로부터 다음 식이 성립함을 알 수 있다.

$$\frac{y_i}{x_i^2} > 0 \tag{6}$$

즉, $\nabla^2 F(\boldsymbol{x})$가 양정치행렬임을 알 수 있다.

$$\Phi(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2 \tag{7}$$

$$\nabla\Phi(\boldsymbol{x}) = \begin{bmatrix} 2x_1 \\ 2x_2 \\ \vdots \\ 2x_n \end{bmatrix}, \quad \nabla^2\Phi(\boldsymbol{x}) = \begin{bmatrix} 2 & 0 & \cdots & 0 \\ 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 2 \end{bmatrix} \tag{8}$$

여기서 $\nabla^2\Phi(\boldsymbol{x})$가 양정치행렬임을 알 수 있다. 식 (4)와 식 (5) 그리고 식 (7)과 식 (8)에서 알 수 있듯이, 다음 식들이 성립한다.

$$A^t \nabla^2 F\left(A\boldsymbol{x} + \boldsymbol{b}\right) A = \sum_{i=1}^{n} \frac{y_i}{\left[\boldsymbol{a}_i^t \boldsymbol{x} + b_i\right]^2} \boldsymbol{a}_i \boldsymbol{a}_i^t \tag{9}$$

$$A^t \nabla F\left(A\boldsymbol{x} + \boldsymbol{b}\right) = \sum_{i=1}^{n} \left[\mathbf{1} - \frac{y_i}{\boldsymbol{a}_i^t \boldsymbol{x} + b_i} \boldsymbol{a}_i\right] \tag{10}$$

따라서, 최적화의 1차조건인 방정식 $\nabla J(\boldsymbol{x}) = \mathbf{0}$를 풀기위한 Newton-Raphson알고리즘은 다음과 같다.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left[\sum_{i=1}^{n} \frac{y_i}{\left[\boldsymbol{a}_i^t \boldsymbol{x}_k + b_i\right]^2} \boldsymbol{a}_i \boldsymbol{a}_i^t + 2\lambda I_n\right]^{-1} \left\{\sum_{i=1}^{n} \left[\mathbf{1} - \frac{y_i}{\boldsymbol{a}_i^t \boldsymbol{x} + b_i}\right] \boldsymbol{a}_i + 2\lambda \boldsymbol{x}_k\right\} \tag{11}$$

식 (5), 식 (6) 그리고 식 (8)에서 알 수 있듯이, $\nabla^2 J(\boldsymbol{x})$은 양정치행렬이다. 즉, 최적화의 2차조건이 만족된다.

식 (11)을 사용해서 이 정규화된 회귀분석문제를 풀기위해서, 다음 MATLAB프로그램 RegularizedRegression101DY.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: RegularizedRegression101DY.m
% Regularized Regression Problem w/ Newton-Raphson Method
% Programmed by CDY
% ------------------------------------------------------------------------
clear all, close all
A = xlsread('RRdata_A.xlsx');
Others = xlsread('RRdata_Others.xlsx');
b = Others(:,1);
t = Others(:,2);
x0 = Others(:,3);
y = Others(:,4);
whos
n = 512;
```

```
lambda = 10^(-3);
xstar = ones(n,1);
Atrans = A';
MaxIter = 100;
for kk=1:MaxIter
    xold = xstar;
    c = A*xold+b;
    Hessian = zeros(n,n);
    Grad = zeros(n,1);
    for ii=1:n
        duma = Atrans(:,ii);
        cdum = c(ii);
        Hessian = Hessian + y(ii)/cdum^2*duma*duma';
        Grad = Grad + (1-y(ii)/cdum)*duma;
    end % end-for ii
    Hessian = Hessian + 2*lambda*eye(n);
    Grad = Grad + 2*lambda*xold;
    xstar = xold - Hessian\Grad;
    normGrad(kk) = norm(Grad);
    if normGrad(kk) < 10^-10
        break
    end % end-if
end % for_kk
format long
kk
normGrad
% Plot
plot(t,xstar,'k-',t,x0,'r-','LineWidth',2);
legend('\bf Estimated Signal','\bf True Signal')
set(gca,'fontsize',11,'fontweigh','bold')
% title('\bf Regression with Newton''s method')
xlabel('\bf time','fontsize',12)
ylabel('\bf Signal','fontsize',12)
saveas(gcf,'RegularizedRegression101aDY.jpg')
figure
plot(1:kk,normGrad(1:kk),'r-','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
title('\bf Norm of Gradient')
xlabel('\bf Iterate Number','fontsize',12)
ylabel('\bf Norm of Gradient','fontsize',12)
saveas(gcf,'RegularizedRegression101bDY.jpg')
save('RegularizedRegression101DY.txt','xstar','normGrad','-ascii')
% End of Program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 RegularizedRegression101DY.m을 실행하면, 그림 6.6.5가 출력된다. 그림 6.6.5에서 적색 실선은 진짜해를 나타내는 곡선이고 흑색 실선은 정규회귀추정법에 의해서 추정된 곡선이다. 정규회귀추정법에는 정규화 단계가 포함되어 있다. 따라서, 진짜 곡선에 비해 추정된 곡선이 더 평활해진다. 그림 6.6.6에는 그래디언트벡터(gradient)의 노름이 급격히 감소함을 알 수 있다. 제6번째 반복단계(iteration step)에서 그래디언트벡터의 노름이 10^{-10} 보다 작아진다. ■

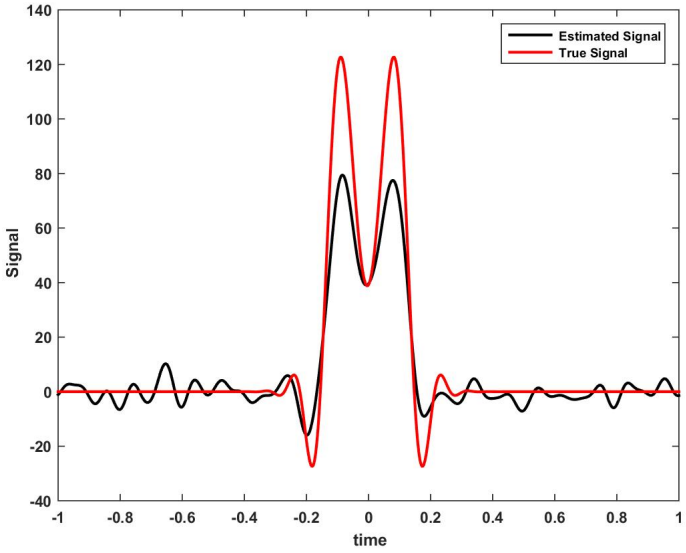


그림 6.6.5. 정규회귀분석

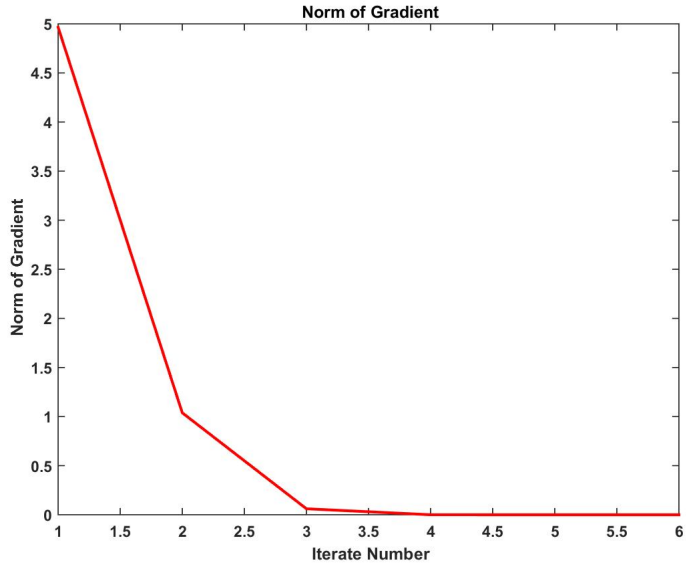


그림 6.6.6. 그래디언트벡터의 노름

6.6.6 능형회귀

다음과 같은 근사방정식을 살펴보자.

$$A\boldsymbol{x} = \boldsymbol{b} \tag{6.6.74}$$

여기서 A는 $m \times n$ 행렬이고 $\boldsymbol{b}$는 $m \times 1$ 벡터이다. 이 문제에서는 식 $m \leq n$이 성립한다고 가정하자. 만약 식 $\operatorname{rank}(A) < m$이 성립하면, 방정식 (6.6.74)에는 무수히 많은 해가 존재한다.

이 해들 대신에 다음 최적화문제의 해를 사용하기로 하자.

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{Minimize}} \left[\frac{1}{2}\|A\boldsymbol{x}-\boldsymbol{b}\|^2+\frac{\lambda}{2}\|\boldsymbol{x}\|^2\right]. \tag{6.6.75}$$

다음 함수를 정의하자.

$$J(\boldsymbol{x}) \doteq \frac{1}{2}\|A\boldsymbol{x}-\boldsymbol{b}\|^2+\frac{\lambda}{2}\|\boldsymbol{x}\|^2 \tag{6.6.76}$$

여기서 λ는 양수이다. 다음 식이 성립한다.

$$J(\boldsymbol{x}) = \frac{1}{2}\left[\boldsymbol{x}^tA^tA\boldsymbol{x}-2\boldsymbol{b}^tA\boldsymbol{x}+\boldsymbol{b}^t\boldsymbol{b}\right]+\frac{\lambda}{2}\boldsymbol{x}^t\boldsymbol{x} \tag{6.6.77}$$

따라서 다음 식들이 성립한다.

$$\nabla J(\boldsymbol{x}) = \frac{dJ(\boldsymbol{x})}{d\boldsymbol{x}} = \left[A^tA+\lambda I\right]\boldsymbol{x}-A^t\boldsymbol{b} \tag{6.6.78}$$

즉, 최적화문제 (6.6.75)의 해 $\boldsymbol{x}^*$는 다음 식을 만족한다.

$$\left[A^tA+\lambda I\right]\boldsymbol{x}^* = A^t\boldsymbol{b} \tag{6.6.79}$$

Newton-Raphson법을 사용해서 식 (6.6.79)을 푸는데 $2n^3/3$ flops가 필요하다. 그러나, 이는 각 λ에 대한 계산량이다.

여러 λ에 대해서 방정식 (6.6.79)을 풀기위해서 특이값분해(singular value decomposition: SVD)를 사용하기로 하자. 행렬 A의 특이값분해는 다음과 같다.

$$A = U\Gamma V^t \tag{6.6.80}$$

여기서 $U_{m\times m}$, $V_{n\times n}$, 그리고 $\Gamma_{m\times n}$은 다음 식들을 만족한다.

$$U^tU = I_m, \quad V^tV = I_n, \quad \Gamma = \begin{bmatrix} \gamma_1 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \gamma_2 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & \gamma_m & 0 & \cdots & 0 \end{bmatrix} \tag{6.6.81}$$

다음 식들이 성립한다.

$$A^t A + \lambda I_n = \left[U\Gamma\, V^t\right]^t \left[U\Gamma\, V^t\right] + \lambda I_n = V\Gamma^t\Gamma V^t + \lambda V I_n V^t = V D V^t \tag{6.6.82}$$

여기서 D는 다음과 같다.

$$D \doteq \Gamma^t\Gamma + \lambda I_n = \begin{bmatrix} \gamma_1^2+\lambda & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \gamma_2^2+\lambda & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & \gamma_m^2+\lambda & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & \lambda & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & \lambda \end{bmatrix} \tag{6.6.83}$$

식 (6.6.79)와 식 (6.6.82)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\boldsymbol{x}^* = \left[A^t A + \lambda I_n\right]^{-1} A^t \boldsymbol{b} = [V D V^t]^{-1}[U\Gamma V^t]\boldsymbol{b} = V D^{-1}\Gamma^t U^t \boldsymbol{b} \tag{6.6.84}$$

식 (6.6.83)에서 알 수 있듯이, 다음 식이 성립한다.

$$D^{-1}\Gamma^t = \begin{bmatrix} \frac{\gamma_1}{\gamma_1^2+\lambda} & 0 & 0 & \cdots & 0 \\ 0 & \frac{\gamma_2}{\gamma_2^2+\lambda} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & \frac{\gamma_m}{\gamma_m^2+\lambda} \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{6.6.85}$$

다음과 같이 벡터들 $\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_m$ 과 $\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n$ 을 정의하자.

$$U \doteq [\boldsymbol{u}_1, \boldsymbol{u}_2, \cdots, \boldsymbol{u}_m], \quad V \doteq [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n] \tag{6.6.86}$$

식 (6.6.84)∼식 (6.6.86)에서 알 수 있듯이, 다음 식이 성립한다.

$$\boldsymbol{x}^* = \left[\frac{\gamma_1}{\gamma_1^2+\lambda}\boldsymbol{v}_1,\ \frac{\gamma_2}{\gamma_2^2+\lambda}\boldsymbol{v}_2,\ \cdots,\ \frac{\gamma_m}{\gamma_m^2+\lambda}\boldsymbol{v}_m \right] \begin{bmatrix} \boldsymbol{u}_1^t\boldsymbol{b} \\ \boldsymbol{u}_2^t\boldsymbol{b} \\ \vdots \\ \boldsymbol{u}_m^t\boldsymbol{b} \end{bmatrix} \tag{6.6.87}$$

식 (6.6.87)를 다음과 같이 쓸 수 있다.

$$\boldsymbol{x}^* = \sum_{i=1}^{m} \frac{\gamma_i \boldsymbol{u}_i^t\boldsymbol{b}}{\gamma_i^2+\lambda}\boldsymbol{v}_i \tag{6.6.88}$$

이 $\boldsymbol{x}^*$를 방정식 (6.6.74)의 능형회귀추정해(ridge regression estimator solution)이라 부른다.

예제 6.6.6 방정식 $A\boldsymbol{x}=\boldsymbol{b}$의 능형회귀추정해를 구하기 위해서, 다음 MATLAB프로그램 RidgeRegression101DY.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: RidgeRegression101DY.m
%  Ridge Regression
%  Programmed by CDY
%------------------------------------------------------------------------
clear all, close all, clc, format long
% Making Data
seedd = (1001^2+1)/2;
rng(seedd)
Af = exp(randn(100,40));
xTrue = (log((11:50)')).^2;
bf = Af*xTrue;
A = Af(1:30,:);
b = bf(1:30);
% Ridge regression
[ m n ] = size(A)
[ U Sigma V ] = svd(A);
lambda = [ 0.01 5 10 ]
for kk = 1:3
    xdum2 = zeros(n,1);
   for ii=1:m
       xdum2 = xdum2 + Sigma(ii,ii)/(Sigma(ii,ii)^2+lambda(kk)) ...
               *(U(:,ii)'*b)*V(:,ii);
   end % end-for-ii
   xstar(:,kk)= xdum2;
   DoubleCheckk(kk) = norm(A*xstar(:,kk)-b);
end % end-for-kk
% Estimation
VarNum = (1:40)';
plot(VarNum,xTrue,'k.',VarNum,xstar(:,1),'r-.',VarNum,xstar(:,2),'g-', ...
     VarNum,xstar(:,3),'b:','LineWidth',1.5);
set(gca,'fontsize',11,'fontweigh','bold')
```

```
legend('True x','\lambda = 0.01','\lambda = 5','\lambda = 10', ...
       'location','SE')
saveas(gcf,'RidgeRegression101aDY.jpg')
% Prediction
figure
ObsNum = (1:100)';
bhat = Af*xstar;
hold on
plot(ObsNum,bhat(:,1),'g-',ObsNum,bhat(:,2),'r--', ...
     ObsNum,bhat(:,3),'b:','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
legend('\lambda = 0.01','\lambda = 5','\lambda = 10','location','NW')
hold off
saveas(gcf,'RidgeRegression101bDY.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB 프로그램 RidgeRegression101DY.m을 실행하면, 최적화문제 (6.6.75)를 풀어서 능형회귀추정해를 구한다. 우선 정규난수들을 원소로 하는 100×40 행렬 A_f를 생성하고, 40×1 벡터 $\boldsymbol{x}_{True} = \left[\ln^2(1+10), \ln^2(2+10), \cdots, \ln^2(40+10)\right]$를 생성하고, 이들을 이용해서 100×1 벡터 $\boldsymbol{b}_f = A_f \boldsymbol{x}_{True}$을 계산한다. 행렬 A_f의 첫 30행들로 이루어진 행렬이 A 그리고 벡터 $\boldsymbol{b}_f$의 첫 30행들로 이루어진 벡터가 $\boldsymbol{b}$이다.

다음 단계로 능형모수 λ가 0.01, 5 그리고 10 인 경우에 대해 능형회귀추정해 (6.6.88)을 계산한다. 이 능형회귀추정해를 그린 그래프가 그림 6.6.7에 그려진다. 그림 6.6.7에서 알 수 있듯이, λ가 커질수록 추정값들이 더 평활하다. 이 능형회귀추정해를 사용해서 예측한 결과가 그림 6.6.8에 그려진다. 그림 6.6.8에서 알 수 있듯이, 이 경우에는 λ에 따라 예측값들이 유의적으로 달라지지 않는다. ■

제6.7절 예제들

6.7.1 비선형최소제곱추정

다음 최적화문제를 살펴보자.

$$\underset{\boldsymbol{x} \in \mathbb{R}^n}{\text{Minimize}} \, \| A\boldsymbol{x} - \boldsymbol{b} \|^2 . \tag{6.7.1}$$

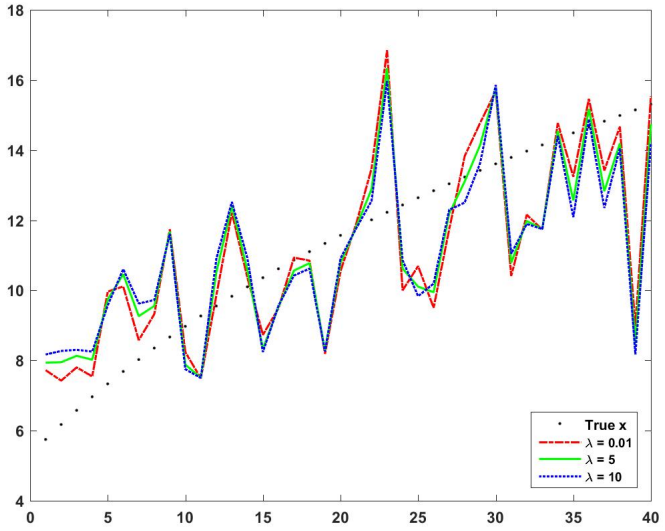


그림 6.6.7. 능형회귀추정해

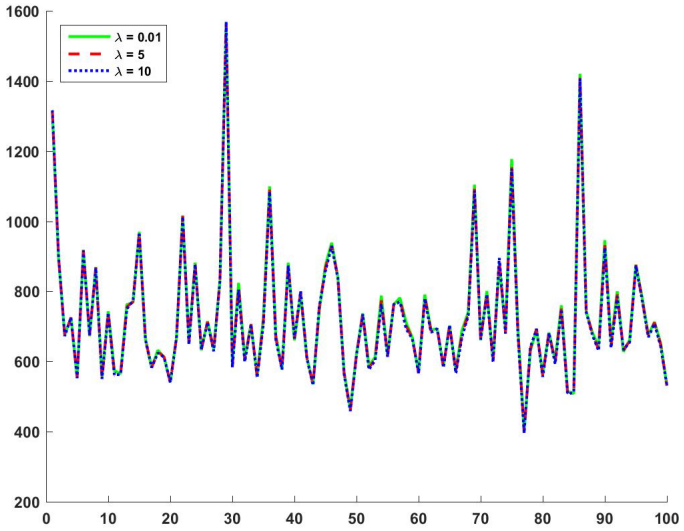


그림 6.6.8. 능형회귀예측값

여기서 행렬 $A\,(\in \mathbb{R}^{m\times n})$과 벡터들 $\boldsymbol{b}\,(\in \mathbb{R}^m)$와 $\boldsymbol{x}\,(\in \mathbb{R}^n)$는 각각 다음과 같다.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1^t \\ \boldsymbol{a}_2^t \\ \vdots \\ \boldsymbol{a}_m^t \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{6.7.2}$$

각 $i(=1,\,2,\,\cdots,\,m\,)$에 대해서 함수 $\epsilon_i : \mathbb{R}^n \mapsto \mathbb{R}$를 다음과 같이 정의하자.

$$\epsilon_i(\boldsymbol{x}) = \boldsymbol{a}_i^t\boldsymbol{x} - b_i \tag{6.7.3}$$

다음 식들이 성립한다.

$$f(\boldsymbol{x}) \doteq \|A\boldsymbol{x} - \boldsymbol{b}\,\|^2 = \sum_{i=1}^{m}\left[\boldsymbol{a}_i^t\boldsymbol{x} - b_i\,\right]^2 = \sum_{i=1}^{m}\epsilon_i^2(\boldsymbol{x}) \tag{6.7.4}$$

다음 식들이 성립한다.

$$\frac{\partial\epsilon_i^2(\boldsymbol{x})}{\partial x_j} = 2\epsilon_i(\boldsymbol{x})\frac{\partial\epsilon_i(\boldsymbol{x})}{\partial x_j} \tag{6.7.5}$$

$$\frac{\partial^2\epsilon_i^2(\boldsymbol{x})}{\partial x_k\partial x_j} = \frac{\partial}{\partial x_k}\left(2\epsilon_i(\boldsymbol{x})\frac{\partial\epsilon_i(\boldsymbol{x})}{\partial x_j}\right) = 2\frac{\partial\epsilon_i(\boldsymbol{x})}{\partial x_k}\frac{\partial\epsilon_i(\boldsymbol{x})}{\partial x_j} + 2\epsilon_i(\boldsymbol{x})\frac{\partial^2\epsilon_i(\boldsymbol{x})}{\partial x_k\partial x_j} \tag{6.7.6}$$

따라서 다음 식들이 성립한다.

$$\nabla f(\boldsymbol{x}) = \nabla\left(\sum_{i=1}^{m}\epsilon_i^2(\boldsymbol{x})\right) = 2\sum_{i=1}^{m}\epsilon_i(\boldsymbol{x})\nabla\epsilon_i(\boldsymbol{x}) \tag{6.7.7}$$

$$\nabla^2 f(\boldsymbol{x}) = \nabla^2\left(\sum_{i=1}^{m}\epsilon_i^2(\boldsymbol{x})\right) = 2\sum_{i=1}^{m}\nabla\epsilon_i(\boldsymbol{x})\nabla\epsilon_i(\boldsymbol{x})^t + 2\sum_{i=1}^{m}\epsilon_i(\boldsymbol{x})\nabla^2\epsilon_i(\boldsymbol{x}) \tag{6.7.8}$$

함수 $\epsilon_i(\boldsymbol{x})$의 점 $\boldsymbol{x} = \boldsymbol{x}_0$에서 선형근사식은 다음과 같다.

$$e_i(\boldsymbol{x};\boldsymbol{x}_0) = \epsilon_i(\boldsymbol{x}_0) + \nabla\epsilon_i(\boldsymbol{x}_0)^t\left[\boldsymbol{x} - \boldsymbol{x}_0\right] = \nabla\epsilon_i(\boldsymbol{x}_0)^t\boldsymbol{x} + \left[\epsilon_i(\boldsymbol{x}_0) - \nabla\epsilon_i(\boldsymbol{x}_0)^t\boldsymbol{x}_0\right] \tag{6.7.9}$$

다음 벡터들을 정의하자.

$$a_i(\boldsymbol{x}_0) \doteq \nabla\epsilon_i(\boldsymbol{x}_0), \quad b_i(\boldsymbol{x}_0) \doteq \nabla\epsilon_i(\boldsymbol{x}_0)^t\boldsymbol{x}_0 - \epsilon_i(\boldsymbol{x}_0) \tag{6.7.10}$$

다음과 같은 목적함수를 정의하자.

$$L(\boldsymbol{x};\boldsymbol{x}_0) \doteq \sum_{i=1}^{m}e_i^2(\boldsymbol{x};\boldsymbol{x}_0) = \sum_{i=1}^{m}\left[a_i(\boldsymbol{x}_0)^t\boldsymbol{x} - b_i(\boldsymbol{x}_0)\right]^2 \tag{6.7.11}$$

다음과 같은 행렬과 벡터들을 정의하자.

$$A(\boldsymbol{y}) \doteq \begin{bmatrix} \boldsymbol{a}_1(\boldsymbol{y})^t \\ \boldsymbol{a}_2(\boldsymbol{y})^t \\ \vdots \\ \boldsymbol{a}_m(\boldsymbol{y})^t \end{bmatrix}, \quad \boldsymbol{b}(\boldsymbol{y}) \doteq \begin{bmatrix} b_1(\boldsymbol{y}) \\ b_2(\boldsymbol{y}) \\ \vdots \\ b_m(\boldsymbol{y}) \end{bmatrix}, \quad \boldsymbol{e}(\boldsymbol{y}) \doteq \begin{bmatrix} e_1(\boldsymbol{y}) \\ e_2(\boldsymbol{y}) \\ \vdots \\ e_m(\boldsymbol{y}) \end{bmatrix} \tag{6.7.12}$$

다음 식이 성립한다.

$$L(\boldsymbol{x}; \boldsymbol{x}_0) = \|A(\boldsymbol{x}_0)\,\boldsymbol{x} - \boldsymbol{b}(\boldsymbol{x}_0)\|^2 \tag{6.7.13}$$

식 (6.7.13)에서 알 수 있듯이, 최적화문제 (6.7.1)을 다음과 같은 근사 최적화문제로 쓸 수 있다.

$$\underset{\boldsymbol{x}\in\mathbb{R}^n}{\text{Minimize}} \|A(\boldsymbol{x}_0)\,\boldsymbol{x} - \boldsymbol{b}(\boldsymbol{x}_0)\|^2 . \tag{6.7.14}$$

이 최소제곱합의 정규방정식은 다음과 같다.

$$A(\boldsymbol{x}_0)^t A(\boldsymbol{x}_0)\,\boldsymbol{x} = A(\boldsymbol{x}_0)^t \boldsymbol{b}(\boldsymbol{x}_0) \tag{6.7.15}$$

따라서, 다음 반복식을 적용해서 정규방정식 (6.7.15)의 해, 즉 최소제곱추정벡터들 $\{\boldsymbol{x}_k \,|\, k = 1, 2, \cdots\}$를 구할 수 있다.

$$A(\boldsymbol{x}_k)^t A(\boldsymbol{x}_k)\,\boldsymbol{x}_{k+1} = A(\boldsymbol{x}_k)^t \boldsymbol{b}(\boldsymbol{x}_k) \tag{6.7.16}$$

다음 식이 성립함을 쉽게 알 수 있다.

$$\boldsymbol{b}(\boldsymbol{x}_k) = A(\boldsymbol{x}_k)\boldsymbol{x}_k - \boldsymbol{e}(\boldsymbol{x}_k) \tag{6.7.17}$$

식 (6.7.17)를 사용해서 다음 식을 유도할 수 있다.

$$A(\boldsymbol{x}_k)^t \boldsymbol{b}(\boldsymbol{x}_k) = A(\boldsymbol{x}_k)^t A(\boldsymbol{x}_k)\boldsymbol{x}_k - \frac{1}{2}\nabla f(\boldsymbol{x}_k) \tag{6.7.18}$$

식 (6.7.18)를 식 (6.7.16)에 대입하면, 다음 점화식을 얻는다.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left[2A(\boldsymbol{x}_k)^t A(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k) \tag{6.7.19}$$

다음 식들이 성립한다.

$$2A(\boldsymbol{x}_k)^t A(\boldsymbol{x}_k) = 2\sum_{i=1}^{m} \nabla\epsilon_i(\boldsymbol{x}_k)\nabla\epsilon_i(\boldsymbol{x}_k)^t \tag{6.7.20}$$

$$\nabla^2 f(\boldsymbol{x}_k) = 2\sum_{i=1}^{m} \nabla\epsilon_i(\boldsymbol{x}_k)\nabla\epsilon_i(\boldsymbol{x}_k)^t + 2\sum_{i=1}^{m} \epsilon_i(\boldsymbol{x}_k)\nabla^2\epsilon_i(\boldsymbol{x}_k) \tag{6.7.21}$$

$$2\sum_{i=1}^{m} \nabla\epsilon_i(\boldsymbol{x}_k)\nabla\epsilon_i(\boldsymbol{x}_k)^t \neq 2\sum_{i=1}^{m} \nabla\epsilon_i(\boldsymbol{x}_k)\nabla\epsilon_i(\boldsymbol{x}_k)^t + 2\sum_{i=1}^{m} \epsilon_i(\boldsymbol{x}_k)\nabla^2\epsilon_i(\boldsymbol{x}_k) \tag{6.7.22}$$

여기서 식 (6.7.21)은 식 (6.7.8)에 의해서 성립한다. 식 (6.7.20) ∼ 식 (6.7.22)에서 알 수 있듯이, 점화식 (6.7.19)는 Newton-Raphson식과 다르다.

6.7.2 로지스틱회귀

다음과 같은 로지스틱회귀모형을 살펴보자.

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^t\boldsymbol{x}\right) = \frac{1}{1+\exp\left(-\boldsymbol{\theta}^t\boldsymbol{x}\right)} = \Pr(y=1 \mid \boldsymbol{x}\,;\,\boldsymbol{\theta}) \tag{6.7.23}$$

여기서 $\boldsymbol{\theta}\,(\in \mathbb{R}^n)$는 모수벡터이고, $\boldsymbol{x}\,(\in \mathbb{R}^n)$는 설명변수벡터이다.

주어진 관찰점들 $\{(\boldsymbol{x}_i, y_i)\,;\, i=1,2,\cdots,m\}$을 이용해서 이 로지스틱회귀모형을 추정하기 위해서, 다음과 같이 함수 $J(\boldsymbol{\theta})$를 정의하자.

$$J(\boldsymbol{\theta}) \doteq -\frac{1}{m}\sum_{i=1}^{m}\left\{y_i \ln\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_i\right)\right) + \left[1-y_i\right]\ln\left(1-h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_i\right)\right)\right\} + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{6.7.24}$$

여기서 $\boldsymbol{x}_i = [x_{i,1},\, x_{i,2},\, \cdots,\, x_{i,n}\,]^t$이다. 이에 해당하는 Newton-Raphson식은 다음과 같다.

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} - H^{-1}\nabla_{\boldsymbol{\theta}^{(k)}} J, \quad (k=0,1,\cdots) \tag{6.7.25}$$

여기서 $\nabla_{\boldsymbol{\theta}} J$는 다음과 같다.

$$\nabla_{\boldsymbol{\theta}} J = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^{m}\left[h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_i\right) - y_i\right]x_{i,1} + \frac{\lambda}{m}\theta_1 \\ \frac{1}{m}\sum_{i=1}^{m}\left[h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_i\right) - y_i\right]x_{i,2} + \frac{\lambda}{m}\theta_2 \\ \vdots \\ \frac{1}{m}\sum_{i=1}^{m}\left[h_{\boldsymbol{\theta}}\left(\boldsymbol{x}_i\right) - y_i\right]x_{i,n} + \frac{\lambda}{m}\theta_n \end{bmatrix} \tag{6.7.26}$$

이 $\nabla_{\boldsymbol{\theta}} J$는 $n \times 1$ 벡터이고, y_i는 스칼라이다. 또한 H는 다음과 같다.

$$H = \frac{1}{m} \sum_{i=1}^{m} h_{\boldsymbol{\theta}}(\boldsymbol{x}_i) \left[1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}_i)\right] \boldsymbol{x}_i \left[\boldsymbol{x}_i\right]^t + \frac{\lambda}{m} I_n \tag{6.7.27}$$

다음 예제는 Stanford대학 Andrew Ng교수의 머신러닝 강의록에서 인용한 것이다.

예제 6.7.1 Newton-Raphson법을 방정식 $\nabla_{\boldsymbol{\theta}} J = \mathbf{0}$의 근을 구하기 위해서, 다음 MATLAB프로그램 MachineLearningEx5L00.m을 실행하라.

```
% ------------------------------------------------------------------------
% Filename: MachineLearningEx5L00.m
% Regularized Logistic Regression w/ lambda=0.0
% Andrew Ng@Stanford University
% http://openclassroom.stanford.edu/MainFolder/DocumentPage.php
%        ?course=MachineLearning&doc=exercises/ex5/ex5.html
% ------------------------------------------------------------------------
function ex5Log
clear all; close all; clc
xy = xlsread('MachineLearningEx5');
x = xy(:,1:2);
y = xy(:,3);
whos
% Plot the training data
% Use different markers for positives and negatives
figure
pos = find(y); neg = find(y == 0);
plot(x(pos,1),x(pos,2),'k+','LineWidth', 2,'MarkerSize',7)
hold on
plot(x(neg,1),x(neg,2),'ko','MarkerFaceColor','y','MarkerSize',7)
% Add polynomial features to x by
% calling the feature mapping function
% provided in separate m-file
x = map_feature(x(:,1), x(:,2));
[m, n] = size(x);
% Initialize fitting parameters
theta = zeros(n, 1);
% Define the sigmoid function
g = inline('1.0 ./ (1.0 + exp(-z))');
% setup for Newton's method
MAX_ITR = 15;
J = zeros(MAX_ITR, 1);
% Lambda is the regularization parameter
lambda = 0

% Newton's Method
for i = 1:MAX_ITR
    % Calculate the hypothesis function
    z = x * theta;
    h = g(z);
    % Calculate J (for testing convergence)
    J(i) =(1/m)*sum(-y.*log(h) - (1-y).*log(1-h))+ ...
    (lambda/(2*m))*norm(theta([2:end]))^2;
    % Calculate gradient and hessian.
```

```
    G = (lambda/m).*theta; G(1) = 0; % extra term for gradient
    L = (lambda/m).*eye(n); L(1) = 0;% extra term for Hessian
    grad = ((1/m).*x' * (h-y)) + G;
    H = ((1/m).*x' * diag(h) * diag(1-h) * x) + L;
    % Here is the actual update
    theta = theta - H\grad;
end
% Show J to determine if algorithm has converged
J
% display the norm of our parameters
norm_theta = norm(theta)
% Plot the results
% We will evaluate theta*x over a
% grid of features and plot the contour
% where theta*x equals zero
% Here is the grid range
u = linspace(-1, 1.5, 200);
v = linspace(-1, 1.5, 200);
z = zeros(length(u), length(v));
% Evaluate z = theta*x over the grid
for i = 1:length(u)
    for j = 1:length(v)
        z(i,j) = map_feature(u(i), v(j))*theta;
    end
end
z = z'; % important to transpose z before calling contour
% Plot z = 0
% Notice you need to specify the range [0, 0]
contour(u, v, z, [0, 0], 'LineWidth', 2)
legend('y = 1', 'y = 0', 'Decision boundary')
title(sprintf('\\lambda = %g', lambda), 'FontSize', 14)
hold off
saveas(gcf,'MachineLearningEx5L00a.jpg')
% Uncomment to plot J
figure
plot(0:MAX_ITR-1,J,'o--','MarkerFaceColor','r','MarkerSize',8)
xlabel('Iteration'); ylabel('J')
saveas(gcf,'MachineLearningEx5L00b.jpg')
save('MachineLearningEx5L00.txt','J','norm_theta','-ascii')
end
%-------------------------------------------------------------------------
function out = map_feature(feat1, feat2)
% MAP_FEATURE    Feature mapping function for Exercise 5
%
%   map_feature(feat1, feat2) maps the two input features
%   to higher-order features as defined in Exercise 5.
%
%   Returns a new feature array with more features
%
%   Inputs feat1, feat2 must be the same size
%
% Note: this function is only valid for Ex 5, since the degree is
% hard-coded in.
    degree = 6;
    out = ones(size(feat1(:,1)));
    for i = 1:degree
        for j = 0:i
            out(:, end+1) = (feat1.^(i-j)).*(feat2.^j);
        end
    end
end
```

```
106 %--------------------------------------------------------------------------
```

이 MATLAB프로그램 MachineLearningEx5L00.m을 실행하면, Newton-Raphson알고리즘을 사용해서 $J(\boldsymbol{\theta})$의 최소값을 구한다. 즉, $\left\{J\left(\boldsymbol{\theta}^{(k)}\right)\ ;\ k=0,\,1,\,\cdots\right\}$를 계산한다. 정규모수 λ가 0인 경우, $\left\{J\left(\boldsymbol{\theta}^{(k)}\right)\right\}$를 그린 그래프가 그림 6.7.2에 그려진다. 그림 6.7.2에서 알 수 있듯이, 반복횟수 k가 10 이상이면 $J\left(\boldsymbol{\theta}^{(k)}\right)$는 거의 변화가 없다.

분류문제(classical problem)에서는 $\boldsymbol{\theta}$값에 따라 y가 어디에 속하는가를 결정한다. 다음 식을 만족하는 $\boldsymbol{x}$의 집합을 결정경계(decision boundary)라 한다.

$$\Pr\left(y=1\mid \boldsymbol{x}\ ;\boldsymbol{\theta}\right)=\frac{1}{2} \tag{2}$$

따라서, 결정경계는 다음과 같다.

$$\boldsymbol{\theta}^{t}\boldsymbol{x}=0 \tag{2}$$

정규모수 λ가 1인 경우, 결정경계가 그림 6.7.3에 그려져 있다. 또한 정규모수 λ가 10인 경우, 결정경계가 그림 6.7.4에 그려져 있다. 그림 6.7.2∼그림 6.7.4에서 알 수 있듯이, 정규모수 λ가 커질수록 결정경계가 평활(smooth) 형태임을 알 수 있다. ■

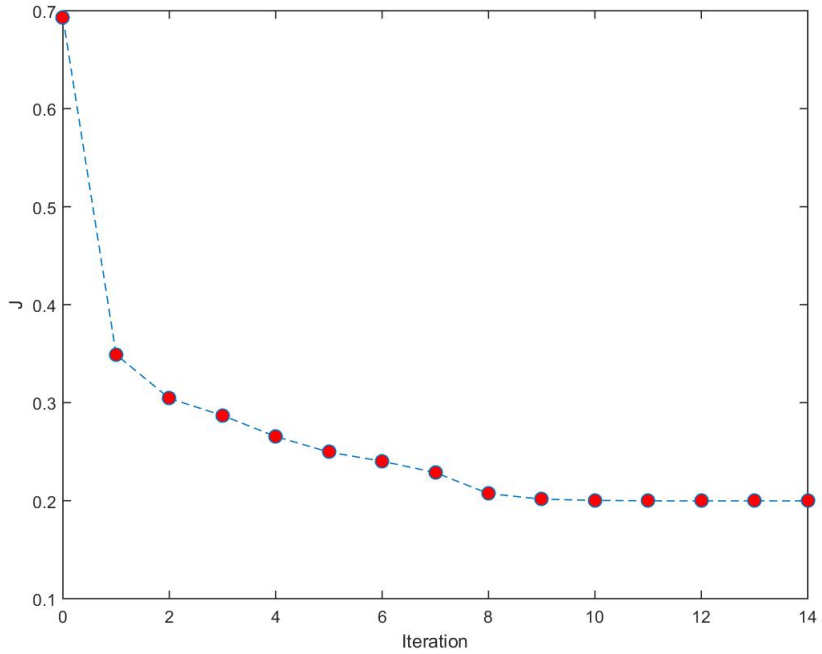


그림 6.7.1. 목적함수 $\{J(\boldsymbol{\theta}_i)\}$

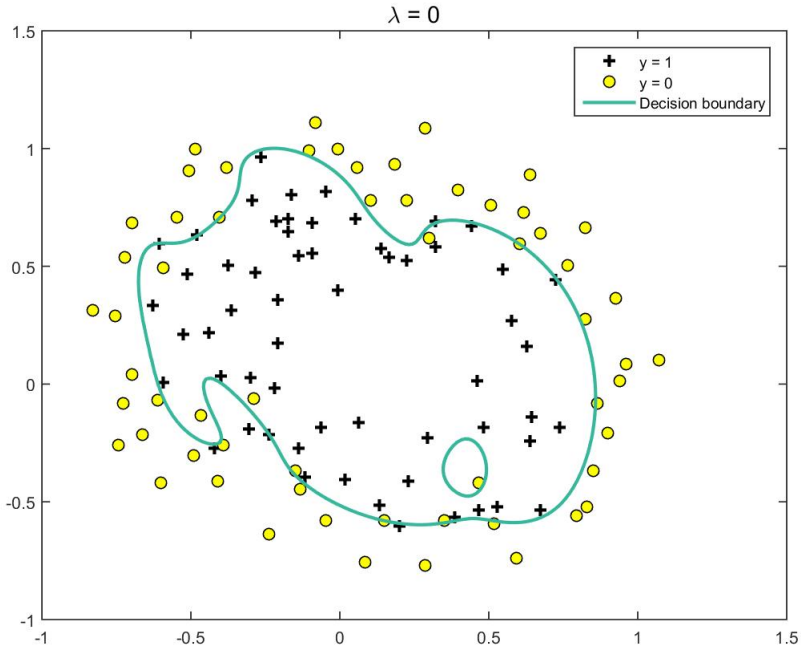


그림 6.7.2. 결정경계 $(\lambda = 0)$

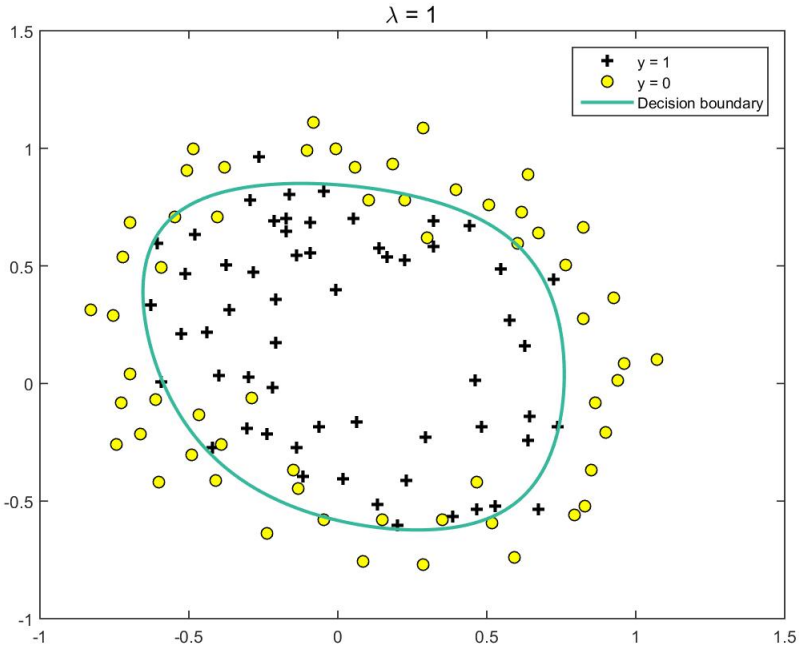


그림 6.7.3. 결정경계 ($\lambda = 1$)

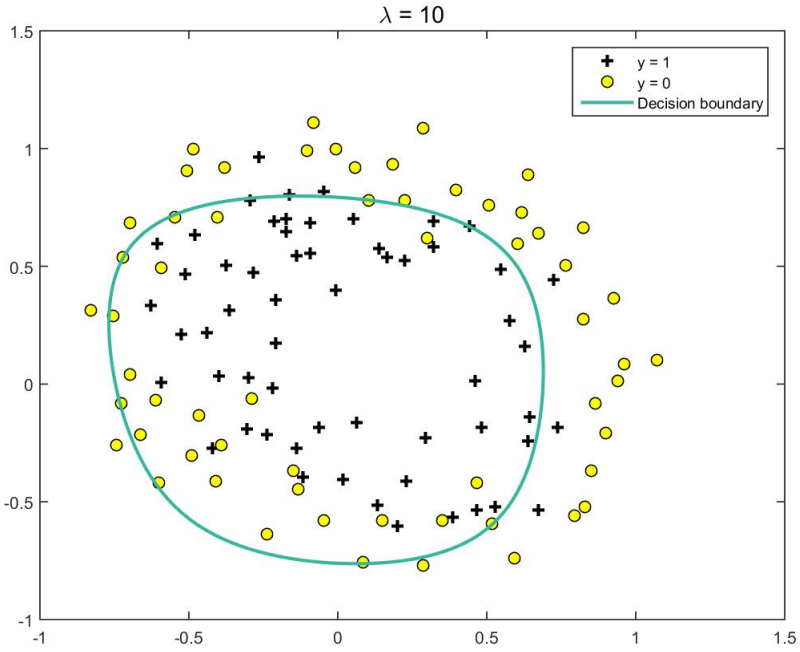


그림 6.7.4. 결정경계 ($\lambda = 10$)

제 7 장

미분과 미분방정식

함수가 식으로 주어지지 않고 데이터로 주어진 경우나, 미분방정식을 수치적으로 풀어야 하는 경우에 수치적으로 미분값을 구해야 할 필요가 있다. 모든 수치해석기법이 그러하듯, 수치미분에서도 오차를 작게 하는 것이 중요하다. 이 장에서는 오차에 중점을 두고 수치미분에 대해서 살펴보자. 학문의 주된 목적은 인과관계를 규명하는 것이고, 인과관계를 나타내는 것이 함수이다. 미분방정식이란 함수를 미지로 하는 방정식이므로, 실제 현상을 미분방정식으로 나타내고 이를 푸는 것이 그 현상을 나타내는 인과관계를 규명하는 것이다. 따라서 미분방정식은 학문 전반에 걸쳐 나타나며, 오늘날 재무학이나 경제학에서 미분방정식이 중요한 역할을 한다는 것을 모르는 독자는 없을 것이다. 이 장에서는 컴퓨터를 사용해서 미분방정식을 푸는 방법을 간단히 소개하고자 한다.

제7.1절 Richardson의 외삽법

함수 $f(x)$의 점 x에서 1차 미분의 정의는 다음과 같다.

$$f' = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{7.1.1}$$

다음과 같이 1차 도함수 $f'(x)$의 전향차분근사(forward difference approximation)를 정의하자.

$$D_1^f(x,h) \doteq \frac{f(x+h) - f(x)}{h} \tag{7.1.2}$$

Taylor정리에 의해서, 다음 식이 성립함을 알 수 있다.

$$D_1^f(x,h) = f'(x) + O(h) \tag{7.1.3}$$

다음과 같이 1차 도함수 $f'(x)$의 후향차분근사(backward difference approximation)를 정의하자.

$$D_1^b(x,h) \doteq \frac{f(x-h)-f(x)}{-h} \tag{7.1.4}$$

Taylor정리에 의해서, 다음 식이 성립함을 알 수 있다.

$$D_1^b(x,h) \doteq f'(x) + O(h) \tag{7.1.5}$$

다음과 같이 1차 도함수 $f'(x)$의 중심차분근사(central difference approximation)를 정의하자.

$$D_1^c(x,h) \doteq \frac{f(x+h)-f(x-h)}{2h} \tag{7.1.6}$$

Taylor정리에 의해서, 다음 식이 성립함을 알 수 있다.

$$D_1^c(x,h) = f'(x) + O(h^2) \tag{7.1.7}$$

전향차분근사식 (7.1.3)에서 가장 큰 오차항인 h의 1차항을 소거해서, 오차가 작은 근사식을 유도해보자. Taylor정리에서 알 수 있듯이, 다음 식들이 성립한다.

$$2D_1^f(x,h) = 2f'(x) + 2\frac{h}{2!}f''(x) + 2\frac{h^2}{3!}f^{(3)}(x) + O(h^3) \tag{7.1.8}$$

$$D_1^f(x,2h) = f'(x) + \frac{2h}{2!}f''(x) + \frac{2^2h^2}{3!}f^{(3)}(x) + O(h^3) \tag{7.1.9}$$

식 (7.1.8)과 식 (7.1.9)에서 알 수 있듯이, 다음 식이 성립한다.

$$2D_1^f(x,h) - D_1^f(x,2h) = f'(x) - 2\frac{h^2}{3!}f^{(3)}(x) + O(h^3) \tag{7.1.10}$$

식 (7.1.10)의 좌변을 $D_2^f(x,2h)$로 표기하자. 식 (7.1.2)에서 알 수 있듯이, 다음 식들이 성립한다.

$$D_2^f(x,2h) \doteq 2D_1^f(x,h) - D_1^f(x,2h) = \frac{-f(x+2h)+4f(x+h)-3f(x)}{2h} \tag{7.1.11}$$

식 (7.1.10)과 식 (7.1.11)에서 알 수 있듯이, 다음 식이 성립한다.

$$f'(x) = \frac{-f(x+2h)+4f(x+h)-3f(x)}{2h} + O(h^2) = D_2^f(x,2h) + O(h^2) \tag{7.1.12}$$

다음과 같이 $D_2^b(x,2h)$를 정의하자.

$$D_2^b(x,2h) \doteq 2D_1^b(x,h) - D_1^b(x,2h) \tag{7.1.13}$$

다음 식이 성립함은 자명하다.

$$D_i^b(x,kh) = D_i^f(x,-kh), \quad (k=1,2) \tag{7.1.14}$$

식 (7.1.12)와 식 (7.1.14)에서 알 수 있듯이, 다음 식들이 성립한다.

$$f'(x) = \frac{f(x-2h) - 4f(x-h) + 3f(x)}{2h} + O(h^2) = D_2^b(x,2h) + O(h^2) \tag{7.1.15}$$

식 (7.1.7)의 중심차분근사에서 가장 큰 오차항인 h의 2차항을 소거해서 오차가 작은 근사식을 유도해보자. Taylor정리에서 알 수 있듯이, 다음 식들이 성립한다.

$$4D_1^c(x,h) = 4f'(x) + 4\frac{h^2}{3!}f^{(3)} + 4\frac{h^4}{5!}f^{(5)}(x) + O(h^5) \tag{7.1.16}$$

$$D_1^c(x,2h) = f'(x) + \frac{2^2h^2}{3!}f^{(3)} + \frac{2^4h^4}{5!}f^{(5)}(x) + O(h^5) \tag{7.1.17}$$

식 (7.1.16)과 식 (7.1.17)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{4D_1^c(x,h) - D_1^c(x,2h)}{2^2-1} = f'(x) - 4\frac{h^4}{5!}f^{(5)}(x) + O(h^5) \tag{7.1.18}$$

식 (7.1.18)의 좌변을 $D_2^c(x,2h)$로 표기하면, 식 (7.1.6)에서 알 수 있듯이 다음 식이 성립한다.

$$D_2^c(x,2h) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} \tag{7.1.19}$$

식 (7.1.18)을 다음과 같이 쓸 수 있다.

$$f'(x) = D_2^c(x,2h) + O(h^4) \tag{7.1.20}$$

Richardson [48]이 제시한 외삽법은 수치해석 전반에서 아주 중요한 역할을 한다. 미지의 M을 함수 $N(h)$로 근사시킬 때, 오차가 다음과 같다고 하자.

$$M - N(h) = c_1h + c_2h^2 + c_3h^3 + \cdots \tag{7.1.21}$$

다음과 같이 함수들 $\{N_k(h)\,;\,k=1,2,\cdots\}$를 정의하자.

$$N_1(h) \doteq N(h) \tag{7.1.22}$$

$$N_n(h) \doteq N_{n-1}\left(\frac{h}{2}\right) + \frac{N_{n-1}\left(\frac{h}{2}\right) - N_{n-1}(h)}{2^{n-1}-1}, \quad (n=2,3,\cdots) \tag{7.1.23}$$

다음 식들이 성립함을 쉽게 증명할 수 있다.

$$M = N_n(h) + O(h^n), \quad (n=1,2,\cdots) \tag{7.1.24}$$

미지의 M을 함수 $L(h)$로 근사시킬 때, 오차가 다음과 같다고 하자.

$$M - L(h) = d_1h^2 + d_2h^4 + d_3h^6 + \cdots \tag{7.1.25}$$

다음과 같이 함수들 $\{L_k(h)\,;\,k=1,2,\cdots\}$를 정의하자.

$$L_1(h) \doteq L(h) \tag{7.1.26}$$

$$L_n(h) \doteq L_{n-1}\left(\frac{h}{2}\right) + \frac{L_{n-1}\left(\frac{h}{2}\right) - L_{n-1}(h)}{4^{n-1}-1}, \quad (n=2,3,\cdots) \tag{7.1.27}$$

다음 식들이 성립함을 쉽게 증명할 수 있다.

$$M = L_n(h) + O(h^{2n}), \quad (n=1,2,\cdots) \tag{7.1.28}$$

식 (7.1.23)과 식 (7.1.27)을 Richardson의 외삽법이라 한다.

식 (7.1.12)와 식 (7.1.15)에서 알 수 있듯이, 1차 도함수 $f'(x)$의 전향차분근사나 후향차분근사에 Richardson의 외삽법 (7.1.23)를 적용할 수 있다. 식 (7.1.19)에서 알 수 있듯이, 1차 도함수 $f'(x)$의 중심차분근사에 Richardson의 외삽법 (7.1.27)를 적용할 수 있다. 이를 높은 차수로 확장하기 위해서, 다음과 같은 함수들을 정의하자.

$$D^f_{n+1}(x,h) \doteq D^f_n\left(x,\frac{h}{2}\right) + \frac{D^f_n\left(x,\frac{h}{2}\right) - D^f_n(x,h)}{2^n-1}, \quad (n=1,2,\cdots) \tag{7.1.29}$$

$$D^b_{n+1}(x,h) \doteq D^b_n\left(x,\frac{h}{2}\right) + \frac{D^b_n\left(x,\frac{h}{2}\right) - D^b_n(x,h)}{2^n-1}, \quad (n=1,2,\cdots) \tag{7.1.30}$$

$$D^c_{n+1}(x,h) \doteq D^c_n\left(x,\frac{h}{2}\right) + \frac{D^c_n\left(x,\frac{h}{2}\right) - D^c_n(x,h)}{4^n-1}, \quad (n=1,2,\cdots) \tag{7.1.31}$$

식 (7.1.24)와 식 (7.1.28)에서 알 수 있듯이, 다음 식들이 성립한다.

$$f'(x) = D_n^f(x,h) + O(h^n), \quad (n = 1, 2, \cdots) \tag{7.1.32}$$

$$f'(x) = D_n^b(x,h) + O(h^n), \quad (n = 1, 2, \cdots) \tag{7.1.33}$$

$$f'(x) = D_n^c(x,h) + O(h^{2n}), \quad (n = 1, 2, \cdots) \tag{7.1.34}$$

예제 7.1.1 Richardson의 외삽법을 사용해서 수치미분을 하기 위해서, 다음 MATLAB 프로그램 RichardsonDerivative101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: RichardsonDerivative101.m
%  Richardson method for numerical derivatives
%  Programmed by CBS
% ------------------------------------------------------------------
clear all; close all, format short
y = @(x) exp(x)
nMax = 11;
h0 = 1/2;
a = 1;
% forward-difference
dF = zeros(nMax,nMax);
h = h0;
for hi = 0:1:nMax-1
    dF(hi+1,1) = (y(a + h)-y(a))/h;
   for nn = 1:1:hi
     dF(hi+1,nn+1) = dF(hi+1,nn)+(dF(hi+1,nn)-dF(hi,nn))/(2^nn-1);
   end
   h = h/2;
end
fprintf('\n Forward Difference approximation \n')
ForRichard = dF(1:5,1:5)
% backward-difference
dB = zeros(nMax,nMax);
h = h0;
for hi = 0:1:nMax-1
    dB(hi+1,1) = (y(a)-y(a - h))/h;
   for nn = 1:1:hi
       dB(hi+1,nn+1) = dB(hi+1,nn) ...
                     + (dB(hi+1,nn)-dB(hi,nn))/(2^nn-1);
   end
   h = h/2;
end
fprintf('\n Backward Difference approximation \n\n')
BackRichard = dB(1:5,1:5)
% Central-difference
dC = zeros(nMax,nMax);
h = h0;
for hi = 0:1:nMax-1
    dC(hi+1,1) = (y(a + h)-y(a - h))/(2*h);
   for nn = 1:1:hi
       dC(hi+1,nn+1) = dC(hi+1,nn) ...
                     + (dC(hi+1,nn)-dC(hi,nn))/(4^nn-1);
   end
   h = h/2;
```

```
end
fprintf('\n Cental Difference approximation \n\n')
CentRichard = dC(1:5,1:5)
% Plotting
nn = (1:1:nMax)';
dFdiag = diag(dF); dBdiag = diag(dB); dCdiag = diag(dC);
plot(nn,dFdiag,'b-',nn,dBdiag,'g--',nn,dCdiag,'r-.' ...
     ,'LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
legend('Forward Difference','Backward Difference', ...
       'Central Difference',4)
xlabel('\bf Iteration Number','fontsize',12)
ylabel('\bf Approximation','fontsize',12)
saveas(gcf,'RichardsonDerivative101','jpg')
save('RichardsonDerivative101.txt','CentRichard','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 명령문을 실행하면, Richardson의 외삽법을 적용해서 함수 $y = e^x$의 점 $x = 1$에서 전향차분근사, 후향차분근사 그리고 중심차분근사에 의한 미분값들을 계산한다. 여기서 $h = 0.5$이다.

전향차분근사에 의한 결과는 다음과 같다.

```
ForRichard =
    3.5268         0         0         0         0
    3.0882    2.6497         0         0         0
    2.8955    2.7027    2.7204         0         0
    2.8050    2.7146    2.7185    2.7183         0
    2.7612    2.7174    2.7183    2.7183    2.7183
```

여기서 행렬 ForRichard의 제(i,j) 원소는 $D_i^f\left(1, \frac{h}{2^{j-i}}\right)$이다. 또한, 이 원소들을 계산하는 순서는 다음과 같다.

$$(1,1) \to (2,1) \to (2,2) \to (3,1) \to (3,2) \to (3,3) \to (4,1) \to (4,2) \to \cdots \tag{1}$$

후향차분근사에 의한 결과는 다음과 같다.

```
BackRichard =
    2.1391         0         0         0         0
    2.4051    2.6711         0         0         0
    2.5553    2.7054    2.7168         0         0
    2.6351    2.7149    2.7181    2.7183         0
    2.6762    2.7174    2.7183    2.7183    2.7183
```

여기서 행렬 BackRichard의 제(i,j) 원소는 $D_i^b\left(1, \frac{h}{2^{j-i}}\right)$이다. 또한, 이 원소들을 계산하는 순서는 식 (1)과 같다.

중심차분근사에 의한 결과는 다음과 같다.

```
CentRichard =

    2.8330         0         0         0         0
    2.7467    2.7179         0         0         0
    2.7254    2.7183    2.7183         0         0
    2.7201    2.7183    2.7183    2.7183         0
    2.7187    2.7183    2.7183    2.7183    2.7183
```

여기서 행렬 CentRichard의 제(i,j)원소는 $D_i^c\left(1,\frac{h}{2^{j-i}}\right)$이다. 또한, 이 원소들을 계산하는 순서는 식 (1)과 같다. ■

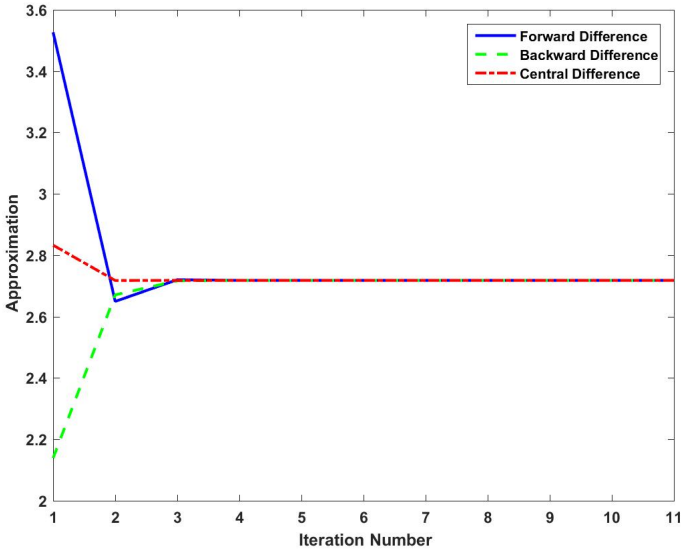


그림 7.1.1. Richardson외삽법에 의한 수치미분

예제 7.1.2 Python을 사용해서 예제 7.1.1을 다시 다루기 위해서, 다음 Python프로그램 RichardsonDerivative101.Py를 실행해 보자.

```
"""
%  Filename: RichardsonDerivative101.Py
%  Richardson method for numerical derivatives
%  Programmed by CBS
"""

import numpy as np
import matplotlib.pyplot as plt

y = lambda x: np.exp(x)
nMax = 11;
h0 = 1/2;
a = 1;
# forward-difference
```

```
dF = np.zeros((nMax,nMax));
h = h0;
a = 1
for hi in range(-1,nMax-1):
    dF[hi+1,0] = (y(a + h)-y(a))/h;
    for nn in range(0,hi+1):
        dF[hi+1,nn+1] = dF[hi+1,nn]+(dF[hi+1,nn]-dF[hi,nn])/(2**(nn+1)-1);
    h = h/2;
print('Forward Difference approximation')
ForRichard = dF[0:5,0:5]
print(ForRichard)

# backward-difference
dB = np.zeros((nMax,nMax));
h = h0;
for hi in range(-1,nMax-1):
    dB[hi+1,0] = (y(a)-y(a - h))/h;
    for nn in range(0,hi+1):
        dB[hi+1,nn+1] = dB[hi+1,nn]+(dB[hi+1,nn]-dB[hi,nn])/(2**(nn+1)-1);
    h = h/2;
print('Backward Difference approximation')
BackRichard = dB[0:5,0:5]
print(BackRichard)
# Central-difference
dC = np.zeros((nMax,nMax));
h = h0;
for hi in range(-1,nMax-1):
    dC[hi+1,0] = (y(a + h)-y(a - h))/(2*h);
    for nn in range(0,hi+1):
        dC[hi+1,nn+1] = dC[hi+1,nn]+(dC[hi+1,nn]-dC[hi,nn])/(4**(nn+1)-1);
    h = h/2;
print('Cental Difference approximation')
CentRichard = dC[0:5,0:5]
print(CentRichard)

# plotting
fig = plt.figure()
nn = np.arange(nMax)
dFdiag = np.diag(dF);
dBdiag = np.diag(dB);
dCdiag = np.diag(dC);
plt.plot(nn,dFdiag,'b-',lw=2.0,label='Forward Difference')
plt.plot(nn,dBdiag,'g--',lw=2.0,label='Backward Difference')
plt.plot(nn,dCdiag,'r-.',lw=2.0,label='Central Difference')
plt.xlabel('Iteration Number',fontsize=12)
plt.ylabel('Approximation',fontsize=12)
plt.show()
fig.savefig('RichardsonDerivative101Py.png')

# End of program
```

이 Python프로그램을 수행한 결과는 예제 7.1.1의 결과와 같다. ■

제7.2절 절단오차와 반올림오차

제7.1절에서 1차 미분의 오차에 대해서 살펴보았다. 작은 h에 대한 오차 $O(h^n)$은 각 숫자를 정확히 표현했을 때 Taylor근사식에 의한 절단오차(truncation error)를 나타낸다. 즉, 각 숫자를 유한 자리수로 표현하는 양자화(quantization)에 따른 반올림오차(round-off error)는 고려되지 않는다. 이 소절에서는 절단오차와 함께 반올림오차를 생각해보자.

함수값들 $f(x+2h), f(x+h), f(x), f(x-h), f(x-2h)$을 양자화한 값들이 각각 $y_2, y_1, y_0, y_{-1}, y_{-2}$이고 반올림오차들이 각각 $e_2, e_1, e_0, e_{-1}, e_{-2}$라고 하면, 다음 식들이 성립한다.

$$y_k = f(x+kh) + e_k, \quad (k = 2, 1, 0, -1, -2) \tag{7.2.1}$$

또한, 각 반올림오차의 절대값은 ϵ보다 작다고 가정하자.

반올림오차를 고려하면, 전향차분근사식이 다음 식들을 만족함을 알 수 있다.

$$\begin{aligned} D_1^f(x,h) &= \frac{y_1 - y_0}{h} = \frac{f(x+h) + e_1 - f(x) - e_0}{h} \\ &= f'(x) + \frac{e_1 - e_0}{h} + \frac{1}{2}K_1 h \end{aligned} \tag{7.2.2}$$

여기서 $K_1 = f''(c_1),\ c_1 \in (x, x+h)$이다. 식 (7.2.2)에서 알 수 있듯이, 다음 식이 성립한다.

$$|D_1^f(x,h) - f'(x)| \le \frac{2\epsilon}{h} + \frac{1}{2}|K_1|h \tag{7.2.3}$$

여기서 절단오차는 $|K_1|h/2$이고 반올림오차는 $2\epsilon/h$이다. 상가상승평균부등식에서 알 수 있듯이, 다음 식들이 성립한다.

$$\frac{2\epsilon}{h} + \frac{1}{2}|K_1|h \ge 2\sqrt{\frac{2\epsilon}{h} \cdot \frac{1}{2}|K_1|h} = 2\sqrt{\epsilon|K_1|} \tag{7.2.4}$$

여기서 등호는 $h^* \doteq 2\sqrt{\frac{\epsilon}{|K_1|}}$에서 성립하므로, 이 h^*에서 절단오차와 반올림오차의 상한을 최소화한다.

반올림오차를 고려하면, 중심차분근사식이 다음 식들을 만족함을 알 수 있다.

$$\begin{aligned} D_1^c(x,h) &= \frac{y_1 - y_{-1}}{2h} \\ &= \frac{f(x+h) + e_1 - f(x-h) - e_{-1}}{2h} = f'(x) + \frac{e_1 - e_{-1}}{2h} + \frac{1}{6}K_2 h^2 \end{aligned} \tag{7.2.5}$$

여기서 $K_2 = f^{(3)}(c_2),\ c_2 \in (x-h, x+h)$ 이다. 식 (7.2.5) 에서 알 수 있듯이, 다음 식이 성립한다.

$$|D_1^c(x,h) - f'(x)| \le \frac{\epsilon}{h} + \frac{1}{6}|K_2|h^2 2 \tag{7.2.6}$$

여기서 절단오차는 $|K_2|h^2/6$ 이고 반올림오차는 ϵ/h 이다. 상가상승평균부등식에서 알 수 있듯이, 다음 식들이 성립한다.

$$\frac{\epsilon}{h} + \frac{1}{6}|K_2|h^2 = \frac{\epsilon}{2h} + \frac{\epsilon}{2h} + \frac{1}{6}|K_2|h^2 \ge 3\sqrt[3]{\frac{\epsilon}{2h}\cdot\frac{\epsilon}{2h}\cdot\frac{1}{6}|K_2|h^2} = \left[\frac{9}{8}\epsilon^2|K_2|\right]^{1/3} \tag{7.2.7}$$

여기서 등호는 $h^* \doteq \left[\frac{3\epsilon}{|K_2|}\right]^{1/3}$ 에서 성립하므로, 이 h^* 에서 절단오차와 반올림오차의 상한을 최소화한다.

반올림오차를 고려하면, $D_2^c(x,2h)$ 가 다음 식들을 만족함을 알 수 있다.

$$\begin{aligned}
&|D_2^c(x,2h) - f'(x)| \\
&= \left|\frac{-f(x+2h) - e_2 + 8f(x+h) + 8e_1 - 8f(x-h) - 8e_{-1} + f(x-2h) + e_{-2}}{12h} - f'(x)\right| \\
&= \left|\frac{-e_2 + 8e_1 - 8e_{-1} + e_{-2}}{12h}\right| + \frac{1}{30}|K_4|h^4
\end{aligned} \tag{7.2.8}$$

여기서 $K_4 = f^{(5)}(c_4),\ c_4 \in (x-2h, x+2h)$ 이다. 식 (7.2.8) 에서 알 수 있듯이, 다음 식이 성립한다.

$$|D_2^c(x,h) - f'(x)| \le \frac{3\epsilon}{2h} + \frac{1}{30}|K_4|h^4 \tag{7.2.9}$$

여기서 절단오차는 $|K_4|h^4/30$ 이고 반올림오차는 $1.5\epsilon/h$ 이다. 다음 식들이 성립한다.

$$\begin{aligned}
&\frac{3\epsilon}{2h} + \frac{1}{30}|K_4|h^4 = \frac{3\epsilon}{8h} + \frac{3\epsilon}{8h} + \frac{3\epsilon}{8h} + \frac{3\epsilon}{8h} + \frac{1}{30}|K_4|h^4 \\
&\ge 4\sqrt[5]{\frac{3\epsilon}{8h}\cdot\frac{3\epsilon}{8h}\cdot\frac{3\epsilon}{8h}\cdot\frac{3\epsilon}{8h}\cdot\frac{1}{30}|K_4|h^4} = \left[\frac{27}{40}\epsilon^4|K_4|\right]^{1/5}
\end{aligned} \tag{7.2.10}$$

여기서 등호는 $h^* \doteq \left[\frac{45\epsilon}{4|K_4|}\right]^{1/5}$ 에서 성립하므로, 이 h^* 에서 절단오차와 반올림오차의 상한을 최소화한다.

지금까지 내용에서 알 수 있듯이, h 가 작아지면 절단오차는 작아지지만 반올림오차는 커진다. 즉, 일종의 불확정성 현상이 발생한다. 이를 스텝크기딜레마 (step size dilemma) 라 부른다. 결론적으로, 수치미분의 오차를 줄이기 위해서는 h 를 너무 작지도 너무 크지도 않게 적당한 것을 선택해야 한다. 식 (7.2.4), 식 (7.2.7) 그리고 식 (7.2.10) 이 그 선택기준이 될 수도 있다.

제7.3절 고차 수치미분

이 절에서는 고차 미분값을 수치적으로 구하는 방법에 대해서 살펴보자. 여기서 사용하는 방법은 근본적으로 앞에서 다룬 1차 미분값을 수치적으로 구하는 방법과 같다.

다음 식들이 성립한다.

$$f(x+h) = \sum_{k=0}^{5} \frac{1}{k!} f^{(k)}(x) h^k + O(h^6) \tag{7.3.1}$$

$$f(x-h) = \sum_{k=0}^{5} \frac{1}{k!} f^{(k)}(x) [-h]^k + O(h^6) \tag{7.3.2}$$

따라서, 다음 식들이 성립한다.

$$\begin{aligned} E_1^c(x,h) &\doteq \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \\ &= f''(x) + \frac{2}{4!} f^{(4)}(x) h^2 + \frac{2}{6!} f^{(6)}(x) h^4 + O(h^6) \end{aligned} \tag{7.3.3}$$

즉, $E_1^c(x,h)$는 $f''(x)$의 근사식이고 오차는 $O(h^2)$이다. 식 (7.3.3)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{4E_1^c(x,h) - E_1^c(x,2h)}{2^2 - 1} = f''(x) - \frac{1}{90} f^{(6)}(x) h^4 + O(h^6) \tag{7.3.4}$$

식 (7.3.4)의 좌변을 $E_2^c(x,2h)$로 표기하자. 즉, 다음 식이 성립한다.

$$E_2^c(x,2h) = \frac{-f(x+2h) + 16f(x+h) - 30f(x) + 16f(x-h) - f(x-2h)}{12h^2} \tag{7.3.5}$$

즉, $E_2^c(x,2h)$는 $f''(x)$의 근사식이고 오차는 $O(h^4)$이다.

식 (7.3.3)에서 알 수 있듯이, $E_1^c(x,h)$는 식 (7.1.25)를 만족한다. 따라서, 식 (7.1.27)을 적용하면, 오차가 $O(h^{2n})$인 2차 미분값의 근사값 $E_n^c(x,h)$를 구할 수 있다. 또한, 제7.1절의 기법을 적용하면, 다음 예제에서와 같이 고차 미분의 근사식을 구할 수 있다.

예제 7.3.1 고차 미분에 관한 차분근사식을 발생시키는 알고리즘을 살펴보자. 고정된 h에 대해서 다음 변수들을 정의하자.

$$f_k \doteq f(x+kh), \quad (k = 0, \pm 1, \pm 2, \cdots) \tag{1}$$

$$f_k^{(l)} \doteq f^{(l)}(x+kh), \quad (l = 0, 1, 2, \cdots, k = 0, \pm 1, \pm 2, \cdots) \tag{2}$$

예를 들어, 다음과 같은 2차 미분의 근사함수 $D(x,h)$을 유도해보자.

$$D(x,h) = \frac{c_2 f_2 + c_1 f_1 + c_0 f_0 + c_{-1} f_{-1} + c_{-2} f_{-2}}{h^2} \tag{3}$$

Taylor정리에 의해서, 다음 식이 성립함을 알 수 있다.

$$\begin{aligned} h^2 D(x,h) =& c_2 \sum_{j=0}^{\infty} \frac{1}{j!} f_0^{(j)} [2h]^j + c_1 \sum_{j=0}^{\infty} \frac{1}{j!} f_0^{(j)} h^j \\ &+ c_0 f_0 + c_{-1} \sum_{j=0}^{\infty} \frac{1}{j!} f_0^{(j)} [-h]^j + c_{-2} \sum_{j=0}^{\infty} \frac{1}{j!} f_0^{(j)} [-2h]^j \end{aligned} \tag{4}$$

식 (4)의 좌변에 $D(x,h)$ 대신 $f''(x)$를 대입한 다음 항등식정리를 적용하면, 다음 방정식들이 성립함을 알 수 있다.

$$c_2 + c_1 + c_0 + c_{-1} + c_{-2} = 0 \tag{5}$$

$$\frac{2^1}{1!} c_2 + \frac{1}{1!} c_1 + \frac{0}{1!} c_0 - \frac{1}{1!} c_{-1} - \frac{2^1}{1!} c_{-2} = 0 \tag{6}$$

$$\frac{2^2}{2!} c_2 + \frac{1}{2!} c_1 + \frac{0}{2!} c_0 + \frac{1}{2!} c_{-1} + \frac{2^2}{2!} c_{-2} = 1 \tag{7}$$

$$\frac{2^3}{3!} c_2 + \frac{1}{3!} c_1 + \frac{0}{3!} c_0 - \frac{1}{3!} c_{-1} - \frac{2^3}{3!} c_{-2} = 0 \tag{8}$$

$$\frac{2^4}{4!} c_2 + \frac{1}{4!} c_1 + \frac{0}{4!} c_0 + \frac{1}{4!} c_{-1} + \frac{2^4}{4!} c_{-2} = 0 \tag{9}$$

이 연립방정식을 풀면, 그 해는 다음과 같다.

$$c_2 = -\frac{1}{12}, \quad c_1 = \frac{16}{12}, \quad c_0 = -\frac{30}{12}, \quad c_{-1} = \frac{16}{12}, \quad c_{-2} = -\frac{1}{12} \tag{10}$$

이 고차 미분에 관한 차분근사식을 발생시키는 알고리즘을 수행하기 위해서, 다음 MATLAB 프로그램 DifferntialApproxCoeff101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: DifferentialApproxCoeff101.m
%  Coefficients for numerical differential approximation
%  Programmed by CBS
% ------------------------------------------------------------------
function DifferentialApproxCoeff101
clear all, close all, format rat
N = 2     % Order of Approximate Differential, (Ex) f^(2)
MAXpt = 2; MINpt = -2;
points = MAXpt:-1:MINpt  % points using for derivatives
                         % (Ex) f_2, f_1, f_0, f_-1, f_-2
```

```
LNpt = MAXpt - MINpt +1;
ll = MAXpt;
for nn = 1:1:LNpt
    A(1,nn) = 1;
    for mm = 2:1:LNpt
        A(mm,nn) = A(mm-1,nn)*ll/(mm-1);
    end
    ll = ll-1;
end
b = zeros(LNpt,1);
b(N+1) = 1;
fprintf('[ A; b ] = \n\n'), disp([ A, b ])
c = (A\b)'
save('DifferentialApproxCoeff101.txt','A','b','c','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행한 결과는 다음과 같다. 이 결과물에서 식 (10)이 올바르게 구해졌음을 확인할 수 있다.

```
N =  2

points =  2        1         0        -1        -2


[ A; b ] =

      1        1        1         1         1         0

      2        1        0        -1        -2         0

      2      1/2        0       1/2         2         1

    4/3      1/6        0      -1/6      -4/3         0

    2/3     1/24        0       1/24       2/3         0


c =  -1/12      4/3      -5/2       4/3      -1/12
```

■

예제 7.3.2 Python을 사용해서 예제 7.3.1을 다시 다루기 위해서, 다음 Python프로그램 DifferentialApproxCoeff101.Py를 실행해 보자.

```
"""
%  Filename: DifferentialApproxCoeff101.m
%  Coefficients for numerical differential approximation
%  Programmed by CBS
"""

from sympy import Symbol, limit, oo

x = Symbol('x'); y = Symbol('y'); t = Symbol('t');
L1 = limit((x**2-3*x+2)/(x**2-1),x,1)
L2 = limit((x**2-3*x+2)/(x**2-1),x,oo)   # oo = infty
```

```
L30 = limit(abs(x)/x,x,0, '-')
L31=  limit(abs(x)/x,x,0, dir='-')
L40 = limit(abs(x)/x,x,0, '+')
L41=  limit(abs(x)/x,x,0, dir='+')
print('L1 =',L1, ', L2 =',L2, ', L30 =',L30, ', L31 =',L31, \
      ', L40 =',L40, ', L41 =',L41)
f1 = (x**2 - y**2)/(x**2+y**2)
L5 = limit(limit(f1,x,2),y,1)
L6 = limit(limit(f1,y,1),x,2)
L7 = limit(limit(f1,x,0),y,0)
L8 = limit(limit(f1,y,0),x,0)
print('L5 =',L5, ', L6 =',L6, ', L7 =',L7, ', L8 =',L8)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 7.3.1의 결과와 같다. ■

예제 7.3.1의 알고리즘을 사용해서 구한 미분의 전향차분근사식을 정리한 것이 표 7.3.1, 후향차분근사식을 정리한 것이 표 7.3.2, 그리고 중심차분근사식을 정리한 것이 표 7.3.3이다. 이 표들에서 다음과 같은 표기법을 사용했음을 상기하라.

$$f_k = f(x+kh), \quad (k = 0, \pm 1, \pm 2, \cdots) \tag{7.3.6}$$

$$f'(x) = \frac{f_1 - f_0}{h} + O(h)$$

$$f'(x) = \frac{-f_2 + 4f_1 - 3f_0}{2h} + O(h^2)$$

$$f''(x) = \frac{f_2 - 2f_1 + f_0}{h^2} + O(h)$$

$$f''(x) = \frac{-f_3 + 4f_2 - 5f_1 + 2f_0}{h^2} + O(h^2)$$

$$f^{(3)}(x) = \frac{f_3 - 3f_2 + 3f_1 - f_0}{h^3} + O(h)$$

$$f^{(3)}(x) = \frac{-3f_4 + 14f_3 - 24f_2 + 18f_1 - 5f_0}{2h^3} + O(h^2)$$

$$f^{(4)}(x) = \frac{f_4 - 4f_3 + 6f_2 - 4f_1 + f_0}{h^4} + O(h)$$

$$f^{(4)}(x) = \frac{-2f_5 + 11f_4 - 24f_3 + 26f_2 - 14f_1 + 3f_0}{h^4} + O(h^2)$$

표 7.3.1. 미분의 전향차분근사식

$f'(x) = \dfrac{f_0 - f_{-1}}{h} + O(h)$
$f'(x) = \dfrac{f_{-2} - 4f_{-1} + 3f_0}{2h} + O(h^2)$
$f''(x) = \dfrac{f_{-2} - 2f_{-1} + f_0}{h^2} + O(h)$
$f''(x) = \dfrac{-f_{-3} + 4f_{-2} - 5f_{-1} + 2f_0}{h^2} + O(h^2)$
$f^{(3)}(x) = \dfrac{-f_{-3} + 3f_{-2} - 3f_{-1} + f_0}{h^3} + O(h)$
$f^{(3)}(x) = \dfrac{3f_{-4} - 14f_{-3} + 24f_{-2} - 18f_{-1} + 5f_0}{2h^3} + O(h^2)$
$f^{(4)}(x) = \dfrac{f_{-4} - 4f_{-3} + 6f_{-2} - 4f_{-1} + f_0}{h^4} + O(h)$
$f^{(4)}(x) = \dfrac{-2f_{-5} + 11f_{-4} - 24f_{-3} + 26f_{-2} - 14f_{-1} + 3f_0}{h^4} + O(h^2)$

표 7.3.2. 미분의 후향차분근사식

제7.4절 부등간격으로 기록된 데이터의 수치미분

지금까지 다룬 방법은 함수가 주어졌거나 데이터가 등간격으로 주어진 경우 수치미분을 하기 위한 것이다. 그러나, 실험을 통해 데이터를 수집하는 경우, 관찰간격이 동일하지 않은 경우가 있다. 이렇게 부등간격으로 기록된 데이터를 수치미분하기 위해서는, 먼저 보간다항식을 구하고 다음으로 수치미분을 하는 것이 일반적이다. 예를 들어, 데이터가 등간격으로 주어지지 않았다 하더라도 Lagrange보간식이나 Hermite보간식을 사용해서 수치미분을 할 수 있다. 우선, 이 보간식을 구한 다음, 이 보간식을 수치미분해서 원래 함수의 수치미분값을 추정하는 것이다. 예를 들어, 서로 다른 세 점들 (x_0, y_0), (x_1, y_1), 그리고 (x_2, y_2)을 지나는 Lagrnage 보간식을 구한 다음, 이 보간식을 미분하면 다음 식을 얻는다.

$$\begin{aligned} f'(x) = f(x_0)\frac{2x - x_1 - x_2}{[x_0 - x_1][x_0 - x_2]} + f(x_1)\frac{2x - x_0 - x_2}{[x_1 - x_0][x_1 - x_2]} \\ + f(x_2)\frac{2x - x_0 - x_1}{[x_2 - x_0][x_2 - x_1]} \end{aligned} \tag{7.4.1}$$

편의상, $x_0 < x_1 < x_2$라고 하자. 이렇게 구한 수치미분은 다음과 같은 장점을 가지고 있다. 첫째, x값들이 등간격을 이루지 않아도 된다. 둘째, 구간 (x_0, x_2) 내 어떤 점에서도 미분값을

$f'(x) = \dfrac{f_1 - f_{-1}}{2h} + O(h^2)$
$f'(x) = \dfrac{-f_{-2} + 8f_1 - 8f_{-1} + f_{-2}}{12h} + O(h^4)$
$f''(x) = \dfrac{f_1 - 2f_0 + f_{-1}}{h^2} + O(h^2)$
$f''(x) = \dfrac{-f_2 + 16f_1 - 30f_0 + 16f_{-1} - f_{-2}}{12h^2} + O(h^4)$
$f^{(3)}(x) = \dfrac{f_2 - 2f_1 + 2f_{-1} - f_{-2}}{2h^3} + O(h^2)$
$f^{(3)}(x) = \dfrac{-f_3 + 8f_2 - 13f_1 + 13f_{-1} - 8f_{-2} + f_{-3}}{8h^3} + O(h^4)$
$f^{(4)}(x) = \dfrac{f_2 - 4f_1 + 6f_0 - 4f_{-1} + f_{-2}}{h^4} + O(h^2)$
$f^{(4)}(x) = \dfrac{-f_3 + 12f_2 + 39f_1 + 56f_0 - 39f_{-1} + 12f_{-2} + f_{-3}}{6h^4} + O(h^4)$

표 7.3.3. 미분의 중심차분근사식

구할 수 있다. 셋째, 이 수치미분값은 중심차분근사와 같은 정확도를 갖는다. 만약 x값들이 등간격을 이루면, 식 (7.4.1)은 중심차분근사식 (7.1.6)이 된다.

데이터에 관찰오차가 포함되어 있다면, 데이터를 미분가능한 회귀식으로 적합시킨 다음 그 회귀식을 미분한다. 이 경우에, 종속변수와 독립변수 사이에 정확한 관계가 알려져 있지 않은 경우에는, 낮은 차수의 멱함수를 사용하는 것이 좋은 선택이 될 수 있다.

제7.5절 MATLAB과 Python을 사용한 미분

MATLAB이나 Python을 이용해서 극한값을 구하거나 미분을 할 수 있다. 이 절에서는 몇 가지 예제들을 바탕으로 이러한 기능을 간단히 소개하고 한다.

다음 예제는 MATLAB Symbolic Math Toobox의 함수 limit.m을 사용해서 극한값을 구하는 것이다.

예제 7.5.1 MATLAB함수 limit.m을 사용해서 극한값을 구하기 위해서 다음 MATLAB 프로그램 LimitExample101.m을 실행하자.

```
% ------------------------------------------------------------------------
% Filename LimitExample101.m
% Limit through MATLAB
% Programmed by CBS
% ------------------------------------------------------------------------
clear, close all
syms x y t
L1 = limit((x^2-3*x+2)/(x^2-1),x,1)
L2 = limit((x^2-3*x+2)/(x^2-1),x,inf)
L3 = limit(abs(x)/x,x,0,'left')
L4 = limit(abs(x)/x,x,0,'right')
f1 = (x^2 - y^2)/(x^2+y^2)
L5 = limit(limit(f1,x,2),y,1)
L6 = limit(limit(f1,y,1),x,2)
L7 = limit(limit(f1,x,0),y,0)
L8 = limit(limit(f1,y,0),x,0)
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 LimitExample101.m을 실행하면, 다음 식들이 성립함을 확인할 수 있다.

$$\lim_{x\to 1}\frac{x^2-3x+2}{x^2-1}=-\frac{1}{2} \tag{1}$$

$$\lim_{x\to \infty}\frac{x^2-3x+2}{x^2-1}=1 \tag{2}$$

$$\lim_{x\to 0-}\frac{|x|}{x}=-1, \qquad \lim_{x\to 0+}\frac{|x|}{x}=1 \tag{3}$$

$$\lim_{y\to 1}\lim_{x\to 2}\frac{x^2-y^2}{x^2+y^2}=\frac{3}{5}, \qquad \lim_{x\to 2}\lim_{y\to 1}\frac{x^2-y^2}{x^2+y^2}=\frac{3}{5} \tag{4}$$

$$\lim_{y\to 0}\lim_{x\to 0}\frac{x^2-y^2}{x^2+y^2}=-1, \qquad \lim_{x\to 0}\lim_{y\to 0}\frac{x^2-y^2}{x^2+y^2}=1 \tag{5}$$

■

예제 7.5.2 Python을 사용해서 예제 7.5.1을 다시 다루기 위해서, 다음 Python프로그램 LimitExample101.Py를 실행해 보자.

```
"""
% Filename LimitExample101.m
% Limit through Python
% Programmed by CBS
"""

from sympy import Symbol, limit, oo

x = Symbol('x'); y = Symbol('y'); t = Symbol('t');
L1 = limit((x**2-3*x+2)/(x**2-1),x,1)
L2 = limit((x**2-3*x+2)/(x**2-1),x,oo)    # oo = infty
```

```
L30 = limit(abs(x)/x,x,0, '-')
L31=  limit(abs(x)/x,x,0, dir='-')
L40 = limit(abs(x)/x,x,0, '+')
L41=  limit(abs(x)/x,x,0, dir='+')
print('L1 =',L1, ', L2 =',L2, ', L30 =',L30, ', L31 =',L31, \
      ', L40 =',L40, ', L41 =',L41)
f1 = (x**2 - y**2)/(x**2+y**2)
L5 = limit(limit(f1,x,2),y,1)
L6 = limit(limit(f1,y,1),x,2)
L7 = limit(limit(f1,x,0),y,0)
L8 = limit(limit(f1,y,0),x,0)
print('L5 =',L5, ', L6 =',L6, ', L7 =',L7, ', L8 =',L8)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 7.5.1의 결과와 같다. ■

다음 예제는 MATLAB Symbolic Math Toobox의 함수 diff.m을 사용한 미분에 관한 것으로서, 평균값정리를 구현한 것이다.

예제 7.5.3 평균값정리를 구현하기 위해서, 다음 MATLAB프로그램 MeanValueTheorem101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename MeanValueTheorem101.m
% Mean Value Theorem through MATLAB
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
syms x m p
f = x^3 - 3*x
a = -1, b = 3
fa = subs(f,x,a), fb = subs(f,x,b)
[ m, p ] = solve(m*a+p-fa,m*b+p-fb)
df = diff(f)   % differntiate f
xc = solve(df-m)
c = double(xc)
yc = subs(f,x,xc)
ycd = double(yc)
% Ploting
xx = -2:0.02:4;
ff = xx.^3 - 3*xx;
tanline = ycd(2) + double(m)*(xx-c(2));
secline = fa + (fb-fa)/(b-a)*(xx-a);
hold on
plot(xx,ff,'r',xx,tanline,'k--',xx,secline,'b:','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
legend('y = f(x)','Tangent Line','Secant Line','location','NW')
plot([a,b],[fa,fb],'ko','LineWidth',3)
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
axis([-2 4 -20 60 ])
hold off
saveas(gcf,'MeanValueTheorem101.jpg')
```

```
% End of Program
% --------------------------------------------------------------
```

이 MATLAB프로그램 MeanValueTheorem101.m을 실행하면, 구간 $[a,b]=[-1,3]$에서 함수 $f(x)=x^3-3x$의 할선(secant line)과 기울기가 같은 접선(tangent line)을 구한다. 먼저, 다음 함수값들을 출력한다.

$$f(a)=2, \quad f(b)=18 \tag{1}$$

두 점들 $(a,f(a))$와 $(b,f(b))$를 지나는 할선은 다음과 같다.

$$y=mx+p=4x+6 \tag{2}$$

또한, 함수 $f(x)=x^3-3x$의 도함수는 다음과 같음을 확인할 수 있다.

$$f'(x)=3x^2-3 \tag{3}$$

기울기가 $m=4$가 되는 x값들은 다음과 같다.

$$c_1=-\frac{\sqrt{21}}{3}, \quad c_2=\frac{\sqrt{21}}{3} \tag{4}$$

또한, 이에 해당하는 y값들은 각각 다음과 같다.

$$y_1=\frac{2\sqrt{21}}{9}, \quad y_2=-\frac{2\sqrt{21}}{9} \tag{5}$$

즉, 기울기가 $m=4$인 접선들은 다음과 같다.

$$y+\frac{2\sqrt{21}}{9}=4\left[x-\frac{\sqrt{21}}{3}\right] \tag{6}$$

$$y-\frac{2\sqrt{21}}{9}=4\left[x+\frac{\sqrt{21}}{3}\right] \tag{7}$$

함수 $f(x)=x^3-3x$, 할선 (2) 그리고 접선 (7)를 그린 것이 그림 7.5.1이다. ■

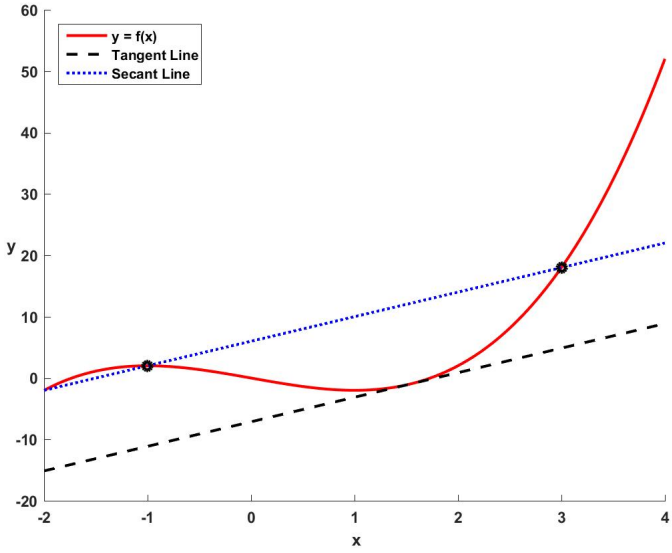


그림 7.5.1. 평균값정리

예제 7.5.4 Python을 사용해서 예제 7.5.3을 다시 다루기 위해서, 다음 Python프로그램 MeanValueTheorem101.Py를 실행해 보자.

```
"""
% Filename MeanValueTheorem101.Py
% Mean Value Theorem through Python
% Programmed by CBS
"""

import sympy as sy
import numpy as np
import matplotlib.pyplot as plt

x = sy.Symbol('x'); m = sy.Symbol('m'); p = sy.Symbol('p');
f = x**3 - 3*x
a = -1; b = 3
fa = f.subs(x,a); fb = f.subs(x,b)
systemEq = sy.Matrix(( (a,1,fa),(b,1,fb)))
mp = sy.solve_linear_system(systemEq, m,p)
print(mp)
mVal = 4; pVal = 6
df = sy.diff(f)   # differntiate f
print(df)
xc = sy.solve((df-mVal),x)
print(xc)
yc0 = f.subs(x,xc[0])
yc1 = f.subs(x,xc[1])
print(yc0,yc1)

## plotting
fig = plt.figure()
xx = np.linspace(-2,4,201)
ff = xx**3 - 3*xx;
tanline = yc1 + mVal*(xx-xc[1])
secline = fa + (fb-fa)/(b-a)*(xx-a)
plt.plot(xx,ff,'r',lw=2,label='y = f(x)')
```

```
plt.plot(xx,tanline,'k--',lw=2,label='Tangent Line')
plt.plot(xx,secline,'b:',lw=2,label='Secant Line')
plt.legend(loc='upper center', shadow=True)
plt.axis([-2, 4, -20, 60 ])
plt.plot([a,b],[fa,fb],'ko',lw=3)
plt.xlabel('x'), plt.ylabel('y')
plt.savefig('MeanValueTheorem101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.5.3의 결과와 같다. ■

수치미분을 구하기 위해서는 수열의 차분을 구할 필요가 있다. 다음은 차분을 구하는 예제이다.

예제 7.5.5 MATLAB함수 diff.m을 사용해서 주어진 수열의 차분을 구할 수 있다. 이 함수의 사용법은 다음과 같다.

```
>> xkk = diff(x,k)
```

이 명령문을 실행하면, 변수 x의 k차 차분이 변수 xkk에 저장된다. 차분을 구하는 예로서 다음 MATLAB프로그램 SequenceDifference101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
% Filename SequenceDifference101.m
% Differences of a Sequence
% Programmed by CBS
% ------------------------------------------------------------------------
clear, close all
nn3 = ((0:1:10)').^3;
OutD = NaN.*ones(11,4);
OutD(:,1) = nn3;
OutD(1:10,2) = diff(nn3,1);
OutD(1:9,3) = diff(nn3,2);
OutD(1:8,4) = diff(nn3,3)
save('SequenceDifference101.txt','OutD','-ascii')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 SequenceDifference101.m을 실행한 결과는 다음과 같다. 이 결과물에서 MATLAB함수 diff.m의 사용법을 확인할 수 있다.

```
     0     1     6     6
     1     7    12     6
     8    19    18     6
    27    37    24     6
    64    61    30     6
```

125	91	36	6
216	127	42	6
343	169	48	6
512	217	54	NaN
729	271	NaN	NaN
1000	NaN	NaN	NaN

■

예제 7.5.6 Python을 사용해서 예제 7.5.5를 다시 다루기 위해서, 다음 Python프로그램 SequenceDifference101.Py를 실행해 보자.

```
"""
% Filename SequenceDifference101.m
% Differences of a Sequence
% Programmed by CBS
"""

import numpy as np

nn3 = (np.arange(0,11))**3;
OutD = np.empty((11,4));
OutD[:,:] = np.nan
OutD[:,0] = nn3;
OutD[0:10,1] = np.diff(nn3,1);
OutD[0:9,2] = np.diff(nn3,2);
OutD[0:8,3] = np.diff(nn3,3)
print('OutD =', OutD)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 7.5.5의 결과와 같다. ■

제7.6절 심볼릭계산

MATLAB의 Symbolic Math Toolbox나 Python의 SymPy패키지를 사용해서 수치계산이 아닌 심볼릭계산을 할 수 있다. 앞에서도 간단한 심볼릭계산을 하였으나, 이 절에서는 예제들을 통해서 MATLAB과 Python을 사용한 심볼릭계산에 대해 좀 더 살펴보자.

예제 7.6.1 심볼릭변수를 지정하거나 식을 간단히 표기하는 예를 살펴보기 위해서, 다음 MATLAB프로그램 SymbolicComputation101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename SymbolicComputation101.m
% Symbolic Variables and Functions
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary SymbolicComputation101.txt
syms a b
y = 2/3 + 3/9 + 7/6
y1 = sym(y)
g = a^2 + 2*a*b + b^2
g1 = simplify(g)
g2 = factor(g)
pretty(g)
diary off
% End oh Program
% ------------------------------------------------------------------
```

MATLAB명령어 syms에는 심볼릭변수들을 지정한다. 즉, MATLAB명령문 'syms a b'는 변수들 a와 b가 심볼릭변수들임을 나타낸다. MATLAB명령어 sym은 숫자변수를 심볼릭변수로 변환한다. 즉, MATLAB명령문 'y1 = sym(y)'는 변수 y1에 '13/6'를 할당한다. MATLAB명령문 'g = a^2+2*a*b+b^2'은 새로운 변수 g를 생성한다. MATLAB명령어 'simplify'는 변수를 간단하게 만든다. 즉, MATLAB명령문 'g1 = simplify(g)'는 변수 g1에 $(a+b)^2$를 할당한다. MATLAB명령어 'factor'는 인수분해를 한다. 즉, MATLAB명령문 'g2 = factor(g)'는 변수 g2에 인수들 $[a+b, a+b]$를 할당한다. MATLAB명령어 'pretty'는 식을 좀 더 보기 쉽게 표기하도록 한다. 즉, MATLAB명령문 'pretty(g)'는 변수 g를 보기 좋게 $a^2+2ab+b^2$으로 표기한다. ■

예제 7.6.2 Python을 사용해서 예제 7.6.1을 다시 다루기 위해서, 다음 Python프로그램 SymbolicComputation101.Py를 실행해 보자.

```
"""
% Filename SymbolicComputation101.Py
% Symbolic Variables and Functions
% Programmed by CBS
"""

from sympy import Symbol, simplify, factor

a = Symbol('a');  b = Symbol('b');
y = 2/3 + 3/9 + 7/6
y1 = Symbol('y')
print('y = ', y, ', y1 =', y1)
g = a**2 + 2*a*b + b**2
g1 = simplify(g)
g2 = factor(g)
print('g =', g, ', g1 =', g1, ', g2 =', g2)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 7.6.1의 결과와 같다. ■

예제 7.6.3 심볼릭계산을 사용해서 방정식을 푸는 예를 살펴보기 위해서, 다음 MATLAB 프로그램 SymbolicComputation102.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename Symbolic Computation102.m
% Function solve
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary SymbolicComputation102.txt
syms a b c x
sol11 = solve(a*x^2+b*x+c)
sol12 = solve('a*x^2+b*x+c = 0')
y = a*x^2+b*x+c;
sol13 = solve(y,x)
sol14 = solve(y,a)
sol15 = solve(y,b)
sol16 = solve(y,c)
z = x^4 + x^2 +1;
sol21 = solve(z)
sol22 = vpa(sol21)
sol23 = vpa(sol21,10)
sol24 = double(sol21)
w = x^4 + 2*x^3 + 3*x^2 + 4*x +5;
w1 = sym2poly(w)
w2 = poly2sym(w1)
diary off
% End oh Program
% ------------------------------------------------------------------
```

MATLAB함수 solve.m을 사용해서 비선형방정식을 풀 수 있다. 다음 MATLAB명령문들은 2차방정식 $ax^2 + bx + c = 0$를 변수 x에 대해서 푸는 것이다.

```
sol11 = solve(a*x^2+b*x+c)
sol12 = solve('a*x^2+b*x+c = 0')
y = a*x^2+b*x+c;  sol13 = solve(y,x)
```

다음 MATLAB명령문들은 방정식 $ax^2 + bx + c = 0$를 각각 변수 a, 변수 b, 그리고 변수 c에 대해서 푸는 것이다.

```
sol14 = solve(y,a)
sol15 = solve(y,b)
sol16 = solve(y,c)
```

다음 MATLAB명령문들은 방정식 $x^4 + x^2 + 1 = 0$을 수 x에 대해서 푸는 것이다.

```
z = x^4 + x^2 +1;  sol21 = solve(z)
sol22 = vpa(sol21)
```

```
sol23 = vpa(sol21,10)
sol24 = double(sol23)
```

MATLAB명령문 'sol21 = solve(z)'에 의한 해들은 다음과 같다.

```
sol21 =
 - (3^(1/2)*1i)/2 - 1/2
   1/2 - (3^(1/2)*1i)/2
   (3^(1/2)*1i)/2 - 1/2
   (3^(1/2)*1i)/2 + 1/2
```

이 심볼릭변수가 내포하는 수를 부동소수형(variable-precision floating-point arithmetic: VPA)으로 나타내기 위해서는 MATLAB함수 vpa.m을 사용한다. MATLAB명령문 'sol22 = vpa(sol21)'에 의한 결과물은 다음과 같다.

```
sol22 =
 - 0.5 - 0.866025403784438646763723170752941i
   0.5 - 0.866025403784438646763723170752941i
 - 0.5 + 0.866025403784438646763723170752941i
   0.5 + 0.866025403784438646763723170752941i
```

유효숫자를 명시하기 위해서는 MATLAB명령문 'vpa(x,d)'를 사용한다. 예를 들어, MATLAB명령문 'sol23 = vpa(sol21,10)'에 의한 결과물은 다음과 같다.

```
sol23 =
 - 0.5 - 0.8660254038i
   0.5 - 0.8660254038i
 - 0.5 + 0.8660254038i
   0.5 + 0.8660254038i
```

MATLAB함수 double.m을 사용해도 심볼릭변수가 내포하는 수를 부동소수형으로 나타낼 수 있다. MATLAB명령문 'sol24 = double(sol21)'에 의한 결과물은 다음과 같다.

```
sol24 =
  -0.5000 - 0.8660i
   0.5000 - 0.8660i
  -0.5000 + 0.8660i
   0.5000 + 0.8660i
```

MATLAB함수 sym2poly.m은 심볼릭변수로 표현된 멱함수(polynomial)의 계수들을 포함하는 벡터를 생성한다. 또한, MATLAB함수 poly2sym.m은 멱함수의 계수들을 포함하는 벡터로부터 심볼릭변수를 생성한다. 다음 MATLAB명령문들을 실행해보자.

```
w = x^4 + 2*x^3 + 3*x^2 + 4*x +5;
w1 = sym2poly(w)
w2 = poly2sym(w1)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
w1 =
     1     2     3     4     5
w2 =
     x^4 + 2*x^3 + 3*x^2 + 4*x + 5
```

■

예제 7.6.4 Python을 사용해서 예제 7.6.3을 다시 다루기 위해서, 다음 Python프로그램 SymbolicComputation102.Py를 실행해 보자.

```
"""
% Filename Symbolic Computation102.Py
% Function solve
% Programmed by CBS
"""

import numpy as np
from sympy import solve
from sympy.abc import a, b, c, x
# a = Symbol('a');  b = Symbol('b'); c = Symbol('c');  x = Symbol('x');

# Case 1
sol1 = solve(a*x**2+b*x+c, x)
print('sol1 =', sol1[0], ',  sol2 =', sol1[1])
y = a*x**2+b*x+c;
sol13 = solve(y,x)
print('sol13 =', sol13[0], ',  sol2 =', sol13[1])
sol14 = solve(y,a);  print('sol14 =', sol14)
sol15 = solve(y,b);  print('sol15 =', sol15)
sol16 = solve(y,c);  print('sol16 =', sol16)
# Case 2
z = x**4 + x**2 +1;
sol21 = solve(z)
print('sol211 =', sol21[0], ',  sol212 =', sol21[1])
print('sol213 =', sol21[2], ',  sol214 =', sol21[3])
sol221 = sol21[0].evalf();  print('sol221 =', sol221)
sol222 = sol21[1].evalf();  print('sol222 =', sol221)
sol223 = sol21[2].evalf();  print('sol223 =', sol221)
sol224 = sol21[3].evalf();  print('sol224 =', sol221)
# Case 3
w = np.poly1d([5.4,4.4,3.3,2.2,1.1]); print(w)
w1 = np.poly1d(w); print(w1)
w2 = w.c;  print(w2)              # Coefficients
worder = w.order; print(worder)
w[3] # coefficient of order 3
wval = w(1.1); print(wval)  # value at 1.1

# End of program
```

이 Python프로그램을 수행한 결과는 예제 7.6.3의 결과와 같다. ■

예제 7.6.5 MATLAB에서 문자변수를 다루는 예를 살펴보기 위해서, 다음 MATLAB 프로그램 SymbolicComputation103.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename Symbolic Computation103.m
% Vector, Matrix, and Character
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary SymbolicComputation103.txt
syms a b c x
A = [ cos(x) -sin(x) ; sin(x) cos(x) ]
B = inv(A)
B1 = simplify(B)
c1= 'CBS family'
c11 = lower(c1)
c12 = upper(c11)
c2 = num2str(pi)
c21 = str2num(c2)
c3 = char(117:126)
c31 = abs(c3)
c32 = double(c3)
c33 = uint8(c3)
c4 = 'This is an oasis.'
c41 = findstr(c4,'is')
diary off
% End oh Program
% ------------------------------------------------------------------
```

다음 MATLAB명령문들을 실행해보자.

```
syms a b c x
A = [ cos(x) -sin(x) ; sin(x) cos(x) ]
B = inv(A)
B1 = simplify(B)
```

이 MATLAB명령문들을 실행하면, 심볼릭행렬을 생성하고 이 심볼릭행렬의 역행렬을 생성한다. 이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
A =
  [ cos(x), -sin(x)]
  [ sin(x),  cos(x)]

B =
  [  cos(x)/(cos(x)^2 + sin(x)^2), sin(x)/(cos(x)^2 + sin(x)^2)]
  [ -sin(x)/(cos(x)^2 + sin(x)^2), cos(x)/(cos(x)^2 + sin(x)^2)]

B1 =
  [  cos(x), sin(x)]
  [ -sin(x), cos(x)]
```

다음 MATLAB명령문들을 실행해보자.

```
c1= 'CBS family'
c11 = lower(c1)
c12 = upper(c11)
```

이 MATLAB명령문들을 실행하면, 문자변수를 정의하고, 문자들을 모두 소문자로 변환하거나, 모두 대문자로 변환한다. 이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
c1 =
CBS family

c11 =
cbs family

c12 =
CBS FAMILY
```

다음 MATLAB명령문들을 실행해보자.

```
c2 = num2str(pi)
c21 = str2num(c2)
```

이 MATLAB명령문들을 실행하면, 숫자변수 π를 문자변수 c2로 변환하거나, 문자변수 c2를 숫자변수 c21로 변환한다. 이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
c2 =
3.1416

c21 =
    3.1416
```

다음 MATLAB명령문들을 실행해보자.

```
c3 = char(117:126)
c31 = abs(c3)
c32 = double(c3)
c33 = uint8(c3)
```

이 MATLAB명령문들을 실행하면, 정수에 해당하는 ASCII로 변환하거나 ASCII에 해당하는 정수로 변환한다. 이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
c3 =
uvwxyz{|}~

c31 =
   117   118   119   120   121   122   123   124   125   126

c32 =
   117   118   119   120   121   122   123   124   125   126
```

```
c33 =
  117   118   119   120   121   122   123   124   125   126
```

다음 MATLAB명령문들을 실행해보자.

```
c4 = 'This is an oasis.'
c41 = findstr(c4,'is')
```

이 MATLAB명령문들을 실행하면, 문자변수 c4에서 문자 is에 해당하는 위치를 변수 C41에 할당한다. 이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
c4 =
This is an oasis.

c41 =
     3     6    15
```

■

예제 7.6.6 Python을 사용해서 예제 7.6.5을 다시 다루기 위해서, 다음 Python프로그램 SymbolicComputation103.Py를 실행해 보자.

```
"""
% Filename Symbolic Computation103.Py
% Vector, Matrix, and Character
% Programmed by CBS
"""

import numpy as np
import sympy as sy
from sympy.abc import x

A = sy.Matrix([[ sy.cos(x), -sy.sin(x) ], [ sy.sin(x), sy.cos(x) ]]);
print(A)
B = A**(-1);          print(B)
B1 = sy.simplify(B);    print(B1)
c1= 'CBS family';     print(c1)
c11 = c1.lower();     print(c11)
c12 = c11.upper();    print(c12)
c2 = str(np.pi);      print(c2)
c21 = float(c2);      print(c21)
c3 = [chr(i) for i in range(117,127)];  print(c3)
c31 = [ord(i) for i in c3];  print(c31)
c4 = 'This is an oasis.'
c40 = 'is'
c41 = c4.find(c40);   print(c41)

# End oh Program
```

이 Python프로그램을 수행한 결과는 예제 7.6.5의 결과와 같다. ■

예제 7.6.7 MATLAB에서 논리변수를 다루는 예를 살펴보기 위해서, 다음 MATLAB 프로그램 SymbolicComputation104.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename Symbolic Computation104.m
% Logic functions
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary SymbolicComputation104.txt
A = [ exp(1) inf -inf NaN 0 ]
A1 = all(A)
A2 = any(A)
A3 = find(A)
A4 = isinf(A)
A5 = isfinite(A)
B = 1:9
B1 = isempty(B)
B2 = isprime(B)
C1 = 1:5, C2 = 13:-2:3
C11 = ismember(C1,C2)
C12 = ismember(C2,C1)
diary off
% End oh Program
% ------------------------------------------------------------------
```

다음 MATLAB명령문들을 실행해보자.

```
A = [ exp(1) inf -inf NaN 0 ]
A1 = all(A)
A2 = any(A)
A3 = find(A)
A4 = isinf(A)
A5 = isfinite(A)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
A =
    2.7183       Inf      -Inf       NaN         0

A1 =
     0

A2 =
     1

A3 =
     1     2     3     4

A4 =
     0     1     1     0     0
```

```
A5 =
     1     0     0     0     1
```

이 결과물에서 알 수 있듯이, MATLAB함수 all은 모든 원소들이 참(true, nonzero)인 경우에 결과변수에 1을 할당하고 그렇지 않으면 0을 할당한다. MATLAB함수 any는 단 하나의 원소라도 참(true, nonzero)인 경우에 결과변수에 1을 할당하고 그렇지 않으면 0을 할당한다. MATLAB함수 find는 0이 아닌 원소의 위치를 결과변수에 할당한다. MATLAB함수 isinf는 절대값이 무한대인 원소의 위치에 1을 그렇지 않으면 0을 결과변수에 할당한다. MATLAB함수 isfinite는 유한값인 원소의 위치에 1을 그렇지 않으면 0을 결과변수에 할당한다.

다음 MATLAB명령문들을 실행해보자.

```
B = 1:9
B1 = isempty(B)
B2 = isprime(B)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
B =
     1     2     3     4     5     6     7     8     9

B1 =
     0

B2 =
     0     1     1     0     1     0     1     0     0
```

이 결과물에서 알 수 있듯이, MATLAB명령문 'isempty(B)'는 행렬 B가 빈행렬(empty matrix)이면, 결과변수에 1을 할당하고 그렇지 않으면 0을 할당한다.

다음 MATLAB명령문들을 실행해보자.

```
C1 = 1:5, C2 = 13:-2:3
C11 = ismember(C1,C2)
C12 = ismember(C2,C1)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
C1 =
     1     2     3     4     5

C2 =
    13    11     9     7     5     3

C11 =
     0     0     1     0     1
```

```
C12 =
     0     0     0     0     1     1
```

이 결과물에서 알 수 있듯이, MATLAB명령문 'ismember(C1,C2)'는 행렬 C1의 각 원소에 대해서 이 원소가 행렬 C2의 원소이면 결과변수의 같은 자리에 1을 할당하고, 그렇지 않으면 0을 할당한다. ■

예제 7.6.8 Python을 사용해서 예제 7.6.7을 다시 다루기 위해서, 다음 Python프로그램 SymbolicComputation104.Py를 실행해 보자.

```
"""
% Filename Symbolic Computation104.Py
% Logic functions
% Programmed by CBS
"""

import numpy as np

A = [ np.exp(1), np.inf, -np.inf, np.NaN, 0 ];   print(A)
A1 = all(A);              print(A1)
A2 = any(A);              print(A2)
A3 = np.nonzero(A)[0];    print(A3+1)
A3X = np.nonzero(A);      print(A3X)
A4 = np.isinf(A);         print(A4)
A5 = np.isfinite(A);      print(A5)

B = np.arange(9)+1;       print(B)
if not B.size:
   print('B is empty')
else:
   print('B is not empty')
B1 = (not B.size);        print(B1)
# Python program to display all the prime numbers
PrimeN = [];
LowNo = 2;  UpNo = 9;  # Prime numbers in [ LowNo, UpNo ].
print("Prime numbbers between",LowNo,"and",UpNo,"are:")
for numb in range(LowNo,UpNo + 1):
    if numb > 1:
       for ii in range(2,numb):
           if (numb % ii) == 0:
              break
       else:
           PrimeN = np.append(PrimeN,numb);
B2 = PrimeN;
print('Prime numbers = ', B2)
# End of Prime number program

C1 = np.arange(1,6,1);           print('C1 =', C1)
C2 = np.arange(13,2,-2);         print('C2 =', C2)
def ismember(FirstNo, SecondNo):
    Sind = {}
    for ii, elementt in enumerate(SecondNo):
        if elementt not in Sind:
```

```
            Sind[elementt] = ii+1
    return [Sind.get(itm, None) for itm in FirstNo]
           # None can be replaced by any other "not in SecondNo" value
C11 = ismember(C1,C2);   print('C11', C11)
C12 = ismember(C2,C1);   print('C12', C12)


# End of program
```

이 Python프로그램을 수행한 결과는 예제 7.6.7의 결과와 같다. ■

예제 7.6.9 MATLAB에서 논리변수를 다루는 예를 살펴보기 위해서, 다음 MATLAB 프로그램 SymbolicComputation105.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename Symbolic Computation105.m
% Polynmials
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary SymbolicComputation105.txt
syms a b c d e x
f = a*x^2 + b*x + c
g = d*x + e
fg11 = f*g
fg12 = expand(fg11)
fg13 = simplify(fg12)
fg14 = factor(fg13)
h = 4*x^3 +3*x^2 + 2*x + 1
l = x^2 - 3*x +2
h1 = sym2poly(h)
l1 = sym2poly(l)
h2 = poly2sym(h1)
h3 = roots(h)
h4 = roots(h1)
h5 = polyval(h1,-1)
[ coef pol quo ] = residue(h1,l1)
[ h6 l6 ] = residue(coef,pol,quo)
convhl = conv(h1,l1)
hl = expand(h*l)
[ quohl rhl ] = deconv(h1,l1)
dum1 = (conv(quohl,l1)+rhl) - h1
v = [5 4 3 2 1 ]
v1 = poly(v)
v2 = poly2sym(v1)
M = magic(4)
M1 = poly(M)
M2 = poly2sym(M1)
M3 = polyval(M1,1)
M4 = polyvalm(M1,eye(3))
diary off
% End oh Program
% ------------------------------------------------------------------
```

다음 MATLAB명령문들을 실행해보자.

```
f = a*x^2 + b*x + c
g = d*x + e
fg11 = f*g
fg12 = expand(fg11)
fg13 = simplify(fg12)
fg14 = factor(fg12)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
f =
a*x^2 + b*x + c

g =
e + d*x

fg11 =
(e + d*x)*(a*x^2 + b*x + c)

fg12 =
c*e + b*e*x + c*d*x + a*d*x^3 + a*e*x^2 + b*d*x^2

fg13 =
(e + d*x)*(a*x^2 + b*x + c)

fg14 =
[ e + d*x, a*x^2 + b*x + c]
```

이 결과물에서 알 수 있듯이, MATLAB함수 expand.m은 함수를 전개한다. MATLAB함수 simplify.m은 함수를 간단히 표기하고, MATLAB함수 factor.m은 인수분해를 한다.

다음 MATLAB명령문들을 실행해보자.

```
h = 4*x^3 +3*x^2 + 2*x + 1
l = x^2 - 3*x +2
h1 = sym2poly(h)
l1 = sym2poly(l)
h2 = poly2sym(h1)
h3 = roots(h)
h4 = roots(h1)
h5 = polyval(h1,-1)
[ coef pol quo ] = residue(h1,l1)
[ h6 l6 ] = residue(coef,pol,quo)
convhl = conv(h1,l1)
hl = expand(h*l)
[ quohl rhl ] = deconv(h1,l1)
dum1 = (conv(quohl,l1)+rhl) - h1
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
h =
4*x^3 + 3*x^2 + 2*x + 1

l =
x^2 - 3*x + 2

h1 =
     4     3     2     1

l1 =
     1    -3     2

h2 =
4*x^3 + 3*x^2 + 2*x + 1

h3 =
Empty sym: 0-by-1

h4 =
  -0.6058 + 0.0000i
  -0.0721 + 0.6383i
  -0.0721 - 0.6383i

h5 =
    -2

coef =
    49
   -10

pol =
     2
     1

quo =
     4    15

h6 =
     4     3     2     1

l6 =
     1    -3     2

convhl =
     4    -9     1     1     1     2
```

```
hl =
4*x^5 - 9*x^4 + x^3 + x^2 + x + 2

quohl =
     4    15

rhl =
     0     0    39   -29

dum1 =
     0     0     0     0
```

앞에서 언급했듯이, MATLAB명령문 'h1 = sym2poly(h)'는 심볼릭변수 h로 표현된 멱함수의 계수들을 포함하는 벡터 h1을 생성한다. 또한, MATLAB명령문 'h2 = poly2sym(h1)'은 멱함수의 계수들을 포함하는 벡터 h1으로부터 심볼릭변수 h2를 생성한다. MATLAB함수 roots.m는 멱함수의 근을 생성한다. 유의할 점은 이 MATLAB함수의 입력변수는 심볼릭으로 표현된 h1이 아니라 벡터로 표현된 h2라는 것이다. MATLAB명령문 'h5 = polyval(h1,-1)'은 입력변수값 -1에서 멱함수 h1을 계산한다. MATLAB명령문 '[coef pol quo] = residue(h1,l1)'은 멱함수 h1을 멱함수 l1으로 나눈 계수(coefficient), 극값(pole) 그리고 몫(quotient)을 생성한다. 즉, 다음과 같은 부분분수분해를 한다.

$$\frac{\mathrm{h1}(x)}{\mathrm{l1}(x)} = \sum_{k=1}^{n} \frac{\mathrm{coef}_k}{x - \mathrm{pol}_k} + \mathrm{quo}(x) \tag{1}$$

여기서 $\mathrm{quo}(x)$는 멱함수이다. 또한, MATLAB명령문 '[h6,l6] = residue(coef,pol,quo)'에서는 입력변수들이 계수(coefficient), 극값(pole) 그리고 몫(quotient)이고 출력변수들이 h6와 l6이다.

다음 MATLAB명령문들을 실행해보자.

```
convhl = conv(h1,l1)
hl = expand(h*l)
[ quohl rhl ] = deconv(h1,l1)
dum1 = (conv(quohl,l1)+rhl) - h1
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
convhl =
     4    -9     1     1     1     2

hl =
4*x^5 - 9*x^4 + x^3 + x^2 + x + 2
```

```
quohl =
     4    15

rhl =
     0     0    39   -29

dum1 =
     0     0     0     0
```

MATLAB명령문 'convhl = conv(h1,l1)'은 두 멱함수들 h1과 l1의 콜레곱(convolution) convh1을 생성한다. 이 콜레곱 convh1는 벡터 h1에 해당하는 멱함수와 벡터 l1에 해당하는 멱함수의 콜레곱의 계수들로 이루어진 벡터이다. MATLAB명령문 'hl = expand(h*l)'를 실행하면, 이 결과를 확인할 수 있다. MATLAB명령문 '[quohl rhl] = deconv(h1,l1)'은 멱함수 h1을 멱함수 l1으로 나눈 몫 quohl과 나머지 rh1을 생성한다. 이때 입력변수들 h1과 l1 그리고 출력변수들 quohl과 rhl 모두 벡터들이다. MATLAB명령문 'dum1 = (conv(quohl,l1)+rhl)-h1'를 실행하면, 이 결과를 확인할 수 있다.

다음 MATLAB명령문들을 실행해보자.

```
v = [5 4 3 2 1 ]
v1 = poly(v)
v2 = poly2sym(v1)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
v =
     5     4     3     2     1

v1 =
     1   -15    85  -225   274  -120

v2 =
x^5 - 15*x^4 + 85*x^3 - 225*x^2 + 274*x - 120
```

만약 입력변수 v가 벡터이면, MATLAB명령문 'v1 = poly(v)'는 벡터 v의 원소들을 근으로 하는 멱함수의 계수들의 벡터 v1을 출력한다. MATLAB명령문 'v2 = poly2sym(v1)'은 계수벡터 v에 해당하는 심볼릭멱함수 v2를 생성한다.

다음 MATLAB명령문들을 실행해보자.

```
M = magic(4)
M1 = poly(M)
M2 = poly2sym(M1)
M3 = polyval(M1,1)
M4 = polyvalm(M1,eye(3))
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
M =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1

M1 =
   1.0e+03 *
    0.0010   -0.0340   -0.0800    2.7200   -0.0000

M2 =
x^4 - 34*x^3 - 80*x^2 + 2720*x - 3464245048593955/6189700196426901374495621l2

M3 =
   2.6070e+03

M4 =
   1.0e+03 *
    2.6070         0         0
         0    2.6070         0
         0         0    2.6070
```

만약 입력변수 M가 정방행렬이면, MATLAB명령문 'M1 = poly(M)'은 행렬 M의 특성멱함수 (characteristic polynomial) $\det(xI - M)$의 계수들로 이루어진 벡터를 생성한다. MATLAB 명령문 'M2 = poly2sym(M1)'은 특성멱함수의 계수벡터 M1에 해당하는 심볼릭멱함수 M2를 생성한다. MATLAB명령문 'M3 = polyval(M1,1)'은 특성멱함수 $\det(xI - M)$에서 입력변수 x 대신 값 1을 대입한 함수값 $\det(I - M)$을 생성한다. MATLAB명령문 'M4 = polyvalm(M1,eye(3))'은 특성멱함수 $\det(xI - M)$에서 입력변수 x 대신 행렬 I_3를 대입한 행렬 M4를 생성한다. 이 경우에 특성멱함수 $\det(xI - M)$을 멱함수로 나타낸 다음 입력변수 x 대신 행렬 I_3를 대입함에 유의하라. ■

예제 7.6.10 Python을 사용해서 예제 7.6.9을 다시 다루기 위해서, 다음 Python프로그램 SymbolicComputation105.Py를 실행해 보자.

```
"""
% Filename Symbolic Computation105.m
% Polynmials
% Programmed by CBS
"""

```

```
import numpy as np
import sympy as sy
from sympy.abc import a, b, c, d, e, x

f = a*x**2 + b*x + c
g = d*x + e
fg11 = f*g;                          print(fg11)
fg12 = sy.expand(fg11);              print(fg12)
fg13 = sy.simplify(fg12);            print(fg13)
fg14 = sy.factor(fg13);              print(fg14)

h = sy.Poly(4*x**3 +3*x**2 + 2*x + 1, x);  print(h)
l = sy.Poly(x**2 - 3*x +2,x);              print(l)
h1 = h.as_dict();                          print(h1)
l1 = h.as_dict();                          print(l1)
h2 = sy.Poly(h1,x);                        print(h2)
h3 = sy.solve(h);                          print(h3)
# h4 = sy.solve(h1);
h5 = h.subs({'x': np.pi});                 print(h5)

# Quotient and Remainder
q101, r101 = sy.div(h,l, x)
print('quotient =', q101)
print('residual =', r101)
h101 = (q101*l+r101).expand()
print(h101)
hs = np.array([4.0, 3.0, 2.0, 1.0])
ls = np.array([1.0, -3.0, 2.0])
q102, r102 = np.polydiv(hs, ls)
print('quotient =', q102)
print('residual =', r102)

# Convolution
convhl = np.convolve(hs,ls);          print(convhl)
hl = sy.Poly(convhl,x);                  print(hl)
from scipy import signal
hl101 = signal.convolve(hs,ls);       print(hl101)
quohl, rhl = signal.deconvolve(hl101,ls);
print('quohl =', quohl); print('rhl =', rhl)

# Polynomials
v = np.array([5, 4, 3, 2, 1 ]);     print(v)
v1 = np.poly1d(v);                  print(v1)

# End of program
```

이 Python프로그램을 수행한 결과는 예제 7.6.9의 결과의 일부분과 같다. ■

예제 7.6.11 MATLAB에서 행렬의 함수들을 다루는 예를 살펴보기 위해서, 다음 MATLAB프로그램 MatrixComputation101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename MatrixComputation101.m
% Matrix functions
% Programmed by CBS
% ------------------------------------------------------------------
```

```
clear, close all
diary MatrixComputation101.txt
A = [ 1 2 3 ; 4 5 6 ; 7 8 9 ]
A1 = sqrt(A)
dum1 = A1.*A1 - A
A2 = sqrtm(A)
dum2 = A2*A2 - A
A3 = cond(A)
B = [ 0 1 ; 2 3 ]
B1 = exp(B)
B2 = expm(B)
[B21 B22 ] = eig(B)
dum3 = B21*diag(exp(diag(B22)))/B21 - B2
B3 = eye(2)/B - inv(B)
C = magic(4)
C1 = rank(C)
C2 = det(C)
C3 = pinv(C)  % psuedo-inverse
dum4 = C3*C*C3 - C3
dum5 = C*C3*C - C
diary off
% End oh Program
% ------------------------------------------------------------------
```

다음 MATLAB명령문들을 실행해보자.

```
A = [ 1 2 3 ; 4 5 6 ; 7 8 9 ]
A1 = sqrt(A)
dum1 = A1.*A1 - A
A2 = sqrtm(A)
dum2 = A2*A2 - A
A3 = cond(A)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
A =
     1     2     3
     4     5     6
     7     8     9

A1 =
    1.0000    1.4142    1.7321
    2.0000    2.2361    2.4495
    2.6458    2.8284    3.0000

dum1 =
   1.0e-14 *
         0    0.0444   -0.0444
         0    0.0888   -0.0888
    0.0888    0.1776         0

A2 =
```

```
   0.4498 + 0.7623i   0.5526 + 0.2068i   0.6555 - 0.3487i
   1.0185 + 0.0842i   1.2515 + 0.0228i   1.4844 - 0.0385i
   1.5873 - 0.5940i   1.9503 - 0.1611i   2.3134 + 0.2717i

dum2 =
   1.0e-14 *
   0.1332 + 0.0555i   0.0000 + 0.0444i   0.0444 + 0.0444i
  -0.1776 + 0.0701i  -0.6217 + 0.0222i  -0.4441 - 0.0167i
  -0.4441 + 0.0444i  -0.6217 + 0.0000i  -0.7105 - 0.0777i

A3 =
   1.1439e+17
```

이 결과물에서 알 수 있듯이 MATLAB명령문 'A1 = sqrt(A)'은 입력행렬 A의 각 원소의 제곱근을 포함하는 행렬 A1을 생성한다. MATLAB명령문 'dum1 = A1.*A1 - A'의 결과에서 이를 확인할 수 있다. MATLAB명령문 'A2 = sqrtm(A)'은 식 A2*A2 = A를 만족하는 행렬 A2를 생성한다. 행렬 A2의 각 특성값은 실수부분이 비음이다. 이러한 행렬 A2는 일의적(uniquely)으로 정해진다. MATLAB명령문 'dum2 = A2*A2 - A'의 결과에서 이를 확인할 수 있다. MATLAB명령문 'A3 = cond(A)'은 입력행렬 A의 조건수 A3를 생성한다. 만약 이 조건수가 1에 가까우면, 행렬 A는 역행렬을 구하기에 좋은 조건(well-condition)을 갖는다.

다음 MATLAB명령문들을 실행해보자.

```
B = [ 0 1 ; 2 3 ]
B1 = exp(B)
B2 = expm(B)
[B21 B22] = eig(B)
dum3 = B21*diag(exp(diag(B22)))/B21 - B2
B3 = eye(2)/B - inv(B)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
B =
     0     1
     2     3

B1 =
    1.0000    2.7183
    7.3891   20.0855

B2 =
    5.2892    8.4033
   16.8065   30.4990

B21 =
```

```
   -0.8719   -0.2703
    0.4896   -0.9628

B22 =
   -0.5616         0
         0    3.5616

dum3 =
   1.0e-13 *
   -0.1243   -0.2132
   -0.4619   -0.8527

B3 =
     0     0
     0     0
```

이 결과물에서 알 수 있듯이 MATLAB명령문 'B1 = exp(B)' 입력행렬 B의 각 원소에 대한 지수함수값을 포함하는 행렬 B1을 생성한다. MATLAB명령문 'B2 = expm(B)'는 행렬지수(matrix exponential)를 생성한다. 즉, 다음과 같이 행렬지수 B2를 생성한다.

$$\text{expm(X)} = \text{V*diag(exp(diag(D)))/V} \tag{1}$$

MATLAB명령문 '[B21 B22] = eig(B)'는 행렬 B의 특성벡터행렬을 B21에 특성값들로 구성된 대각행렬을 B22에 생성한다. MATLAB명령문 'dum3 = B21*diag(exp(diag(B22)))/B21 - B2'를 실행하면, 식 (1)이 성립함을 확인할 수 있다. MATLAB명령문 'B3 = eye(2)/B - inv(B)'를 실행하면, MATLAB명령문 'eye(2)/B'와 MATLAB명령문 'inv(B)'가 동일함을 알 수 있다.

다음 MATLAB명령문들을 실행해보자.

```
C = magic(4)
C1 = rank(C)
C2 = det(C)
C3 = pinv(C)  % psuedo-inverse
dum4 = C3*C*C3 - C3
dum5 = C*C3*C - C
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다.

```
C =
    16     2     3    13
     5    11    10     8
     9     7     6    12
```

```
     4    14    15     1

C1 =
     3

C2 =
  -1.4495e-12

C3 =
    0.1021   -0.0739   -0.0614    0.0636
   -0.0364    0.0386    0.0261    0.0011
    0.0136   -0.0114   -0.0239    0.0511
   -0.0489    0.0761    0.0886   -0.0864

dum4 =
   1.0e-16 *
    0.2776   -0.2776   -0.5551    0.9714
    0.0694   -0.0694         0   -0.1041
    0.2255   -0.2082   -0.2776    0.4163
    0.0694    0.1388    0.4163   -0.5551

dum5 =
   1.0e-13 *
    0.1066    0.0622    0.0711    0.0711
    0.0711    0.0178    0.0355    0.0355
    0.0888    0.0089    0.0355    0.0533
    0.0711    0.0533    0.0711    0.0089
```

MATLAB명령문 'C = magic(4)'는 차수가 4인 마방진을 생성한다. 또한, MATLAB함수들 rank.m, det.m 그리고 pinv.m은 각각 랭크 C1, 행렬식 C2 그리고 일반화역행렬 (Moore-Pensrose psuedo-inverse) C3를 생성한다. MATLAB명령문들 'dum4 = C3*C*C3 - C3'과 'dum5 = C*C3*C - C'를 실행하면, C3가 C의 일반화역행렬임을 확인할 수 있다. ■

예제 7.6.12 Python을 사용해서 예제 7.6.11을 다시 다루기 위해서, 다음 Python프로그램 MatrixComputation101.Py를 실행해 보자.

```
"""
% Filename MatrixComputation101.m
% Matrix functions
% Programmed by CBS
"""

import numpy as np
import scipy as sp
```

```
9
10 A = np.array(([ 1, 2, 3], [ 4, 5, 6 ], [7, 8, 9 ]));   print(A)
11 A1 = np.sqrt(A);                print('A1 =', A1)
12 dum1 = A1*A1 - A;               print('dum1 =', dum1)
13 A2 = sp.linalg.sqrtm(A);        print('A2 =', A2)
14 dum2 = np.matmul(A2,A2) - A;    print('dum2 =', dum2)
15 A3 = np.linalg.cond(A);         print('A3 =', A3)
16
17 B = np.array(([ 0, 1], [2, 3] ));    print('B =', B)
18 B1 = np.exp(B);                      print('B1 =', B1)
19 B2 = sp.linalg.expm(B);              print('B2 =', B2)
20 B22, B21 = np.linalg.eig(B)
21 print('B22 = Eigenvalues =', B22)
22 print('B21 = Eigenvectors =', B21)
23
24 dum31 = np.diag(B22);                        print('dum31 =', dum31)
25 dum32 = np.diag(np.diag(np.exp(dum31)));     print('dum32 =', dum32)
26 dum33 = np.matmul(B21,dum32);                print('dum33 =', dum33)
27 dum34 = np.linalg.inv(B21);                  print('dum34 =', dum34)
28 dum35 = np.matmul(dum33,dum34);              print('dum35 =', dum35)
29 dum3  = dum35 - B2;                          print('dum3 =', dum3)
30 Binv = np.linalg.inv(B);                     print('B^{-1} =', Binv)
31 B3 = np.matmul(np.eye(2),Binv) - Binv;       print('B3 = ', B3)
32
33 C = np.array(( [ 16, 2, 3, 13 ], [5, 11, 10, 8 ],
34                [ 9, 7, 6, 12  ], [4, 14, 15, 1 ]))
35 C1 = np.linalg.matrix_rank(C);              print('Rank =', C1)
36 C2 = np.linalg.det(C);                     print('det =', C2)
37 C3 = np.linalg.pinv(C);                    print('Psuedo-inverse =', C3)
38 dum4 = np.matmul(np.matmul(C3,C),C3)-C3;   print('dum4 =', dum4)
39 dum5 = np.matmul(np.matmul(C,C3),C)-C;     print('dum5 =', dum5)
40
41 # End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.6.11의 결과의 일부분과 같다. ■

예제 7.6.13 MATLAB에서 복소행렬과 문자행렬을 다루는 예를 살펴보기 위해서, 다음 MATLAB프로그램 MatrixComputation102.m을 실행해 보자.

```
1  % ------------------------------------------------------------------
2  % Filename MatrixComputation102.m
3  % Eigenvalue of Matrix
4  % Programmed by CBS
5  % ------------------------------------------------------------------
6  clear, close all
7  diary MatrixComputation102.txt
8  A = [ 2 -2 1+i ; 0 1+i 0 ; 0 2*i 1-i ]
9  [ Ap, Ad ] = eig(A)
10 dum1 = Ap*Ad/Ap - A
11 [ At Ab ] = balance(A)
12 dum2 = At\A*At - Ab
13 dum3 = eig(A) - eig(Ab)
14 Apoly = poly(A)
15 Apolyn = poly2sym(Apoly)
16 Apolyno = vpa(Apolyn)
17 B = [1 -3 -2; -1  1 -1; 2 4 5]
```

```
[Bv, Bj] = jordan(B)
dum4 = Bv\B*Bv - Bj
syms t
C = [ 1 0 0 ; 0 cos(t) -sin(t) ; 0 sin(t) cos(t) ];
C1 = det(C)
C2 = simplify(C1)
[ Cp Cd ] = eig(C)
dum5 = Cp*Cd/Cp - C
C3 = charpoly(C)
C4 = simplify(C3)
C5 = poly2sym(C4)
C6 = solve(C5)
C7 = simplify(C6)
diary off
% End oh Program
% ------------------------------------------------------------------
```

다음 MATLAB명령문들을 실행해보자.

```
A = [ 2 -2 1+i ; 0 1+i 0 ; 0 2*i 1-i ]
[ Ap, Ad ] = eig(A)
dum1 = Ap*Ad/Ap - A
[ At Ab ] = balance(A)
dum2 = At\A*At - Ab
dum3 = eig(A) - eig(Ab)
Apoly = poly(A)
Apolyn = poly2sym(Apoly)
Apolyno = vpa(Apolyn)
```

이 MATLAB명령문들을 실행한 결과는 다음과 같다. 이 결과물에서도 알 수 있듯이, MATLAB 함수 eig를 사용해서 복소행렬의 특성값과 특성벡터를 구할 수 있다. MATLAB명령문 '[At Ab] = balance(A)'는 식 At\A*At = Ab를 만족하는 상사변환(similar transformation)을 한다. 여기서 행렬 At는 대각원소들이 2의 정수지수 형태인 대각행렬의 조합행렬(permutaiton matrix)이고, Ab는 가능한 한 행과 열의 노름들이 비슷한 행렬이다. 만약 행렬 A가 대칭이면, Ab는 A와 같고 At는 대각행렬이다. MATLAB명령문 'dum2 = At\A*At - Ab'를 실행하면, 이 상사변환이 실행되었음을 확인할 수 있다. MATLAB명령문 'dum3 = eig(A) - eig(Ab)'에서 알 수 있듯이, 행렬 A의 특성값과 밸런스행렬 Ab의 특성값은 같다. 앞에서도 언급했듯이, MATLAB함수 poly.m은 행렬의 특성방정식의 계수들로 구성된 벡터를 생성한다. 또한, MATLAB함수 poly2sym.m은 주어진 벡터에 해당하는 심볼릭멱함수를 생성한다. MATLAB 함수 vpa.m은 변형정밀도부동점연산(variable-precision floating-point arithmetic)을 한다. 즉, 유효한 자리수로 숫자를 변형한다.

```
>> MatrixComputation102
A =
   2.0000 + 0.0000i  -2.0000 + 0.0000i   1.0000 + 1.0000i
```

```
   0.0000 + 0.0000i   1.0000 + 1.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 2.0000i   1.0000 - 1.0000i

Ap =
    1.0000    0.7071    0.5774
         0         0    0.5774
         0   -0.7071    0.5774

Ad =
   2.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   1.0000 - 1.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 1.0000i

dum1 =
     0     0     0
     0     0     0
     0     0     0

At =
     1     0     0
     0     0     1
     0     1     0

Ab =
   2.0000 + 0.0000i   1.0000 + 1.0000i  -2.0000 + 0.0000i
   0.0000 + 0.0000i   1.0000 - 1.0000i   0.0000 + 2.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0000 + 1.0000i

dum2 =
     0     0     0
     0     0     0
     0     0     0

dum3 =
     0
     0
     0

Apoly =    1    -4     6    -4

Apolyn =  x^3 - 4*x^2 + 6*x - 4

Apolyno =  x^3 - 4.0*x^2 + 6.0*x - 4.0

B =
```

```
     1    -3    -2
    -1     1    -1
     2     4     5

Bv =
    -1     1    -1
    -1     0     0
     2     0     1

Bj =
     2     1     0
     0     2     0
     0     0     3

dum4 =
     0     0     0
     0     0     0
     0     0     0

C1 =  cos(t)^2 + sin(t)^2

C2 =  1

Cp =
[ 1,                0,                                      0]
[ 0, (cos(t) - sin(t)*1i)/sin(t) - cos(t)/sin(t),
               (cos(t) + sin(t)*1i)/sin(t) - cos(t)/sin(t)]
[ 0,                1,                                      1]

Cd =
[ 1,                  0,                  0]
[ 0, cos(t) - sin(t)*1i,                  0]
[ 0,                  0, cos(t) + sin(t)*1i]

dum5 =
[ 0, 0, 0]
[ 0, 0, 0]
[ 0, 0, 0]

C3 =
[ 1, - 2*cos(t) - 1, 2*cos(t) + cos(t)^2 + sin(t)^2, - cos(t)^2 - sin(t)^2]

C4 =
[ 1, - 2*cos(t) - 1, 2*cos(t) + 1, -1]
```

```
C5 =
x^3 + (- 2*cos(t) - 1)*x^2 + (2*cos(t) + 1)*x - 1

C6 =
                                                 1
 cos(t) + ((cos(t) - 1)*(cos(t) + 1))^(1/2)
 cos(t) - ((cos(t) - 1)*(cos(t) + 1))^(1/2)

C7 =
                                1
 cos(t) + (cos(t)^2 - 1)^(1/2)
 cos(t) - (cos(t)^2 - 1)^(1/2)
```

■

예제 7.6.14 Python을 사용해서 예제 7.6.13을 다시 다루기 위해서, 다음 Python프로그램 MatrixComputation102.Py를 실행해 보자.

```
"""
% Filename MatrixComputation102.m
% Eigenvalue of Matrix
% Programmed by CBS
"""

import numpy as np

A = np.array(([ 2, -2, 1+1j ], [ 0, 1+1j, 0 ], [ 0, 2*1j, 1-1j ] ))
Ad, Ap = np.linalg.eig(A)
print('Ad = Eigenvalues =', Ad)
print('Ap = Eigenvectors =', Ap)
Adiag = np.diag(Ad);                          print('Adiag =', Adiag)
ApInv = np.linalg.inv(Ap);                    print('ApInv =', ApInv)
dumA101 = np.matmul(Ap,Adiag);                print('dumA101 =', dumA101)
dumA102 = np.matmul(dumA101,ApInv);           print('dumA102 =', dumA102)
dum1 = dumA102 - A;                           print('dum1 =', dum1)

import sympy as sy
AM = sy.Matrix(([ 2, -2, 1+1j ], [ 0, 1+1j, 0 ], [ 0, 2*1j, 1-1j ] ))
AMpolyn = AM.charpoly();                       print('AMpolyn =', AMpolyn)
B = sy.Matrix(( [1., -3., -2.], [-1., 1., -1.], [2., 4., 5.] ))
(Bv, Bj) = B.jordan_form()
print('Bj = Jordan matrix =', Bj)
print('Bv = Eigenvectors =', Bv)
BvInv = Bv.inv();                             print('BvInv =', BvInv)
dumB101 = Bv*Bj*BvInv;                        print('dumB101 =', dumB101)
dum4 = dumB101 - B;                           print('dum4 =', dum4)

t = sy.Symbol('t')
C = sy.Matrix(( [ 1, 0, 0 ], [ 0, sy.cos(t), -sy.sin(t) ], \
                [ 0, sy.sin(t), sy.cos(t) ] ))
print('C =', C)
C1 = C.det();           print('C1 =', C1)
C11 = sy.simplify(C1);    print('C11 =', C11)
# Try symplify(C1) at iPython console
```

```
(Cp, Cd) = C.diagonalize()
print(C.eigenvals())
print(C.eigenvects())
print('Cd = Eigenvalue matrix =', Cd)
print('Cp = Eigenvectors =', Cp)
CpInv = Cp.inv();                          print('CpInv =', CpInv)
dumC101 = Cp*Cd*CpInv;                     print('dumC101 =', dumC101)
dum5 = dumC101 - C;                        print('dum5 =', dum5)
dum51 = sy.simplify(dum5);                    print('dum51 =', dum51)
C3 = C.charpoly();                         print('C3 =', C3)
C4 = sy.simplify(C3);                         print('C4 =', C4)
C6 = sy.solve(C4);                            print('C6 =', C6)
C7 = sy.simplify(C6);                         print('C7 =', C7)

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.6.13의 결과의 일부분과 같다. ■

제7.7절 미분방정식의 해석해

이 좁은 지면에서 미분방정식에 관해 자세히 다룰 수는 없다. 여기서는 MATLAB과 Python을 이용해서 간단한 미분방정식을 푸는 방법을 다루고자 한다. 이 장에서는 미분방정식의 해석해에 대해서 다루고, 다음 장에서 미분방정식의 수치해를 다룰 것이다. 특히 Sydsæter & Hammond & Seierstad & Strøm [59]의 미분방정식 문제들을 포함해서 다루고자 한다. 이후, 이 절에서는 각 k에 대해 C_k는 상수를 의미한다. 이 절에서는 미분방정식들을 아홉 개 그룹들로 나누어 설명하고 한다.

첫 번째 그룹의 첫 번째 미분방정식은 다음과 같다.

$$f'(t) = af(t) \tag{7.7.1}$$

식 (7.7.1)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{df}{f} = adt \tag{7.7.2}$$

식 (7.7.2)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$f(t) = f(0)e^{at} \tag{7.7.3}$$

첫 번째 그룹의 두 번째 미분방정식과 초기조건은 다음과 같다.

$$f'(t) = 0.8f(t), \quad f(0) = 1 \tag{7.7.4}$$

식 (7.7.3)에서 알 수 있듯이, 다음 식이 성립한다.

$$f(t) = e^{0.8t} \tag{7.7.5}$$

첫 번째 그룹의 세 번째 미분방정식은 다음과 같다.

$$y' = y + \cos t \tag{7.7.6}$$

식 (7.7.6)에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{-t}y' + \left[-e^{-t}y\right] = e^{-t}\cos t \tag{7.7.7}$$

즉, 다음 식이 성립한다.

$$\frac{d\left[e^{-t}y\right]}{dt} = e^{-t}\cos t \tag{7.7.8}$$

식 (7.7.8)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$e^{-t}y = \frac{1}{2}e^{-t}\left[\sin t - \cos t\right] + C_{131} \tag{7.7.9}$$

즉, 다음 식이 성립한다.

$$y = -\frac{1}{\sqrt{2}}\cos\left[t + \frac{\pi}{4}\right] + C_{131}e^{t} \tag{7.7.10}$$

첫 번째 그룹의 네 번째 미분방정식과 초기조건은 다음과 같다.

$$y' = y + \cos t\,, \quad y\left(\frac{\pi}{4}\right) = 1 \tag{7.7.11}$$

식 (7.7.10)과 식 (7.7.11)에서 알 수 있듯이, 다음 식이 성립한다.

$$y = -\frac{1}{\sqrt{2}}\cos\left[t + \frac{\pi}{4}\right] + \exp\left(-\frac{\pi}{4}\right)e^{t} \tag{7.7.12}$$

첫 번째 그룹의 다섯 번째 미분방정식과 초기조건은 다음과 같다.

$$P' = 0.03P\left[1 - \frac{P}{200}\right], \quad P(0) = 5 \tag{7.7.13}$$

식 (7.7.13)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{dP}{P\left[1 - \frac{P}{200}\right]} = 0.03 \tag{7.7.14}$$

식 (7.7.14)를 다음과 같이 쓸 수 있다.

$$\left[\frac{1}{P} + \frac{1}{200 - P}\right] dP = 0.03dt \tag{7.7.15}$$

식 (7.7.15)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$\ln \frac{P}{200 - P} = 0.03t + C_{151} \tag{7.7.16}$$

식 (7.7.16)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$C_{151} = \ln \frac{1}{39} \tag{7.7.17}$$

식 (7.7.16)와 식 (7.7.17)에서 다음 로지스틱곡선(logistic curve)을 얻는다.

$$P(t) = \frac{200}{1 + 39e^{-0.03t}} \tag{7.7.18}$$

첫 번째 그룹의 여섯 번째 미분방정식과 초기조건은 다음과 같다.

$$y' = -\frac{1 + y^2}{y}x \quad y(0) = 1 \tag{7.7.19}$$

식 (7.7.19)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{2y}{1 + y^2}y' = -2x \tag{7.7.20}$$

식 (7.7.20)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$\ln\left(1 + y^2\right) = -x^2 + C_{161} \tag{7.7.21}$$

식 (7.7.21)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$y = \sqrt{2e^{-x^2} - 1} \tag{7.7.22}$$

예제 7.7.1 MATLAB을 이용해서 지금까지 다룬 첫 번째 그룹의 미분방정식 문제들을 살펴보기 위해서, 다음 MATLAB프로그램 ODE101.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename ODE101.m
% Ordinary Differential Equation 101
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary ODE101.txt
sol11 = dsolve('Df = a*f')
sol12 = dsolve('Df = 0.8*f','f(0)=1','t')
eqn13 = 'Dy = y + cos(t)'
sol13 = dsolve(eqn13,'t')
pretty(sol13)
sol14 = dsolve('Dy = y + cos(t)','y(pi/4) = 1','t')
eqn15 = 'DP = 0.03*P*(1-P/200)', init105='P(0)=5'
sol15 = dsolve(eqn15,init105,'t')
sol16 = dsolve('Dy = - (1+y^2)*x/y','y(0)=1','x')
% Plotting
% Should remind that every variable is symbolic.
set(gca,'fontsize',11,'fontweigh','bold')
subplot(2,2,1)
h12 = ezplot(sol12, 't', [-0.9, 0.9])
set(h12,'LineWidth',2,'color',[1, 0, 0])
xlabel('\bf f'), ylabel('\bf t','rotation',0)
subplot(2,2,2)
t = linspace(-0.5,3.5,201);
y14 = eval(vectorize(sol14));
plot(t,y14,'g-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-0.5,3.5])
xlabel('\bf t','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
subplot(2,2,3)
h15 = ezplot(sol15, [-100, 400])
set(h15,'LineWidth',2,'color',[0, 0, 1])
xlabel('\bf t'), ylabel('\bf P','rotation',0)
subplot(2,2,4)
x = linspace(-0.8,0.8,201);
y16 = eval(vectorize(sol16));
plot(x,y16,'k-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-0.8,0.8])
xlabel('\bf x'), ylabel('\bf y','rotation',0)
saveas(gcf,'ODE101.jpg')
% End of Program
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 앞에서 기술한 내용을 확인할 수 있다. 또한 그림 7.7.1이 그려진다. ■

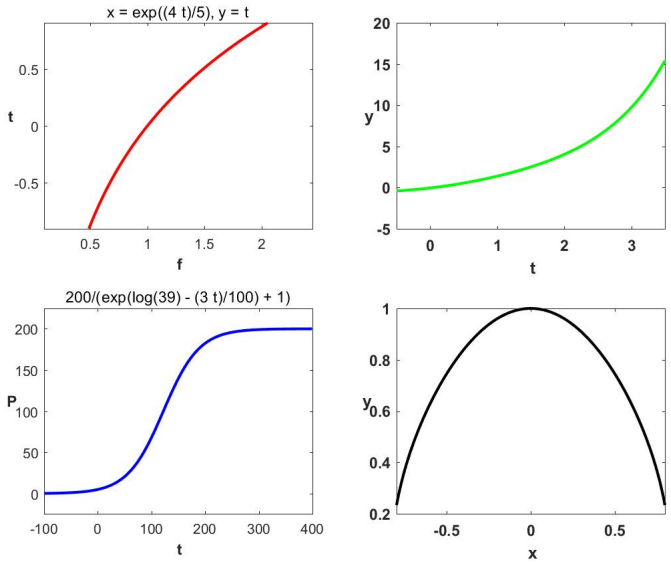


그림 7.7.1. 미분방정식의 해석해 I

예제 7.7.2 Python을 사용해서 예제 7.7.1을 다시 다루기 위해서, 다음 Python프로그램 ODE101.Py를 실행해 보자.

```
"""
% Filename ODE101.Py
% Ordinary Differential Equation 101
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

# Example 11
f = sy.Function('f')
t = sy.symbols('t')
a = sy.symbols('a', nonzero=True)
eq11 = sy.Eq(sy.Derivative(f(t),t), a*f(t))
sol11 = sy.dsolve(eq11)

# Example 12
## ODE
t = sy.Symbol("t", real=True)
a = sy.symbols('a', real=True)
y = sy.Function('y')
eq12 = sy.Eq(sy.diff(y(t),t) - a*y(t))
```

```
print("ODE:")
display(eq12)
## General solution
print("General solution:")
sol12 = sy.dsolve(eq12, y(t)).rhs  # take only right hand side
display(sy.Eq(y(t), sol12))
## Let a = 0.8
sol12b = sol12.subs(a,0.8)
display(sy.Eq(y(t), sol12b))
## Initial conditions:
condition0 = sy.Eq(sol12b.subs(t,0),1)    # y(0) = 1
display(condition0)
## Substitute back into solution:
C1 = sy.symbols("C1")  # generic constants
constant1 = sy.solve([condition0], (C1))
display(constant1)
sol12c = sy.simplify(sol12b.subs(constant1))
print("Solution with initial conditions:")
display(sy.Eq(y(t),sol12c))
display(sol12c)
## plt.subplot(2,2,1)
fig = plt.figure()
plt.subplot(2,2,1)
sol12cLam = sy.lambdify(t, sol12c, modules=['numpy'])
tt = np.linspace(-1,1, 201)
yy = sol12cLam(tt)
plt.plot(tt,yy,'r-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.xlabel('f',fontsize=12)

# Example 13
eq13 = sy.Eq(sy.diff(y(t),t) - y(t) - sy.cos(t))
print("ODE:")
display(eq13)
sol13 = sy.dsolve(eq13, y(t)).rhs  # take only right hand side
sol13 = sy.simplify(sol13)
display(sy.Eq(y(t), sol13))

# Example 14
condition0 = sy.Eq(sol13.subs(t,np.pi/4),1)    # y(pi/4) = 1
display(condition0)
C1 = sy.symbols("C1")  # generic constants
constant1 = sy.solve([condition0], (C1))
display(constant1)
sol14 = sy.simplify(sol13.subs(constant1))
print("Solution with initial conditions:")
display(sy.Eq(y(t),sol14))
## plt.subplot(2,2,2)
plt.subplot(2,2,2)
sol14Lam = sy.lambdify(t, sol14, modules=['numpy'])
tt = np.linspace(-1,4, 201)
yy = sol14Lam(tt)
plt.plot(tt,yy,'g-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.xlabel('y',fontsize=12)

# Example 15: Logistic Curve
P = sy.Function('P')
t = sy.symbols('t')
eq15 = sy.Eq(sy.Derivative(P(t),t), 0.03*P(t)*(1 - P(t)/200.0))
sol15 = sy.dsolve(eq15)
display(sol15)
## plt.subplot(2,2,3)
```

```
plt.subplot(2,2,3)
f15s = lambda t: 200/(1+39*sy.exp(-0.03*t))
f15Vec = np.vectorize(f15s)
tt = np.linspace(-100,400,201)
PP = f15Vec(tt)
plt.plot(tt,PP,color='blue',lw=2)
plt.axis([-100, 400, -50, 250 ])
plt.xlabel('t'), plt.ylabel('P(t)')

# Example 16
y = sy.Function('y')
x = sy.symbols('x')
eq16 = sy.Eq(sy.diff(y(x),x) + (1+y(x)**2)/y(x)*x)
display(eq16)
sol16 = sy.dsolve(eq16)
display(sol16)
sol16s0 = sol16[0].subs(C1,2)
display(sol16s0)
sol16s1 = sol16[1].subs(C1,2)
display(sol16s1)
## plt.subplot(2,2,4)
plt.subplot(2,2,4)
sol16s1Lam = sy.lambdify(x, sol16s1.rhs, modules=['numpy'])
       # The '.rhs' is very Important.
xx = np.linspace(-0.83,0.83, 201)
yy = sol16s1Lam(xx)
plt.plot(xx,yy,'k-',lw=2.0)
plt.axis([-0.8, 0.8, -0.1, 1.1])
plt.xlabel('t',fontsize=12), plt.xlabel('y',fontsize=12)

plt.show()
fig.savefig('ODE101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.7.1의 결과와 같다. ■

두 번째 그룹으로 Sydsæter & Hammond & Seierstad & Strøm [59]의 미분방정식 문제들을 다루기로 하자.

두 번째 그룹의 첫 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = x + t\,, \quad x(0) = 0 \tag{7.7.23}$$

식 (7.7.23)에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{-t}\dot{x} + \left[-e^{-t}x\right] = e^{-t}t \tag{7.7.24}$$

즉, 다음 식이 성립한다.

$$\frac{d\left[e^{-t}x\right]}{dt} = e^{-t}t \tag{7.7.25}$$

식 (7.7.25) 의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$e^{-t}x = -[t+1]e^{-t} + C_{211} \tag{7.7.26}$$

즉, 다음 식이 성립한다.

$$x = -[t+1] + C_{211}e^{t} \tag{7.7.27}$$

식 (7.7.27) 과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$x(t) = -[t+1] + e^{t} \tag{7.7.28}$$

두 번째 그룹의 두 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = -2t\,x^{2}\,, \quad x(1) = -1 \tag{7.7.29}$$

식 (7.7.29) 에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{\dot{x}}{x^{2}} = -2t \tag{7.7.30}$$

식 (7.7.30) 의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$-\frac{1}{x} = -t^{2} + C_{221} \tag{7.7.31}$$

식 (7.7.31) 과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$-\frac{1}{x} = -t^{2} + 2 \tag{7.7.32}$$

즉, 다음 식이 성립한다.

$$x(t) = \frac{1}{t^{2}-2} \tag{7.7.33}$$

두 번째 그룹의 세 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = B[x-a][x-b] \tag{7.7.34}$$

여기서 B는 0이 아닌 상수이고 a와 b는 서로 다른 상수들이다. 식 (7.7.34)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{1}{b-a}\left[\frac{1}{x-a}-\frac{1}{x-b}\right]\dot{x}=B \tag{7.7.35}$$

식 (7.7.35)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$\ln\left|\frac{x-a}{x-b}\right|=[b-a]Bt+C_{231} \tag{7.7.36}$$

즉, 다음 식이 성립한다.

$$\frac{x-a}{x-b}=C_{232}\exp\left([b-a]Bt\right) \tag{7.7.37}$$

두 번째 그룹의 네 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x}=-\frac{1}{2}[x+1][x-2]\,,\quad x(0)=1 \tag{7.7.38}$$

식 (7.7.37)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{x+1}{x-2}=-2\exp\left(\frac{3t}{2}\right) \tag{7.7.39}$$

즉, 다음 식이 성립한다.

$$x(t)=\frac{4\exp\left(\frac{3t}{2}\right)-1}{2\exp\left(\frac{3t}{2}\right)+1} \tag{7.7.40}$$

두 번째 그룹의 다섯 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x}+2tx=4t\,,\quad x(0)=-2 \tag{7.7.41}$$

식 (7.7.41)에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{t^2}\dot{x}+\left[2te^{t^2}x\right]=e^{t^2}\cdot 4t \tag{7.7.42}$$

즉, 다음 식이 성립한다.

$$\frac{d\left[e^{t^2}x\right]}{dt}=4te^{t^2} \tag{7.7.43}$$

식 (7.7.43)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$e^{t^2}x=2e^{t^2}+C_{251} \tag{7.7.44}$$

식 (7.7.44)와 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{t^2}x = 2e^{t^2} + 4 \tag{7.7.45}$$

즉, 다음 식이 성립한다.

$$x(t) = 2 + 4e^{-t^2} \tag{7.7.46}$$

예제 7.7.3 MATLAB을 이용해서 지금까지 다룬 두 번째 그룹의 문제들을 살펴보기 위해서, 다음 MATLAB프로그램 ODE102.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename ODE102.m
% Sydsaester, Hammond, Seierstad, and Strom (2008)
% Further Mathematics for Economic Analysis, Chapter 5
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary ODE102.txt
SHSS511g = dsolve('Dx=x+t','t')
SHSS511 = dsolve('Dx=x+t','x(0)=0','t')
SHSS512g = dsolve('Dx = -2*t*x^2')
SHSS512 = dsolve('Dx = -2*t*x^2','x(1)=-1')
SHSS513g = dsolve('Dx=B*(x-a)*(x-b)','t')
SHSS513 = dsolve('Dx=-1/2*(x+1)*(x-2)','x(0)=1','t')
SHSS513s = simplify(SHSS513)
SHSS514g = dsolve('Dx+2*t*x=4*t')
SHSS514 = dsolve('Dx+2*t*x=4*t','x(0)=-2')
% Plotting
set(gca,'fontsize',11,'fontweigh','bold')
subplot(2,2,1)
h511 = ezplot(SHSS511, 't', [-1, 1])
set(h511,'LineWidth',2,'color',[1, 0, 0])
xlabel('\bf x','fontsize',12)
ylabel('\bf t','fontsize',12,'rotation',0)
subplot(2,2,2)
h512 = ezplot(SHSS512, [2, 6])
set(h512,'LineWidth',2,'color',[0, 1, 0])
xlabel('\bf t','fontsize',12)
ylabel('\bf x','fontsize',12,'rotation',0)
subplot(2,2,3)
h513 = ezplot(SHSS513, [-6, 4])
set(h513,'LineWidth',2,'color',[0, 0, 1])
xlabel('\bf t','fontsize',12)
ylabel('\bf x','fontsize',12,'rotation',0)
subplot(2,2,4)
h514 = ezplot(SHSS514, [-2,1])
set(h514,'LineWidth',2,'color',[0.333,0.333,0.334])
xlabel('\bf t','fontsize',12)
ylabel('\bf x','fontsize',12,'rotation',0)
saveas(gcf,'ODE102.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 앞에서 기술한 내용을 확인할 수 있다. 또한 그림 7.7.2가 그려진다. ■

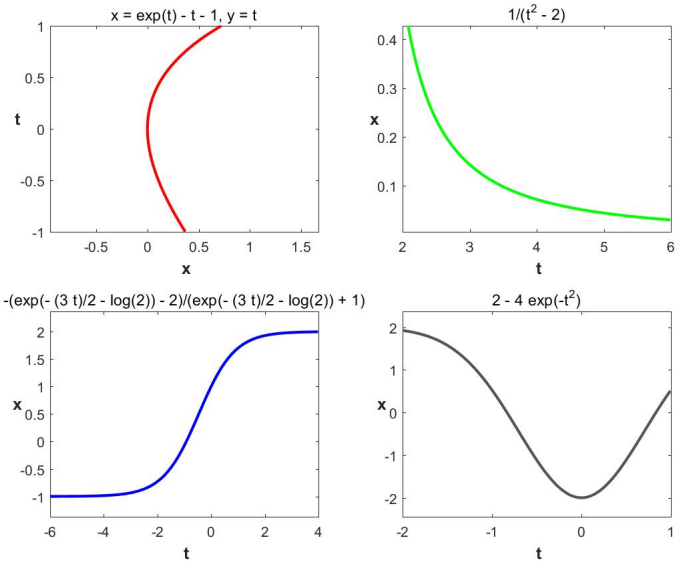


그림 7.7.2. 미분방정식의 해석해 II

예제 7.7.4 Python을 사용해서 예제 7.7.3을 다시 다루기 위해서, 다음 Python프로그램 ODE102.Py를 실행해 보자.

```
"""
% Filename ODE102.Py
% Ordinary Differential Equation 102
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

# Example 11
x = sy.Function('x')
t = sy.symbols('t')
eq11 = sy.Eq(sy.Derivative(x(t),t), x(t)+t)
sol11 = sy.dsolve(eq11)
display(sol11)
sol11s = sy.simplify(sol11.subs(C1,1))
print("Solution with initial conditions:")
display(sol11s)
## plt.subplot(2,2,1)
fig = plt.figure()
plt.subplot(2,2,1)
sol11Lam = sy.lambdify(t, sol11s.rhs, modules=['numpy'])
```

```
tt = np.linspace(-1,1, 201)
xx = sol11Lam(tt)
plt.plot(tt,xx,'r-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.xlabel('x',fontsize=12)

# Example 12
eq12 = sy.Eq(sy.diff(x(t),t),-2*t*x(t)**2)
print("ODE:")
display(eq12)
sol12 = sy.dsolve(eq12, x(t)).rhs   # take only right hand side
sol12 = sy.simplify(sol12)
display(sy.Eq(x(t), sol12))
condition0 = sy.Eq(sol12.subs(t,1),-1)    # x(1) = -1
display(condition0)
C1 = sy.symbols("C1")  # generic constants
constant1 = sy.solve([condition0], (C1))
display(constant1)
sol12s = sy.simplify(sol12.subs(constant1))
print("Solution with initial conditions:")
display(sy.Eq(x(t),sol12s))
## plt.subplot(2,2,2)
plt.subplot(2,2,2)
sol12Lam = sy.lambdify(t, sol12s, modules=['numpy'])
tt = np.linspace(2,6, 201)
xx = sol12Lam(tt)
plt.plot(tt,xx,'g-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.xlabel('y',fontsize=12)

# Example 13: Losistic Curve
eq13 = sy.Eq(sy.diff(x(t),t)/(x(t)+1)/(x(t)-2), -1/2)
print("ODE:")
display(eq13)
sol13 = sy.dsolve(eq13, x(t)).rhs
sol13 = sy.simplify(sol13)
display(sy.Eq(x(t), sol13))
print(sol13)
condition0 = sy.Eq(sol13.subs(t,0),1)    # x(0)=1
display(condition0)
C1 = sy.symbols("C1")  # generic constants
constant1 = sy.solve([condition0], (C1))
display(constant1)
print(constant1)
c31 = complex(-0.231049060186648,-np.pi/3); print(c31)
c32 = complex(-0.231049060186648,np.pi/3);  print(c32)
c33 = complex(-0.231049060186648,np.pi);  print(c33)
sol131s = sy.simplify(sol13.subs(C1,c31))
print("Solution with initial conditions:")
display(sy.Eq(x(t),sol131s))
sol132s = sy.simplify(sol13.subs(C1,c32))
print("Solution with initial conditions:")
display(sy.Eq(x(t),sol132s))
sol133s = sy.simplify(sol13.subs(C1,c33))
print("Solution with initial conditions:")
display(sy.Eq(x(t),sol133s))
## plt.subplot(2,2,3)
plt.subplot(2,2,3)
f131s = lambda t: (-2*np.exp(1.5*t)+0.5)/(-1*np.exp(1.5*t)-0.5)
f131sVec = np.vectorize(f131s)
tt = np.linspace(-6,4,201)
xx = f131sVec(tt)
plt.plot(tt,xx,color='blue',lw=2)
```

```
plt.xlabel('t',fontsize=12), plt.xlabel('y',fontsize=12)
plt.axis([-6, 4, -1.5, 2.5 ])

# Example 14
x = sy.Function('x')
t = sy.symbols('t')
eq14 = sy.Eq(sy.diff(x(t),t) + 2*t*x(t)-4*t,0)
display(eq14)
sol14 = sy.dsolve(eq14).rhs
display(sol14)
## Initial conditions
cnd0 = sy.Eq(sol14.subs(t,0),-2)        # y(0) = -1
display(cnd0)
C1 = sy.symbols("C1")             # generic constants
const14 = sy.solve([cnd0], (C1))
display(const14)
## Substitute back into solution
sol14s = sy.simplify(sol14.subs(const14))
print("Solution with initial conditions:")
display(sy.Eq(x(t),sol14s))
## plt.subplot(2,2,4)
plt.subplot(2,2,4)
sol14sLam = sy.lambdify(t, sol14s, modules=['numpy'])
tt = np.linspace(-2,1,201)
xx = sol14sLam(tt)
plt.plot(tt,xx,'k-',lw=2.0)
plt.axis([-2, 1, -2.5, 2.5 ])
plt.xlabel('t',fontsize=12), plt.xlabel('x',fontsize=12)

plt.show()
fig.savefig('ODE102Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.7.3의 결과와 같다. ■

세 번째 그룹의 첫 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{y} = 2y + x^2, \quad y(0) = 1 \tag{7.7.47}$$

식 (7.7.47)에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{-2x}\dot{y} + \left[-2e^{-2x}y\right] = x^2 e^{-2x} \tag{7.7.48}$$

즉, 다음 식이 성립한다.

$$\frac{d\left[e^{-2x}y\right]}{dx} = x^2 e^{-2x} \tag{7.7.49}$$

식 (7.7.49)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$e^{-2x}y = \int x^2 e^{-2x}\,dx = -\frac{2x^2 + 2x + 1}{4}e^{-2x} + C_{311} \tag{7.7.50}$$

식 (7.7.50)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{-2x}y = -\frac{2x^2+2x+1}{4}e^{-2x} + \frac{5}{4} \tag{7.7.51}$$

즉, 다음 식이 성립한다.

$$y = -\frac{2x^2+2x+1}{4} + \frac{5}{4}e^{2x} \tag{7.7.52}$$

세 번째 그룹의 두 번째 미분방정식과 초기조건은 다음과 같다.

$$y'' + 2y' = e^x\,, \quad y(0) = 1\,, \quad y'(0) = 0 \tag{7.7.53}$$

식 (7.7.53)에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{2x}y'' + \left[2e^{2x}y'\right] = e^{3x} \tag{7.7.54}$$

즉, 다음 식이 성립한다.

$$\frac{d\left[e^{2x}y'\right]}{dx} = e^{3x} \tag{7.7.55}$$

식 (7.7.55)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$e^{2x}y' = \frac{1}{3}e^{3x} + C_{321} \tag{7.7.56}$$

식 (7.7.56)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{2x}y' = \frac{1}{3}e^{3x} - \frac{1}{3} \tag{7.7.57}$$

즉, 다음 식이 성립한다.

$$y' = \frac{1}{3}e^{x} - \frac{1}{3}e^{-2x} \tag{7.7.58}$$

식 (7.7.58)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$y = \frac{1}{3}e^{x} + \frac{1}{6}e^{-2x} + \frac{1}{2} \tag{7.7.59}$$

세 번째 그룹의 세 번째 미분방정식과 초기조건은 다음과 같다.

$$y'' + 2y' = \sin x\,, \quad y(0) = -1\,, \quad y'(0) = 1 \tag{7.7.60}$$

식 (7.7.60)에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{2x}y'' + \left[2e^{2x}y'\right] = e^{2x}\sin x \tag{7.7.61}$$

즉, 다음 식이 성립한다.

$$\frac{d\left[e^{2x}y'\right]}{dx} = e^{2x}\sin x \tag{7.7.62}$$

식 (7.7.62)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$e^{2x}y' = \frac{2\sin x - \cos x}{5}e^{2x} + C_{331} \tag{7.7.63}$$

식 (7.7.63)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{2x}y' = \frac{2\sin x - \cos x}{5}e^{2x} + \frac{6}{5} \tag{7.7.64}$$

즉, 다음 식이 성립한다.

$$y' = \frac{2\sin x - \cos x}{5} + \frac{6}{5}e^{-2x} \tag{7.7.65}$$

식 (7.7.65)에서 알 수 있듯이, 다음 식이 성립한다.

$$y = \frac{-2\cos x - \sin x}{5} - \frac{3}{5}e^{-2x} + C_{332} \tag{7.7.66}$$

식 (7.7.66)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$y = \frac{-2\cos x - \sin x}{5} - \frac{3}{5}e^{-2x} \tag{7.7.67}$$

세 번째 그룹의 네 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{y} = -ty - t\,, \quad y(0) = 0 \tag{7.7.68}$$

식 (7.7.68)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{\dot{y}}{y+1} = -t \tag{7.7.69}$$

식 (7.7.69)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$\ln(y+1) = -\frac{1}{2}t^2 + C_{341} \tag{7.7.70}$$

식 (7.7.70)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$\ln(y+1) = -\frac{1}{2}t^2 \tag{7.7.71}$$

즉, 다음 식이 성립한다.

$$y(t) = \exp\left(-\frac{1}{2}t^2\right) - 1 \tag{7.7.72}$$

예제 7.7.5 MATLAB을 이용해서 지금까지 다룬 세 번째 그룹의 문제들을 살펴보기 위해서, 다음 MATLAB프로그램 ODE103.m을 실행해보자.

```
% ------------------------------------------------------------------------
%  Filename: ODE103.m
%  Ordinary Differencial Equation 103
%  Programmed by CBS
%------------------------------------------------------------------------
clear all, close all, clc
diary ODE103.txt
yD = dsolve('Dy=2*y+x^2','y(0)=1','x')
eqnE ='D2y+2*Dy=exp(x)',
initE = 'y(0)=1,Dy(0)=0'
yE = dsolve(eqnE,initE,'x')
yF = dsolve('D2y+2*Dy=sin(x)','y(0)=-1','Dy(0)=1','x')
yF = simplify(yF)
yF = expand(yF)
yG = dsolve('Dy=-y*t-t','y(0)=0','t')
% Plotting
% Should remind that every variable is symbolic.
subplot(2,2,1)
hD = ezplot(yD, [-0.5,5.5])
set(gca,'fontsize',11,'fontweigh','bold')
set(hD,'LineWidth',2,'color',[1, 0, 0])
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
subplot(2,2,2)
x = linspace(-0.5,5.5,201);
yEE = eval(vectorize(yE));
plot(x,yEE,'g-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-0.5,5.5])
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
subplot(2,2,3)
hF = ezplot(yF, [-2, 4])
set(gca,'fontsize',11,'fontweigh','bold')
set(hF,'LineWidth',2,'color',[0, 0, 1])
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
```

```
subplot(2,2,4)
t = linspace(-0.5,0.5,201);
yGG = eval(vectorize(yG));
plot(t,yGG,'k-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[-0.5,0.5])
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
axis([-0.5, 0.5, -0.13, 0.01 ])
saveas(gcf,'ODE103.jpg')
diary off
% End of Program
%--------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 앞에서 기술한 내용을 확인할 수 있다. 또한 그림 7.7.3이 그려진다. ■

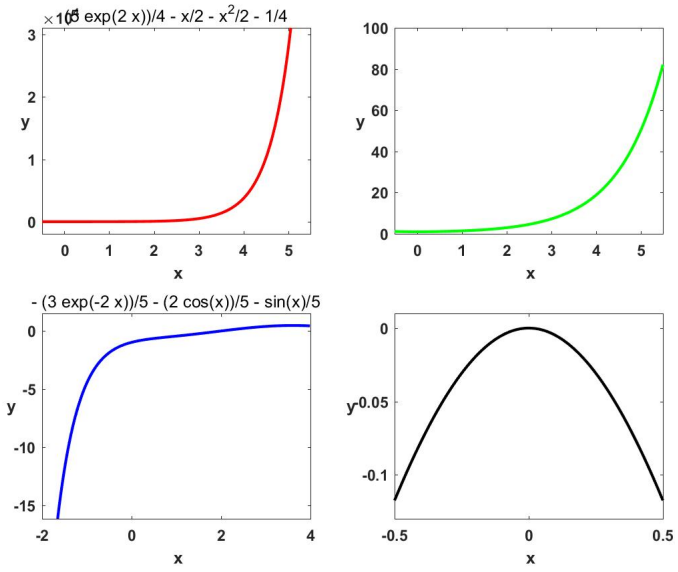


그림 7.7.3. 미분방정식의 해석해 III

예제 7.7.6 Python을 사용해서 예제 7.7.5을 다시 다루기 위해서, 다음 Python프로그램 ODE103.Py를 실행해 보자.

```
"""
% Filename ODE103.Py
% Ordinary Differential Equation 103
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
```

```
sy.init_printing()  # LaTeX-like pretty printing for IPython

# Example 11
y = sy.Function('y')
x = sy.symbols('x')
eq11 = sy.Eq(sy.Derivative(y(x),x) - 2*y(x)-x**2)
sol11 = sy.dsolve(eq11).rhs
display(sol11)
## Initial condition
condition0 = sy.Eq(sol11.subs(x,0),1)    # y(0) = 1
display(condition0)
C1 = sy.symbols("C1")  # generic constants
constant1 = sy.solve([condition0], (C1))
display(constant1)
sol11s = sy.simplify(sol11.subs(constant1))
print("Solution with initial conditions:")
display(sol11s)
## plt.subplot(2,2,1)
fig = plt.figure()
plt.subplot(2,2,1)
sol11Lam = sy.lambdify(x, sol11s, modules=['numpy'])
xx = np.linspace(-1,6, 201)
yy = sol11Lam(xx)
plt.plot(xx,yy,'r-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.xlabel('y',fontsize=12)
plt.axis([-1,6,0,3*10**4])

# Example 12
y = sy.Function('y')
x = sy.symbols('x')
eq12 = sy.Eq(sy.diff(y(x),x,x)+2*sy.diff(y(x),x)-sy.exp(x),0)
display(eq12)
sol12 = sy.dsolve(eq12).rhs
display(sol12)
## Initial conditions
cnd0 = sy.Eq(sol12.subs(x,0), 1)        # y(0) = 1
display(cnd0)
dsol12 = sol12.diff(x)
display(dsol12)
cnd1 = sy.Eq(sol12.diff(x).subs(x,0), 0) # y'(0) = 0
display(cnd1)
##  Solve for C1, C2
C1, C2 = sy.symbols("C1, C2")            # generic constants
const12 = sy.solve([cnd0, cnd1], (C1, C2))
display(const12)
## Substitute back into solution
sol12s = sy.simplify(sol12.subs(const12))
print("Solution with initial conditions:")
display(sy.Eq(y(x),sol12s))
## plt.subplot(2,2,2)
plt.subplot(2,2,2)
sol12sLam = sy.lambdify(x, sol12s, modules=['numpy'])
xx = np.linspace(-0.5,5.5,201)
yy = sol12sLam(xx)
plt.plot(xx,yy,'k-',lw=2.0)
plt.axis([-0.5, 5.5, -0.1, 80])
plt.xlabel('x',fontsize=12), plt.xlabel('y',fontsize=12)

# Example 13
y = sy.Function('y')
x = sy.symbols('x')
```

```
eq13 = sy.Eq(sy.diff(y(x),x,x)+2*sy.diff(y(x),x)-sy.sin(x),0)
display(eq13)
sol13 = sy.dsolve(eq13).rhs
display(sol13)
## Initial conditions
cnd0 = sy.Eq(sol13.subs(x,0), -1)        # y(0) = -1
display(cnd0)
dsol13 = sol13.diff(x)
display(dsol13)
cnd1 = sy.Eq(sol13.diff(x).subs(x,0), 1) # y'(0) = 0
display(cnd1)
##  Solve for C1, C2
C1, C2 = sy.symbols("C1, C2")            # generic constants
const13 = sy.solve([cnd0, cnd1], (C1, C2))
display(const13)
## Substitute back into solution
sol13s = sy.simplify(sol13.subs(const13))
print("Solution with initial conditions:")
display(sy.Eq(y(x),sol13s))
## plt.subplot(2,2,3)
plt.subplot(2,2,3)
sol13sLam = sy.lambdify(x, sol13s, modules=['numpy'])
xx = np.linspace(-2,4,201)
yy = sol13sLam(xx)
plt.plot(xx,yy,'k-',lw=2.0)
plt.axis([-2, 4, -16, 1])
plt.xlabel('x',fontsize=12), plt.xlabel('y',fontsize=12)

# Example 14
y = sy.Function('y')
t = sy.symbols('t')
eq14 = sy.Eq(sy.diff(y(t),t)+t*y(t)+t)
print("ODE:")
display(eq14)
sol14 = sy.dsolve(eq14, y(t)).rhs   # take only right hand side
sol14 = sy.simplify(sol14)
display(sol14)
display(sy.Eq(y(t), sol14))
condition0 = sy.Eq(sol14.subs(t,0),0)    # y(0)=0
display(condition0)
C1 = sy.symbols("C1")  # generic constants
constant1 = sy.solve([condition0], (C1))
display(constant1)
sol14s = sy.simplify(sol14.subs(constant1))
print("Solution with initial conditions:")
display(sy.Eq(y(t),sol14s))
## plt.subplot(2,2,4)
plt.subplot(2,2,4)
sol14sLam = sy.lambdify(t, sol14s, modules=['numpy'])
tt = np.linspace(-0.5, 0.5, 201)
yy = sol14sLam(tt)
plt.plot(tt,yy,'g-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.xlabel('y',fontsize=12)
plt.axis([-0.5, 0.5, -0.13, 0.01 ])

plt.show()
fig.savefig('ODE103Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.7.5의 결과와 같다. ■

네 번째 그룹으로 Sydsæter & Hammond & Seierstad & Strøm [59] 의 미분방정식 문제들을 다루기로 하자.

네 번째 그룹의 첫 번째 미분방정식과 초기조건은 다음과 같다.

$$1 + tx^2 + t^2 x\dot{x} = 0\,, \quad x(2) = 1 \tag{7.7.73}$$

식 (7.7.73) 에서 알 수 있듯이, 다음 식이 성립한다.

$$2t \cdot x^2 + t^2 \cdot 2x\dot{x} = -2 \tag{7.7.74}$$

즉, 다음 식이 성립한다.

$$\frac{d\left[t^2 x^2\right]}{dt} = -2 \tag{7.7.75}$$

식 (7.7.75) 의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$t^2 x^2 = -2t + C_{411} \tag{7.7.76}$$

식 (7.7.76) 과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$x^2(t) = \frac{8 - 2t}{t^2} \tag{7.7.77}$$

네 번째 그룹의 두 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = -tx + t^3 x^3\,, \quad x(0) = 3 \tag{7.7.78}$$

식 (7.7.78) 은 Bernoullie방정식이므로 확률변환 $z \doteq 1/x^2$ 를 취하면, 식 (7.7.78) 로부터 다음 식을 얻을 수 있다.

$$\dot{z} - 2tz = -2t^3 \tag{7.7.79}$$

식 (7.7.79) 에서 알 수 있듯이, 다음 식이 성립한다.

$$\exp\left(-t^2\right)\dot{z} + \left[-2t\exp\left(-t^2\right)\right] z = -2t^3 \exp\left(-t^2\right) \tag{7.7.80}$$

즉, 다음 식이 성립한다.

$$\frac{d\left[\exp\left(-t^2\right) z\right]}{dt} = -2t^3 \exp\left(-t^2\right) \tag{7.7.81}$$

식 (7.7.81)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$\exp\left(-t^2\right) z = -2 \int t^3 \exp\left(-t^2\right)\, dt = t^2 \exp\left(-t^2\right) + \exp\left(-t^2\right) + C_{421} \tag{7.7.82}$$

즉, 다음 식이 성립한다.

$$z = t^2 + 1 + C_{421} \exp\left(t^2\right) \tag{7.7.83}$$

식 (7.7.83)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$z = t^2 + 1 - \frac{8}{9} \exp\left(t^2\right) \tag{7.7.84}$$

즉, 다음 식이 성립한다.

$$x^2 = \frac{1}{1 + t^2 - \frac{8}{9} \exp\left(t^2\right)} \tag{7.7.85}$$

네 번째 그룹의 세 번째 미분방정식과 초기조건은 다음과 같다.

$$y' - 1 + 2x[y - x]^2 = 0\,, \quad y(0) = -\frac{1}{2} \tag{7.7.86}$$

식 (7.7.86)에서 알 수 있듯이, 다음 식이 성립한다.

$$[y - x]' + 2x[y - x]^2 = 0 \tag{7.7.87}$$

식 (7.7.87)에 변수변환 $z \doteq y - x$를 적용하면, 다음 식을 얻는다.

$$\frac{z'}{z^2} = -2x \tag{7.7.88}$$

식 (7.7.88)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$-\frac{1}{z} = -x^2 + C_{432} \tag{7.7.89}$$

식 (7.7.89)와 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$-\frac{1}{z} = -x^2 + 2 \tag{7.7.90}$$

즉, 다음 식이 성립한다.

$$z = \frac{1}{x^2 - 2} \tag{7.7.91}$$

따라서 다음 식이 성립한다.

$$y = x + \frac{1}{x^2 - 2} \tag{7.7.92}$$

네 번째 그룹의 네 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{k} = \sqrt{k} - k\,, \quad k(0) = 0 \tag{7.7.93}$$

식 (7.7.93)은 신고전파 모형인 Solow-Swan성장모형의 특수한 경우이다. 식 (7.7.93)에서 알 수 있듯이, 변수 $z \doteq \sqrt{k}$는 다음 식을 만족한다.

$$\dot{z} + \frac{1}{2}z = \frac{1}{2} \tag{7.7.94}$$

식 (7.7.94)에서 알 수 있듯이, 다음 식이 성립한다.

$$e^{t/2}\dot{z} + \frac{1}{2}e^{t/2}z = \frac{1}{2}e^{t/2} \tag{7.7.95}$$

즉, 다음 식이 성립한다.

$$\frac{d}{dt}\left[e^{t/2}z\right] = \frac{1}{2}e^{t/2} \tag{7.7.96}$$

식 (7.7.96)의 양변을 적분하면, 다음 식이 성립함을 알 수 있다.

$$e^{t/2}z = e^{t/2} + C_{441} \tag{7.7.97}$$

식 (7.7.97)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$z = 1 - e^{-t/2} \tag{7.7.98}$$

즉, 다음 식이 성립한다.

$$k(t) = \left[1 - e^{-t/2}\right]^2 \tag{7.7.99}$$

예제 7.7.7 MATLAB을 이용해서 지금까지 다룬 네 번째 그룹 문제들을 살펴보기 위해서, 다음 MATLAB프로그램 ODE104.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename ODE104.m
% Sydsaester, Hammond, Seierstad, and Strom (2008)
% Further Mathematics for Economic Analysis, Chapter 5, No.2
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary ODE104.txt
syms t x
SHSS521g = dsolve('1+t*x^2+t^2*x*Dx','t')
SHSS521 = dsolve('1+t*x^2+t^2*x*Dx','x(2)=1.0','t')
SHSS521sq = SHSS521^2
SHSS522g = dsolve('Dx=-t*x+t^3*x^3')
SHSS522 = dsolve('Dx=-t*x+t^3*x^3','x(0)=3')
SHSS522sq = SHSS522^2
SHSS523g = dsolve('Dy-1+2*x*(y-x)^2','x')
SHSS523 = dsolve('Dy-1+2*t*(y-t)^2','y(0)=-1/2')
% Neoclassical Solow-Swan model
SHSS524g = dsolve('Dk=sqrt(k)-k','t')
SHSS524 = dsolve('Dk=sqrt(k)-k','k(0)=0','t')
dSHSS524 = diff(SHSS524)
tt = linspace(0,20,201);
xk = double(subs(SHSS524,'t',tt));
yk = double(subs(dSHSS524,'t',tt));
% Plotting
set(gca,'fontsize',11,'fontweigh','bold')
subplot(2,2,1)
h521 = ezplot(SHSS521, 't', [0.5,4])
set(h521,'LineWidth',2,'color',[1, 0, 0])
xlabel('\bf x','fontsize',12)
ylabel('\bf t','fontsize',12,'rotation',0)
subplot(2,2,2)
h522 = ezplot(SHSS522, 't', [-0.1,0.1])
set(h522,'LineWidth',2,'color',[0, 1, 0])
xlabel('\bf x','fontsize',12)
ylabel('\bf t','fontsize',12,'rotation',0)
subplot(2,2,3)
h523 = ezplot(SHSS523, [0,1])
set(h523,'LineWidth',2,'color',[0, 0, 1])
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
axis([0, 1, -0.5, 0.1])
subplot(2,2,4)
plot(xk(2,:),yk(2,:),'k-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf k','fontsize',12)
ylabel('\bf dk','fontsize',12,'rotation',0)
saveas(gcf,'ODE104.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 앞에서 기술한 내용을 확인할 수 있다. 또한 그림 7.7.4가 그려진다. ■

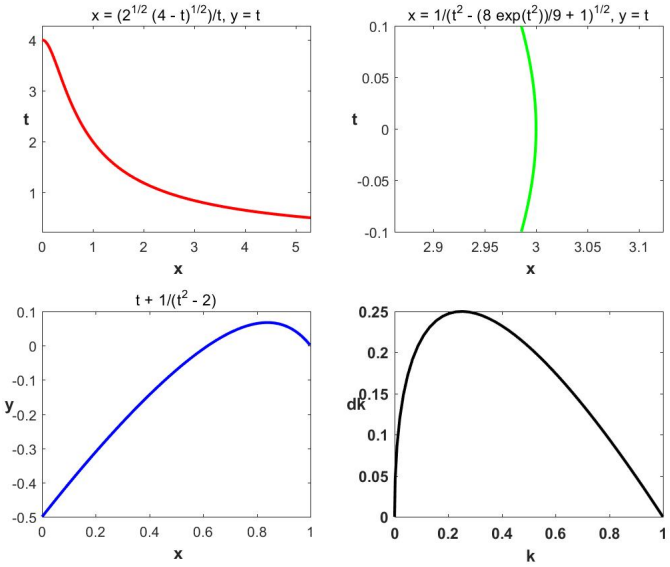


그림 7.7.4. 미분방정식의 해석해 IV

예제 7.7.8 Python을 사용해서 예제 7.7.7을 다시 다루기 위해서, 다음 Python프로그램 ODE104.Py를 실행해 보자.

```
"""
% Filename ODE104.Py
% Ordinary Differential Equation 104
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

# Example 11
x = sy.Function('x')
t = sy.symbols('t')
eq11 = sy.Eq(t**2*x(t)*sy.Derivative(x(t),t) + t*x(t)**2 +1,0)
display(eq11)
sol11 = sy.dsolve(eq11)
display(sol11)
sol11a = sol11[1].rhs
display(sol11a)
## Initial condition
condition0 = sy.Eq(sol11a.subs(t,2),1)   # y(0) = 1
display(condition0)
C1 = sy.symbols("C1")  # generic constants
constant1 = sy.solve([condition0], (C1))
display(constant1)
sol11s = sol11a.subs(C1,8)
print("Solution with initial conditions:")
display(sol11s)
## plt.subplot(2,2,1)
fig = plt.figure()
plt.subplot(2,2,1)
```

```
sol11Lam = sy.lambdify(t, sol11s, modules=['numpy'])
tt = np.linspace(0,4.5, 201)
xx = sol11Lam(tt)
plt.plot(tt,xx,'r-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.ylabel('x',fontsize=12)

# Example 12
x = sy.Function('x')
t = sy.symbols('t')
eq12 = sy.Eq(sy.diff(x(t),t)+ t*x(t)-t**3*x(t)**3,0)
display(eq12)
sol12 = sy.dsolve(eq12)
display(sol12)
sol12r = sy.simplify(sol12.rhs)
display(sol12r)
## Initial condition
condition0 = sy.Eq(sol12r.subs(t,0),3)    # y(0) = -0.5
display(condition0)
C1 = sy.symbols("C1")  # generic constants
constant1 = sy.solve([condition0], (C1))
display(constant1)
sol12s = sy.simplify(sol12.subs(C1,-8/9))
print("Solution with initial conditions:")
display(sol12s)
## plt.subplot(2,2,2)
plt.subplot(2,2,2)
sol12sLam = sy.lambdify(t, sol12s.rhs, modules=['numpy'])
tt = np.linspace(-0.1,0.1,201)
xx = sol12sLam(tt)
plt.plot(tt,xx,'g-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.ylabel('x',fontsize=12)
plt.axis([-0.1, 0.1, 2.98, 3.01])

# Example 13
y = sy.Function('y')
x = sy.symbols('x')
eq13 = sy.Eq(sy.diff(y(x),x)-1+2*x*(y(x)-x)**2,0)
display(eq13)
sol13 = sy.dsolve(eq13).rhs
display(sol13)
## Change variable z = y-x
z = sy.Function('z')
x = sy.symbols('x')
eq13v = sy.Eq(sy.diff(z(x),x)+2*x*z(x)**2,0)
display(eq13v)
sol13v = sy.dsolve(eq13v).rhs
display(sol13v)
## Initial conditions
cnd0 = sy.Eq(sol13v.subs(x,0), -1/2)        # z(0) = -0.5
display(cnd0)
##  Solve for C1,
C1 = sy.symbols("C1")            # generic constants
const13 = sy.solve([cnd0], (C1))
display(const13)
## Substitute back into solution
sol13s = sy.simplify(sol13v.subs(const13))
print("Solution with initial conditions:")
display(sol13s)
## plt.subplot(2,2,3)
plt.subplot(2,2,3)
sol13sLam = sy.lambdify(x, sol13s+x, modules=['numpy'])
```

```
xx = np.linspace(0,1,201)
yy = sol13sLam(xx)
plt.plot(xx,yy,'b-',lw=2.0)
plt.axis([0, 1, -0.5, 0.1])
plt.xlabel('x',fontsize=12), plt.xlabel('y',fontsize=12)

# Example 14
z = sy.Function('z')
t = sy.symbols('t')
eq14 = sy.Eq(sy.diff(z(t),t)+0.5*z(t)-0.5,0)
print("ODE:")
display(eq14)
sol14 = sy.dsolve(eq14, z(t)).rhs   # take only right hand side
sol14 = sy.simplify(sol14)
display(sol14)
display(sy.Eq(z(t), sol14))
# Initial condition
condition0 = sy.Eq(sol14.subs(t,0),0)    # y(0)=0
display(condition0)
C1 = sy.symbols("C1")  # generic constants
constant1 = sy.solve([condition0], (C1))
display(constant1)
sol14s = sy.simplify(sol14.subs(constant1))
print("Solution with initial conditions:")
display(sy.Eq(z(t),sol14s))
## plt.subplot(2,2,4)
plt.subplot(2,2,4)
sol14sLam = sy.lambdify(t, sol14s**2, modules=['numpy'])
sol14sLamD = sy.lambdify(t, sy.diff(sol14s**2), modules=['numpy'])
tt = np.linspace(0, 10, 201)
kk = sol14sLam(tt)
dd = sol14sLamD(tt)
plt.plot(kk,dd,'k-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.xlabel('k',fontsize=12)
plt.axis([0, 1, 0, 0.26 ])

plt.show()
fig.savefig('ODE104Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.7.7의 결과와 같다. ■

다섯 번째 그룹의 첫 번째 미분방정식과 초기조건은 다음과 같다.

$$2y'' - 5y' + 2y = 0\,, \quad y(0) = 1\,, \;\; y(1) = 0 \tag{7.7.100}$$

미분방정식 (7.7.100)에 해당하는 특성방정식은 다음과 같다.

$$2\lambda^2 - 5\lambda + 2 = [2\lambda - 1][\lambda - 2] = 0 \tag{7.7.101}$$

즉, 특성근들은 $\lambda_1 = \frac{1}{2}$과 $\lambda_2 = 2$이다. 따라서, 이 미분방정식의 일반해는 다음과 같다.

$$y = C_1 e^{t/2} + C_2 e^{2t} \tag{7.7.102}$$

식 (7.7.102)와 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$y = \frac{e^{3/2}}{e^{3/2}-1} e^{t/2} - \frac{1}{e^{3/2}-1} e^{2t} \tag{7.7.103}$$

다섯 번째 그룹의 두 번째 미분방정식과 초기조건은 다음과 같다.

$$2y'' - 5y' + 2y = 0\,, \quad y(0) = 1\,, \;\; y'(0) = 0 \tag{7.7.104}$$

식 (7.7.102)에서 알 수 있듯이, 이 미분방정식의 일반해는 다음과 같다.

$$y = C_1 e^{t/2} + C_2 e^{2t} \tag{7.7.105}$$

식 (7.7.105)와 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$y = \frac{4}{3} e^{t/2} - \frac{1}{3} e^{2t} \tag{7.7.106}$$

다섯 번째 그룹의 세 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{y}^2 + y^2 = 1\,, \quad y(0) = 0 \tag{7.7.107}$$

미분방정식 (7.7.107)의 해가 다음과 같다고 가정하자.

$$y_g = a\cos t + b\sin t \tag{7.7.108}$$

다음 식이 성립한다.

$$\dot{y}_g = -a\sin t + b\cos t \tag{7.7.109}$$

식 (7.7.108)와 식 (7.7.109)에서 알 수 있듯이, 다음 식이 성립한다.

$$[\dot{y}_g]^2 + y_g^2 = a^2 + b^2 \tag{7.7.110}$$

다음 식이 성립한다고 가정하자.

$$a^2 + b^2 = 1 \tag{7.7.111}$$

식 (7.7.107)과 식 (7.7.110)에서 알 수 있듯이, 식 (7.7.111)이 성립하면 함수 y_g는 미분방정식 (7.7.107)의 일반해이다. 초기조건 $y(0) = 0$을 만족하는 함수 y_g는 다음 식을 만족한다.

$$b = \dot{y}_g(0) = 1 \tag{7.7.112}$$

식 (7.7.108), 식 (7.7.111)과 식 (7.7.112)에서 알 수 있듯이, 초기값문제 (7.7.107)의 해는 다음과 같다.

$$y_g = \pm \sin t \tag{7.7.113}$$

다섯 번째 그룹의 네 번째 미분방정식과 초기조건은 다음과 같다.

$$y''' - y'' + y' - y = 0\,, \quad y(0) = 2\,, \;\; y'(0) = 0\,, \;\; y''(0) = 1 \tag{7.7.114}$$

미분방정식 (7.7.114)에 해당하는 특성방정식은 다음과 같다.

$$\lambda^3 - \lambda^2 + \lambda^1 - 1 = [\lambda - 1][\lambda - i][\lambda + i] = 0 \tag{7.7.115}$$

즉, 특성근들은 $\lambda_1 = 1$, $\lambda_2 = i$와 $\lambda_3 = -i$이다. 따라서, 이 미분방정식의 일반해는 다음과 같다.

$$y = C_1 e^x + C_2 \cos x + C_3 \sin x \tag{7.7.116}$$

식 (7.7.116)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$y = \frac{3}{2} e^x + \frac{1}{2} \cos x - \frac{3}{2} \sin x \tag{7.7.117}$$

다섯 번째 그룹의 다섯 번째 미분방정식과 초기조건은 다음과 같다.

$$x^3 y''' + 16x^2 y'' + 79xy' + 125y = 0\,, \quad y(1) = 0\,, \;\; y'(1) = 1\,, \;\; y''(1) = 1 \tag{7.7.118}$$

미분방정식 (7.7.118)은 Cauchy-Euler미분방정식이다. 이 미분방정식의 해가 다음과 같다고 가정하자.

$$y = x^r \tag{7.7.119}$$

식 (7.7.119)를 식 (7.7.118)에 대입하면, 다음 식이 성립함을 알 수 있다.

$$r[r-1][r-2]+16r[r-1]+79r+125=0 \tag{7.7.120}$$

방정식 (7.7.120)의 근들은 $r=-5$, $r=-4-3i$, 그리고 $r=-4+3i$이다. 따라서 미분방정식 (7.7.118)의 일반해는 다음과 같다.

$$y=C_1x^{-5}+C_2x^{-4-3i}+C_3x^{-4+3i} \tag{7.7.121}$$

식 (7.7.121)과 초기조건에서 알 수 있듯이, 초기값문제 (7.7.118)의 해는 다음과 같다.

$$y=\frac{1}{x^5}-\frac{\cos(3\ln x)}{x^4}+\frac{2}{3}\frac{\sin(3\ln x)}{x^4} \tag{7.7.122}$$

다음 식들이 성립한다.

$$y'=-\frac{5}{x^6}+6\frac{\cos(3\ln x)}{x^5}+\frac{2}{3}\frac{\sin(3\ln x)}{x^5} \tag{7.7.123}$$

$$y''=\frac{30}{x^7}-\frac{59}{3}\frac{\cos(3\ln x)}{x^6}+29\frac{\sin(3\ln x)}{x^6} \tag{7.7.124}$$

$$y'''=-\frac{210}{x^8}+115\frac{\cos(3\ln x)}{x^7}+205\frac{\sin(3\ln x)}{x^7} \tag{7.7.125}$$

식 (7.7.122)∼식 (7.7.125)로부터 다음 식이 성립함을 알 수 있다.

$$x^3y'''+16x^2y''+79xy'+125y=0\,,\quad y(1)=0\,,\quad y'(1)=1\,,\quad y''(1)=1 \tag{7.7.126}$$

즉, 식 (7.7.122)의 함수 y가 미분방정식 (7.7.118)을 만족한다.

예제 7.7.9 MATLAB을 이용해서 지금까지 다룬 다섯 번째 그룹 문제들을 살펴보기 위해서, 다음 MATLAB프로그램 ODE105.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename ODE105.m
% Higher-order Ordinary Differential Equations
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
% diary ODE105.txt
sol21 = dsolve('2*D2y-5*Dy+2*y','t')
sol22 = dsolve('2*D2y-5*Dy+2*y','y(0)=1','y(1)=0')
sol23 = dsolve('2*D2y-5*Dy+2*y','y(0)=1','Dy(0)=0')
```

```
sol24 = dsolve('(Dy)^2 + y^2 = 1','y(0) = 0','t')
sol25 = expand(sol24)
sol26g = dsolve('D3y-D2y+Dy-y','x')
% sol26 = dsolve('D3y-D2y+Dy-y','y(0)=2','y(pi/4)=1','y(pi/2)=0','x')
sol26 = dsolve('D3y-D2y+Dy-y','y(0)=2','Dy(0)=0','D2y(0)=1','x')
% Cauchy-Euler equation
sol27 = dsolve('x^3*D3y+16*x^2*D2y+79*x*Dy+125*y', ...
               'y(1) = 0','Dy(1) = 1','D2y(1) = 1','x')
y27 = sol27, Dy27 = diff(sol27)
D2y27 = diff(Dy27), D3y27 = diff(D2y27)
syms r
f = r*(r-1)*(r-2) + 16*r*(r-1) + 79*r + 125
fsol = solve(f)
% Plotting
set(gca,'fontsize',11,'fontweigh','bold')
subplot(2,2,1)
h22 = ezplot(sol22, 't', [-1, 1])
set(h22,'LineWidth',2,'color',[1, 0, 0])
title('2*D2y-5*Dy+2*y=0')
xlabel('\bf y','fontsize',12)
ylabel('\bf t','fontsize',12,'rotation',0)
subplot(2,2,2)
h23 = ezplot(sol23, 't', [-1, 1])
set(h23,'LineWidth',2,'color',[0, 1, 0])
title('2*D2y-5*Dy+2*y=0')
xlabel('\bf y','fontsize',12)
ylabel('\bf t','fontsize',12,'rotation',0)
subplot(2,2,3)
h26 = ezplot(sol26, [-6, 4])
set(h26,'LineWidth',2,'color',[0, 0, 1])
title('D3y-D2y+Dy-y=0')
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
subplot(2,2,4)
h27 = ezplot(sol27, [0.5,3])
set(h27,'LineWidth',2,'color',[0.333,0.333,0.334])
title('x^3*D3y+16*x^2*D2y+79*x*Dy+125*y')
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
saveas(gcf,'ODE105.jpg')
% diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 앞에서 기술한 내용을 확인할 수 있다. 또한 그림 7.7.5가 그려진다. ■

예제 7.7.10 Python을 사용해서 예제 7.7.9을 다시 다루기 위해서, 다음 Python프로그램 ODE105.Py를 실행해 보자.

```
"""
% Filename ODE105.Py
% Ordinary Differential Equation 104
% Programmed by CBS
```

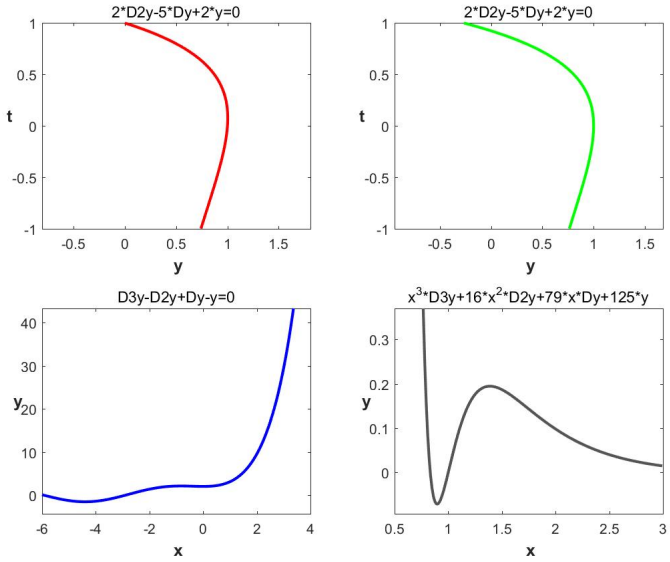


그림 7.7.5. 미분방정식의 해석해 V

```
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

# Example 11
y = sy.Function('y')
t = sy.symbols('t')
eq11 = sy.Eq(2*sy.Derivative(y(t),t,t)-5*sy.Derivative(y(t),t) + 2*y(t),0)
display(eq11)
sol11 = sy.dsolve(eq11).rhs
display(sol11)
## Initial condition
cnd0 = sy.Eq(sol11.subs(t,0),1)          # y(0) = 1
display(cnd0)
cnd1 = sy.Eq(sol11.subs(t,1), 0)         # y(1) = 0
display(cnd1)
##  Solve for C1, C2
C1, C2 = sy.symbols("C1, C2")             # generic constants
const11 = sy.solve([cnd0, cnd1], (C1, C2))
display(const11)
## Substitute back into solution
sol11s = sy.simplify(sol11.subs(const11))
print("Solution with initial conditions:")
display(sy.Eq(y(t),sol11s))
## plt.subplot(2,2,1)
fig = plt.figure()
plt.subplot(2,2,1)
sol11Lam = sy.lambdify(t, sol11s, modules=['numpy'])
tt = np.linspace(-1,1,201)
yy = sol11Lam(tt)
plt.plot(tt,yy,'r-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.ylabel('y',fontsize=12)

```

```
# Example 12: Another Initial condition
condition0 = sy.Eq(sol11.subs(t,0),1)   # y(0) = 1
display(condition0)
condition1 = sy.Eq(sol11.diff(t).subs(t,0), 0)        # y(1) = 0
display(condition1)
##  Solve for C1, C2
C1, C2 = sy.symbols("C1, C2")            # generic constants
constant12 = sy.solve([condition0, condition1], (C1, C2))
display(constant12)
## Substitute back into solution
sol12s = sy.simplify(sol11.subs(constant12))
print("Solution with initial conditions:")
display(sy.Eq(y(t),sol12s))
## plt.subplot(2,2,2)
plt.subplot(2,2,2)
sol12Lam = sy.lambdify(t, sol11s, modules=['numpy'])
tt = np.linspace(-1,1,201)
yy = sol12Lam(tt)
plt.plot(tt,yy,'g-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.ylabel('y',fontsize=12)

# Example 13
y = sy.Function('y')
t = sy.symbols('t')
eq13 = sy.Eq(sy.diff(y(t),t)**2 + y(t)**2 -1,0)
display(eq13)
sol13 = sy.dsolve(eq13)
display(sol13)
sol13r = sy.simplify(sol13.rhs)
display(sol13r)
## Initial condition
condition0 = sy.Eq(sol13r.subs(t,0),0)    # y(0) = 0
display(condition0)
C1 = sy.symbols("C1")  # generic constants
constant1 = sy.solve([condition0], (C1))
display(constant1)
sol13sA = sy.simplify(sol13.subs(C1,np.pi/2))
sol13sB = sy.simplify(sol13.subs(C1,3*np.pi/2))
print("Solution with initial conditions:")
display(sol13sA)
display(sol13sB)

# Example 14
y = sy.Function('y')
x = sy.symbols('x')
eq14 = sy.Eq(sy.diff(y(x),x,3)-sy.diff(y(x),x,2) \
             +sy.diff(y(x),x)-y(x),0)
display(eq14)
sol14 = sy.dsolve(eq14).rhs
display(sol14)
## Initial conditions
condition0 = sy.Eq(sol14.subs(x,0),2)                # y(0) = 2
display(condition0)
condition1 = sy.Eq(sol14.diff(x).subs(x,0), 0)       # y'(0) = 0
display(condition1)
condition2 = sy.Eq(sol14.diff(x,2).subs(x,0), 1)     # y''(0) = 1
display(condition2)
##  Solve for C1, C2, C3
C1, C2, C3 = sy.symbols("C1, C2, C3")            # generic constants
constant123 = sy.solve([condition0, condition1, condition2], \
                       (C1, C2, C3))
```

```
display(constant123)
## Substitute back into solution
sol14s = sy.simplify(sol14.subs(constant123))
print("Solution with initial conditions:")
display(sol14s)
## plt.subplot(2,2,3)
plt.subplot(2,2,3)
sol14sLam = sy.lambdify(x, sol14s, modules=['numpy'])
xx = np.linspace(-6,4,201)
yy = sol14sLam(xx)
plt.plot(xx,yy,'b-',lw=2.0)
plt.axis([-6, 4, -5, 45])
plt.xlabel('x',fontsize=12), plt.xlabel('y',fontsize=12)

# Example 15
y = sy.Function('y')
x = sy.symbols('x')
eq15 = sy.Eq(x**3*sy.diff(y(x),x,3)+16*x**2*sy.diff(y(x),x,2) \
             +79*x*sy.diff(y(x),x)+125*y(x),0)
display(eq15)
sol15 = sy.dsolve(eq15).rhs
display(sol15)
## Initial conditions
condition0 = sy.Eq(sol15.subs(x,1),0)                # y(1) = 0
display(condition0)
condition1 = sy.Eq(sol15.diff(x).subs(x,1), 1)       # y'(1) = 1
display(condition1)
condition2 = sy.Eq(sol15.diff(x,2).subs(x,1), 1)     # y''(1) = 1
display(condition2)
##  Solve for C1, C2, C3
C1, C2, C3 = sy.symbols("C1, C2, C3")            # generic constants
constant123 = sy.solve([condition0, condition1, condition2], \
                       (C1, C2, C3))
display(constant123)
## Substitute back into solution
sol15s = sy.simplify(sol15.subs(constant123))
print("Solution with initial conditions:")
display(sol15s)
## plt.subplot(2,2,3)
plt.subplot(2,2,4)
sol15sLam = sy.lambdify(x, sol15s, modules=['numpy'])
xx = np.linspace(0.5,3,201)
yy = sol15sLam(xx)
plt.plot(xx,yy,'k-',lw=2.0)
plt.axis([0.5,3,-0.1,0.35])
plt.xlabel('x',fontsize=12), plt.xlabel('y',fontsize=12)

plt.show()
fig.savefig('ODE105Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.7.9의 결과와 같다. ■

여섯 번째 그룹으로 Sydsæter & Hammond & Seierstad & Strøm [59]의 미분방정식 문제들을 다루기로 하자.

여섯 번째 그룹의 첫 번째 미분방정식과 초기조건은 다음과 같다.

$$\ddot{x} - 3x = 0\,, \quad x(0) = 1\,, \;\; \dot{x}(0) = 0 \tag{7.7.127}$$

미분방정식 (7.7.127)에 해당하는 특성방정식은 다음과 같다.

$$\lambda^2 - 3 = [\lambda - \sqrt{3}][\lambda + \sqrt{3}] = 0 \tag{7.7.128}$$

즉, 특성근들은 $\lambda_1 = \sqrt{3}$과 $\lambda_2 = -\sqrt{3}$이다. 따라서, 이 미분방정식의 일반해는 다음과 같다.

$$x = C_1 e^{\sqrt{3}t} + C_2 e^{-\sqrt{3}t} \tag{7.7.129}$$

식 (7.7.129)와 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$x = \frac{1}{2} e^{\sqrt{3}t} + \frac{1}{2} e^{-\sqrt{3}t} \tag{7.7.130}$$

여섯 번째 그룹의 두 번째 미분방정식과 초기조건은 다음과 같다.

$$\ddot{x} - 4\dot{x} + 4x = 0\,, \quad x(0) = 1\,, \;\; \dot{x}(0) = 0 \tag{7.7.131}$$

미분방정식 (7.7.131)에 해당하는 특성방정식은 다음과 같다.

$$\lambda^2 - 4\lambda + 4 = [\lambda - 2]^2 = 0 \tag{7.7.132}$$

즉, 특성근은 $\lambda = 2$로서 중복도가 2이다. 따라서, 이 미분방정식의 일반해는 다음과 같다.

$$x = [C_1 + C_2 t]\, e^{2t} \tag{7.7.133}$$

식 (7.7.133)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$x = [1 - 2t]\, e^{2t} \tag{7.7.134}$$

여섯 번째 그룹의 세 번째 미분방정식과 초기조건은 다음과 같다.

$$\ddot{x} - 6\dot{x} + 13x = 0\,, \quad x(0) = 1\,, \;\; x\left(\frac{\pi}{4}\right) = 1 \tag{7.7.135}$$

미분방정식 (7.7.135)에 해당하는 특성방정식은 다음과 같다.

$$\lambda^2 - 6\lambda + 13 = 0 \tag{7.7.136}$$

즉, 특성근들은 $\lambda_1 = 3 + 2i$과 $\lambda_2 = 3 - 2i$이다. 따라서, 이 미분방정식의 일반해는 다음과 같다.

$$x = e^{3t}\left[C_1 \cos 2t + C_2 \sin 2t\right] \tag{7.7.137}$$

식 (7.7.137)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$x = e^{3t}\left[\cos 2t + \exp\left(-\frac{3\pi}{4}\right)\sin 2t\right] \tag{7.7.138}$$

여섯 번째 그룹의 네 번째 미분방정식과 초기조건은 다음과 같다.

$$\ddot{x} - 4\dot{x} + 4x = 2\cos 2t\,, \quad x(0) = 1\,, \;\; \dot{x}(0) = 0 \tag{7.7.139}$$

이 비동치미분방정식(nonhomogeneous ODE)의 특수해(particular solution)가 다음과 같다고 하자.

$$x_p(t) = A\cos 2t + B\sin 2t \tag{7.7.140}$$

다음 식들이 성립한다.

$$\dot{x}_p(t) = 2B\cos 2t - 2A\sin 2t \tag{7.7.141}$$

$$\ddot{x}_p(t) = -4A\cos 2t - 4B\sin 2t \tag{7.7.142}$$

다음 식이 성립해야한다.

$$\ddot{x}_p - 4\dot{x}_p + 4x_p = 2\cos 2t \tag{7.7.143}$$

식 (7.7.140)∼식 (7.7.142)를 식 (7.7.143)에 대입하면, 다음 식을 얻는다.

$$-8B\cos 2t + 8A\sin 2t \equiv 2\cos 2t \tag{7.7.144}$$

따라서, 다음 식들이 성립한다.

$$A = 0, \quad B = -\frac{1}{4} \tag{7.7.145}$$

즉, 특수해는 다음과 같다.

$$x_p(t) = -\frac{1}{4}\sin 2t \tag{7.7.146}$$

초기값문제 (7.7.139) 에 해당하는 동치미분방정식 (homogeneous ODE) 은 다음과 같다.

$$\ddot{x} - 4\dot{x} + 4x = 0 \tag{7.7.147}$$

식 (7.7.133) 에서 알 수 있듯이, 이 동치미분방정식의 해는 다음과 같다.

$$x_h = [c_0 + c_1 t]\, e^{2t} \tag{7.7.148}$$

식 (7.7.146) 와 식 (7.7.148) 에서 알 수 있듯이, 이 초기값문제 (7.7.139) 의 해가 다음과 같다.

$$x = \left[1 - \frac{3t}{2}\right] e^{2t} - \frac{1}{4}\sin 2t \tag{7.7.149}$$

예제 7.7.11 MATLAB을 이용해서 지금까지 다룬 여섯 번째 그룹의 문제들을 살펴보기 위해서, 다음 MATLAB프로그램 ODE106.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename ODE106.m
% Sydsaester, Hammond, Seierstad, and Strom (2008)
% Further Mathematics for Economic Analysis, Chapter 6, No.1
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary ODE106.txt
SHSS611g = dsolve('D2x -3*x=0','t')
SHSS611 = dsolve('D2x-3*x=0','x(0)=1','Dx(0)=0','t')
SHSS612g = dsolve('D2x-4*Dx+4*x=0','t')
SHSS612 = dsolve('D2x-4*Dx+4*x=0','x(0)=1','Dx(0)=0','t')
SHSS613g = dsolve('D2x-6*Dx+13*x','t')
SHSS613 = dsolve('D2x-6*Dx+13*x','x(0)=1','x(pi/4)=1','t')
SHSS614g = dsolve('D2x-4*Dx+4*x=2*cos(2*t)','t')
SHSS614g = simplify(SHSS614g)
SHSS614 = dsolve('D2x-4*Dx+4*x=2*cos(2*t)','x(0)=1','Dx(0)=0','t')
SHSS614 = simplify(SHSS614)
% Plotting
set(gca,'fontsize',11,'fontweigh','bold')
subplot(2,2,1)
h611 = ezplot(SHSS611, 't', [-1,1])
set(h611,'LineWidth',2,'color',[1, 0, 0])
xlabel('\bf x'), ylabel('\bf t','rotation',0)
subplot(2,2,2)
h612 = ezplot(SHSS612, 't', [-6,1])
set(h612,'LineWidth',2,'color',[0, 1, 0])
xlabel('\bf x'), ylabel('\bf t','rotation',0)
subplot(2,2,3)
```

```
h613 = ezplot(SHSS613, [2,9])
set(h613,'LineWidth',2,'color',[0, 0, 1])
xlabel('\bf t'), ylabel('\bf x','rotation',0)
subplot(2,2,4)
h614 = ezplot(SHSS614, [-3,1])
set(h614,'LineWidth',2,'color',[0.333, 0.333, 0.334])
xlabel('\bf t'), ylabel('\bf x','rotation',0)
saveas(gcf,'ODE106.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 앞에서 기술한 내용을 확인할 수 있다. 또한 그림 7.7.6이 그려진다. ■

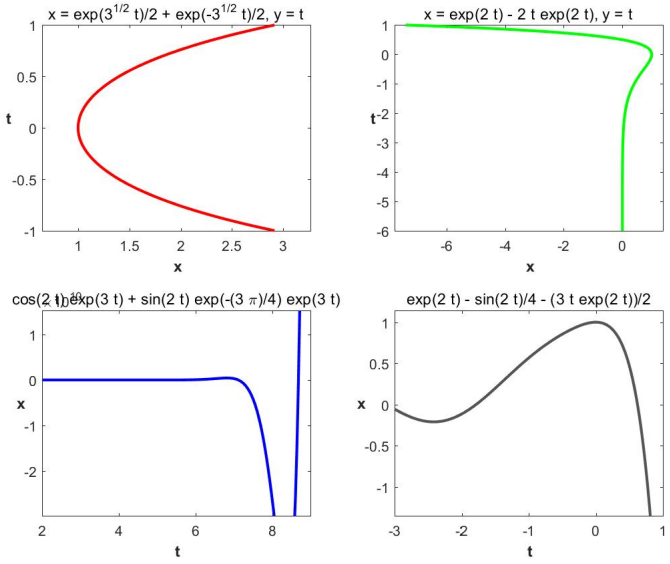


그림 7.7.6. 미분방정식의 해석해 VI

예제 7.7.12 Python을 사용해서 예제 7.7.11을 다시 다루기 위해서, 다음 Python프로그램 ODE106.Py를 실행해 보자.

```
"""
% Filename ODE106.Py
% Ordinary Differential Equation 104
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython
```

```
12
13 # Example 11
14 x = sy.Function('x')
15 t = sy.symbols('t')
16 eq11 = sy.Eq(sy.Derivative(x(t),t,t) - 3*x(t),0)
17 display(eq11)
18 sol11 = sy.dsolve(eq11)
19 display(sol11)
20 sol11r = sy.simplify(sol11.rhs)
21 display(sol11r)
22 ## Initial condition
23 condition0 = sy.Eq(sol11r.subs(t,0),1)                # x(0) = 1
24 display(condition0)
25 condition1 = sy.Eq(sol11r.diff(t).subs(t,0), 0)       # x'(0) = 0
26 display(condition1)
27 ##  Solve for C1, C2
28 C1, C2 = sy.symbols("C1, C2")             # generic constants
29 constant12 = sy.solve([condition0, condition1], (C1, C2))
30 display(constant12)
31 ## Substitute back into solution
32 sol11s = sy.simplify(sol11r.subs(constant12))
33 print("Solution with initial conditions:")
34 display(sy.Eq(x(t),sol11s))
35 ## plt.subplot(2,2,1)
36 fig = plt.figure()
37 plt.subplot(2,2,1)
38 sol11sLam = sy.lambdify(t, sol11s, modules=['numpy'])
39 tt = np.linspace(-1,1,201)
40 xx = sol11sLam(tt)
41 plt.plot(tt,xx,'r-',lw=2.0)
42 plt.xlabel('t',fontsize=12), plt.ylabel('x',fontsize=12)
43
44 # Example 12
45 x = sy.Function('x')
46 t = sy.symbols('t')
47 eq12 = sy.Eq(sy.Derivative(x(t),t,t)-4*sy.Derivative(x(t),t)+4*x(t),0)
48 display(eq12)
49 sol12 = sy.dsolve(eq12)
50 display(sol12)
51 sol12r = sy.simplify(sol12.rhs)
52 display(sol12r)
53 ## Initial condition
54 condition0 = sy.Eq(sol12r.subs(t,0),1)                # x(0) = 1
55 display(condition0)
56 condition1 = sy.Eq(sol12r.diff(t).subs(t,0), 0)       # x'(0) = 0
57 display(condition1)
58 ##  Solve for C1, C2
59 C1, C2 = sy.symbols("C1, C2")             # generic constants
60 constant12 = sy.solve([condition0, condition1], (C1, C2))
61 display(constant12)
62 ## Substitute back into solution
63 sol12s = sy.simplify(sol12r.subs(constant12))
64 print("Solution with initial conditions:")
65 display(sy.Eq(x(t),sol12s))
66 ## plt.subplot(2,2,2)
67 plt.subplot(2,2,2)
68 sol12sLam = sy.lambdify(t, sol12s, modules=['numpy'])
69 tt = np.linspace(-6,1,201)
70 xx = sol12sLam(tt)
71 plt.plot(tt,xx,'g-',lw=2.0)
72 plt.xlabel('t',fontsize=12), plt.ylabel('x',fontsize=12)
```

```
73
74 # Example 13
75 x = sy.Function('x')
76 t = sy.symbols('t')
77 eq13 = sy.Eq(sy.Derivative(x(t),t,t)-6*sy.Derivative(x(t),t)+13*x(t),0)
78 display(eq13)
79 sol13 = sy.dsolve(eq13)
80 display(sol13)
81 sol13r = sy.simplify(sol13.rhs)
82 display(sol13r)
83 ## Initial condition
84 condition0 = sy.Eq(sol13r.subs(t,0),1)              # x(0) = 1
85 display(condition0)
86 condition1 = sy.Eq(sol13r.subs(t,np.pi/4), 1)      # x(pi/4) = 1
87 display(condition1)
88 ##  Solve for C1, C2
89 C1, C2 = sy.symbols("C1, C2")             # generic constants
90 constant12 = sy.solve([condition0, condition1], (C1, C2))
91 display(constant12)
92 ## Substitute back into solution
93 sol13s = sy.simplify(sol13r.subs(constant12))
94 print("Solution with initial conditions:")
95 display(sy.Eq(x(t),sol13s))
96 DoubleCheck = np.exp(-3*np.pi/4)
97 print(DoubleCheck)
98 ## plt.subplot(2,2,3)
99 plt.subplot(2,2,3)
100 sol13sLam = sy.lambdify(t, sol13s, modules=['numpy'])
101 tt = np.linspace(2,9,201)
102 xx = sol13sLam(tt)
103 plt.plot(tt,xx,'b-',lw=2.0)
104 plt.xlabel('t',fontsize=12), plt.ylabel('x',fontsize=12)
105
106 # Example 14
107 x = sy.Function('x')
108 t = sy.symbols('t')
109 eq14 = sy.Eq(sy.Derivative(x(t),t,t)-4*sy.Derivative(x(t),t)+4*x(t) \
110              -2*sy.cos(2*t),0)
111 display(eq14)
112 sol14 = sy.dsolve(eq14)
113 display(sol14)
114 sol14r = sy.simplify(sol14.rhs)
115 display(sol14r)
116 ## Initial condition
117 condition0 = sy.Eq(sol14r.subs(t,0),1)               # x(0) = 1
118 display(condition0)
119 condition1 = sy.Eq(sol14r.diff(t).subs(t,0), 0)      # x'(0) = 0
120 display(condition1)
121 ##  Solve for C1, C2
122 C1, C2 = sy.symbols("C1, C2")             # generic constants
123 constant12 = sy.solve([condition0, condition1], (C1, C2))
124 display(constant12)
125 ## Substitute back into solution
126 sol14s = sy.simplify(sol14r.subs(constant12))
127 print("Solution with initial conditions:")
128 display(sy.Eq(x(t),sol14s))
129 ## plt.subplot(2,2,4)
130 plt.subplot(2,2,4)
131 sol14sLam = sy.lambdify(t, sol14s, modules=['numpy'])
132 tt = np.linspace(-3,1,201)
133 xx = sol14sLam(tt)
```

```
plt.plot(tt,xx,'k-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.ylabel('x',fontsize=12)

plt.show()
fig.savefig('ODE106Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.7.11의 결과와 같다. ■

일곱 번째 그룹의 첫 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = 2y, \quad \dot{y} = x + y\,, \quad x(0) = 1\,, \;\; y(0) = 0 \tag{7.7.150}$$

미분방정식 (7.7.150)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\ddot{y} = \dot{x} + \dot{y} = 2y + \dot{y} \tag{7.7.151}$$

$$\ddot{x} = 2\dot{y} = 2x + 2y = 2x + \dot{x} \tag{7.7.152}$$

즉, x와 y는 다음 미분방정식의 해이다.

$$\dddot{z} - \ddot{z} - 2\dot{z} = 0 \tag{7.7.153}$$

미분방정식 (7.7.153)에 해당하는 특성방정식은 다음과 같다.

$$\lambda^2 - \lambda - 2 = [\lambda - 2][\lambda + 1] = 0 \tag{7.7.154}$$

즉, 특성근들은 $\lambda_1 = 2$과 $\lambda_2 = -1$이다. 따라서, 이 미분방정식의 일반해는 다음과 같다.

$$z = C_1 e^{2t} + C_2 e^{-t} \tag{7.7.155}$$

식 (7.7.155)와 초기조건에서 알 수 있듯이, 다음 식들이 성립한다.

$$x(t) = \frac{2}{3}e^{-t} + \frac{1}{3}e^{2t} \tag{7.7.156}$$

$$y(t) = -\frac{1}{3}e^{-t} + \frac{1}{3}e^{2t} \tag{7.7.157}$$

일곱 번째 그룹의 두 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = 2y + 6, \quad \dot{y} = x + y - 3\,, \quad x(0) = 7\,, \;\; y(0) = -3 \tag{7.7.158}$$

식 $(\dot{x},\, \dot{y}) = (0,\, 0)$을 만족하는 균형점(equilibrium point)은 $(x,\, y) = (6,\, -3)$이다. 따라서 다음과 같은 변수들을 정의하자.

$$z \doteq x - 6, \quad w \doteq y + 3 \tag{7.7.159}$$

초기값문제 (7.7.158)을 다음과 같이 쓸 수 있다.

$$\dot{z} = 2w, \quad \dot{w} = z + w\,, \quad z(0) = 1\,, \;\; w(0) = 0 \tag{7.7.160}$$

식 (7.7.156)과 식 (7.7.157)에서 알 수 있듯이, 다음 식들이 성립한다.

$$z(t) = \frac{2}{3}e^{-t} + \frac{1}{3}e^{2t}, \quad w(t) = -\frac{1}{3}e^{-t} + \frac{1}{3}e^{2t} \tag{7.7.161}$$

식 (7.7.159)와 식 (7.7.161)에서 알 수 있듯이, 다음 식들이 성립한다.

$$x(t) = \frac{2}{3}e^{-t} + \frac{1}{3}e^{2t} + 6, \quad y(t) = -\frac{1}{3}e^{-t} + \frac{1}{3}e^{2t} - 3 \tag{7.7.162}$$

일곱 번째 그룹의 세 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = -2x + y + 2, \quad \dot{y} = -2y + 8\,, \quad x(0) = 2\,, \;\; y(0) = 8 \tag{7.7.163}$$

식 $(\dot{x},\, \dot{y}) = (0,\, 0)$을 만족하는 균형점은 $(x,\, y) = (3,\, 4)$이다. 따라서 다음과 같은 변수들을 정의하자.

$$z \doteq x - 3, \quad w \doteq y - 4 \tag{7.7.164}$$

초기값문제 (7.7.163)을 다음과 같이 쓸 수 있다.

$$\dot{z} = -2z + w, \qquad \dot{w} = -2w \tag{7.7.165}$$

$$z(0) = -1, \qquad w(0) = 4 \tag{7.7.166}$$

식 (7.7.165)과 식 (7.7.166)에서 알 수 있듯이, 다음 식들이 성립한다.

$$w(t) = w(0)e^{-2t} = 4e^{-2t} \tag{7.7.167}$$

식 (7.7.165)과 식 (7.7.167)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\dot{z} = -2z + w = -2z + 4e^{-2t} \tag{7.7.168}$$

따라서 다음 식이 성립한다.

$$e^{2t}\dot{z} + 2e^{2t}z = 4 \tag{7.7.169}$$

즉, 다음 식이 성립한다.

$$\frac{d}{dt}\left[e^{2t}z\right] = 4 \tag{7.7.170}$$

식 (7.7.170)과 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$z = [4t - 1]e^{-2t} \tag{7.7.171}$$

따라서 다음 식이 성립한다.

$$x(t) = [4t - 1]e^{-2t} + 3 \tag{7.7.172}$$

식 (7.7.164)와 식 (7.7.167)에서 알 수 있듯이, 다음 식이 성립한다.

$$y(t) = 4e^{-2t} + 4 \tag{7.7.173}$$

일곱 번째 그룹의 네 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = y\,, \quad \dot{y} = x + \cos t\,, \quad x(0) = 1\,, \;\; y(0) = 1 \tag{7.7.174}$$

다음 식들이 성립한다.

$$\ddot{x} - x = \cos t\,, \quad x(0) = 1 \quad \dot{x}(0) = 1 \tag{7.7.175}$$

이 비동치미분방정식의 특수해가 다음과 같다고 하자.

$$x_p(t) = A\cos t + B\sin t \tag{7.7.176}$$

다음 식들이 성립한다.

$$\dot{x}_p(t) = B\cos t - A\sin t \tag{7.7.177}$$

$$\ddot{x}_p(t) = -A\cos t - B\sin t \tag{7.7.178}$$

다음 식이 성립한다.

$$\ddot{x}_p - x_p = \cos t \tag{7.7.179}$$

식 (7.7.176)∼식 (7.7.178)을 식 (7.7.179)에 대입하면, 다음 식들을 얻는다.

$$A = -\frac{1}{2}, \quad B = 0 \tag{7.7.180}$$

즉, 이 비동치미분방정식의 특수해는 다음과 같다.

$$x_p(t) = -\frac{1}{2}\cos t \tag{7.7.181}$$

초기값문제 (7.7.174)와 식 (7.7.181)에서 알 수 있듯이, 비동치미분방정식의 특수해 $y_p(t)$는 다음과 같다.

$$y_p(t) = \frac{1}{2}\sin t \tag{7.7.182}$$

초기값문제 (7.7.174)에 해당하는 동치미분방정식은 다음과 같다.

$$\ddot{x} - x = 0 \tag{7.7.183}$$

이 동치미분방정식의 해는 다음과 같다.

$$x_h = c_1 e^t - c_2 e^{-t} \tag{7.7.184}$$

따라서, 미분방정식 (7.7.174)의 일반해는 다음과 같다.

$$x_g = c_1 e^t - c_2 e^{-t} - \frac{1}{2}\cos t \tag{7.7.185}$$

초기값문제 (7.7.174)의 초기조건과 식 (7.7.185)에서 알 수 있듯이, 이 초기값문제의 해는 다음과 같다.

$$x = \frac{5}{4}e^t + \frac{1}{4}e^{-t} - \frac{1}{2}\cos t \tag{7.7.186}$$

식 (7.7.186)을 식 (7.7.174)에 대입하면, 다음 식이 성립함을 알 수 있다.

$$y = \frac{5}{4}e^{t} - \frac{1}{4}e^{-t} + \frac{1}{2}\sin t \tag{7.7.187}$$

예제 7.7.13 MATLAB을 이용해서 지금까지 다룬 일곱 번째 그룹의 문제들을 살펴보기 위해서, 다음 MATLAB프로그램 ODE107.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename ODE107.m
% Sydsaester, Hammond, Seierstad, and Strom (2008)
% Further Mathematics for Economic Analysis, Chapter 6, No.2
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary ODE107.txt
SHSS621g = dsolve('Dx=2*y','Dy=x+y','t')
SHSS621g.x, SHSS621g.y
simplify(SHSS621g.x), simplify(SHSS621g.y)
SHSS621 = dsolve('Dx=2*y','Dy=x+y','x(0)=1','y(0)=0','t')
SHSS621.x, SHSS621.y
simplify(SHSS621.x), simplify(SHSS621.y)
SHSS622g = dsolve('Dx=2*y+6','Dy=x+y-3','t')
SHSS622g.x, SHSS622g.y
simplify(SHSS622g.x), simplify(SHSS622g.y)
SHSS622 = dsolve('Dx=2*y+6','Dy=x+y-3','x(0)=7','y(0)=-3','t')
SHSS622.x, SHSS622.y
simplify(SHSS622.x), simplify(SHSS622.y)
SHSS623g = dsolve('Dx=-2*x+y+2','Dy=-2*y+8','t')
SHSS623g.x, SHSS623g.y
simplify(SHSS623g.x), simplify(SHSS623g.y)
SHSS623 = dsolve('Dx=-2*x+y+2','Dy=-2*y+8','x(0)=2','y(0)=8','t')
SHSS623.x, SHSS623.y
simplify(SHSS623.x), simplify(SHSS623.y)
SHSS624g = dsolve('Dx=y','Dy=x+cos(t)','t')
SHSS624g.x, SHSS624g.y
simplify(SHSS624g.x), simplify(SHSS624g.y)
SHSS624 = dsolve('Dx=y','Dy=x+cos(t)','x(0)=1','y(0)=1','t')
SHSS624.x, SHSS624.y
simplify(SHSS624.x), simplify(SHSS624.y)
% Plotting
subplot(2,2,1)
tt = linspace(-1,1,201);
x1 = double(subs(SHSS621.x,'t',tt));
y1 = double(subs(SHSS621.y,'t',tt));
plot(x1,y1,'r-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
subplot(2,2,2)
tt = linspace(-1,1,201);
x2 = double(subs(SHSS622.x,'t',tt));
y2 = double(subs(SHSS622.y,'t',tt));
plot(x2,y2,'g-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
```

```
subplot(2,2,3)
tt = linspace(0,2,201);
x3 = double(subs(SHSS623.x,'t',tt));
y3 = double(subs(SHSS623.y,'t',tt));
plot(x3,y3,'b-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
subplot(2,2,4)
tt = linspace(-pi/2,pi/2,201);
x4 = double(subs(SHSS624.x,'t',tt));
y4 = double(subs(SHSS624.y,'t',tt));
plot(x4,y4,'k-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
axis('equal')
xlabel('\bf x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
saveas(gcf,'ODE107.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 앞에서 기술한 내용을 확인할 수 있다. 또한 그림 7.7.7이 그려진다. ■

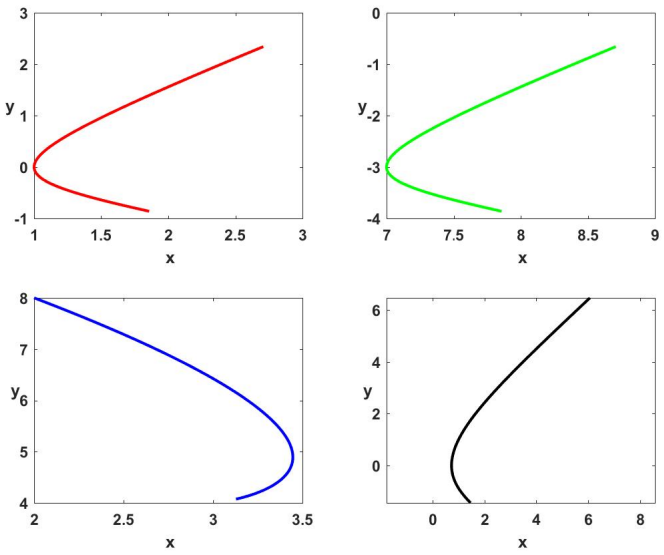


그림 7.7.7. 미분방정식의 해석해 VII

예제 7.7.14 Python을 사용해서 예제 7.7.13을 다시 다루기 위해서, 다음 Python프로그램 ODE107.Py를 실행해 보자.

```
"""
```

```
% Filename ODE107.Py
% Ordinary Differential Equation 104
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

# Example 11
x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t), t), 2*y(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), x(t)+y(t))
display(eq2)
sol11 = sy.dsolve((eq1, eq2))
display(sol11)
## Initial condition
C1, C2 = sy.symbols("C1, C2")            # generic constants
constants = sy.solve((sol11[0].subs(t,0).subs(x(0),1), \
                      sol11[1].subs(t,0).subs(y(0),0)),{C1,C2})
display(constants)
## Substitute back into solution
xsol11 = sy.expand(sol11[0].rhs.subs(constants))
ysol11 = sy.simplify(sol11[1].rhs.subs(constants))
print("Solution with initial conditions:")
display(xsol11)
display(ysol11)
display(sy.Eq(x(t),xsol11))
display(sy.Eq(y(t),ysol11))
## Double checking
display(sy.simplify(eq1.subs(x(t),xsol11).subs(y(t),ysol11)))
display(sy.simplify(eq2.subs(x(t),xsol11).subs(y(t),ysol11)))
display(sy.Eq(sy.simplify(sy.diff(xsol11,t))-2*sy.simplify(ysol11)))
display(sy.Eq(sy.simplify(sy.diff(ysol11,t)) \
              -sy.simplify(xsol11)-sy.simplify(ysol11)))
## plt.subplot(2,2,1)
fig = plt.figure()
plt.subplot(2,2,1)
sol11xLam = sy.lambdify(t,xsol11, modules=['numpy'])
sol11yLam = sy.lambdify(t,ysol11, modules=['numpy'])
tt = np.linspace(-1,1,201)
xx = sol11xLam(tt)
yy = sol11yLam(tt)
plt.plot(xx,yy,'r-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)

# Example 12
x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t), t), 2*y(t)+6)
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), x(t)+y(t)-3)
display(eq2)
sol12 = sy.dsolve((eq1, eq2))
display(sol12)
```

```
## Initial condition
C1, C2 = sy.symbols("C1, C2")             # generic constants
constants = sy.solve((sol12[0].subs(t,0).subs(x(0),7), \
                      sol12[1].subs(t,0).subs(y(0),-3)),{C1,C2})
display(constants)
## Substitute back into solution
xsol12 = sy.expand(sol12[0].rhs.subs(constants))
ysol12 = sy.simplify(sol12[1].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol12))
display(sy.Eq(y(t),ysol12))
## Double checking
display(sy.Eq(sy.simplify(sy.diff(xsol12,t))-2*sy.simplify(ysol12)-6))
display(sy.Eq(sy.simplify(sy.diff(ysol12,t)) \
              -sy.simplify(xsol12)-sy.simplify(ysol12)-3))
## plt.subplot(2,2,2)
plt.subplot(2,2,2)
sol12xLam = sy.lambdify(t,xsol12, modules=['numpy'])
sol12yLam = sy.lambdify(t,ysol12, modules=['numpy'])
tt = np.linspace(-1,1,201)
xx = sol12xLam(tt)
yy = sol12yLam(tt)
plt.plot(xx,yy,'g-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)

# Example 13
x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t), t), -2*x(t)+y(t)+2)
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), -2*y(t)+8)
display(eq2)
sol13 = sy.dsolve((eq1, eq2))
display(sol13)
## Initial condition
C1, C2 = sy.symbols("C1, C2")             # generic constants
constants = sy.solve((sol13[0].subs(t,0).subs(x(0),2), \
                      sol13[1].subs(t,0).subs(y(0),8)),{C1,C2})
display(constants)
## Substitute back into solution
xsol13 = sy.expand(sol13[0].rhs.subs(constants))
ysol13 = sy.simplify(sol13[1].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol13))
display(sy.Eq(y(t),ysol13))
## Double checking
display(sy.Eq(sy.simplify(sy.diff(xsol13,t)) \
              +2*sy.simplify(xsol13)-sy.simplify(ysol13)-2))
display(sy.Eq(sy.simplify(sy.diff(ysol13,t))+2*sy.simplify(xsol13)-8))
## plt.subplot(2,2,3)
plt.subplot(2,2,3)
sol13xLam = sy.lambdify(t,xsol13, modules=['numpy'])
sol13yLam = sy.lambdify(t,ysol13, modules=['numpy'])
tt = np.linspace(0,2,201)
xx = sol13xLam(tt)
yy = sol13yLam(tt)
plt.plot(xx,yy,'b-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)

# Example 14
```

```
x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t), t), y(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), x(t) + sy.cos(t))
display(eq2)
# sol14a = sy.dsolve((eq1, eq2))
# display(sol14a)
eq3 = sy.Eq(sy.diff(x(t),t,2)-x(t)-sy.cos(t),0)
sol14 = sy.dsolve(eq3).rhs
display(sol14)
## Initial conditions
cnd0 = sy.Eq(sol14.subs(t,0), 1)          # x(0) = 1
display(cnd0)
cnd1 = sy.Eq(sol14.diff(t).subs(t,0), 1) # x'(0) = 0
display(cnd1)
##  Solve for C1, C2
C1, C2 = sy.symbols("C1, C2")             # generic constants
const12 = sy.solve([cnd0, cnd1], (C1, C2))
display(const12)
## Substitute back into solution
sol14x = sy.simplify(sol14.subs(const12))
sol14y = sy.diff(sy.simplify(sol14.subs(const12)),t)
print("Solution with initial conditions:")
display(sy.Eq(x(t),sol14x))
display(sy.Eq(y(t),sol14y))
## Double checking
display(sy.Eq(sy.simplify(sy.diff(sol14x,t))-sy.simplify(sol14y)))
display(sy.Eq(sy.simplify(sy.diff(sol14y,t))-sy.simplify(sol14x)-sy.cos(t)))
## plt.subplot(2,2,4)
plt.subplot(2,2,4)
sol14xLam = sy.lambdify(t,sol14x, modules=['numpy'])
sol14yLam = sy.lambdify(t,sol14y, modules=['numpy'])
tt = np.linspace(-np.pi/2,np.pi/2,201)
xx = sol14xLam(tt)
yy = sol14yLam(tt)
plt.plot(xx,yy,'k-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)

plt.show()
fig.savefig('ODE107Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.7.13의 결과와 같다. ■

여덟 번째 그룹의 첫 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = -y, \quad \dot{y} = x\,, \quad x(0) = 1\,, \quad y(0) = 0 \tag{7.7.188}$$

초기값문제 (7.7.188)을 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{7.7.189}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x \\ y \end{bmatrix}\,, \quad A = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{7.7.190}$$

초기값문제 (7.7.189)의 해는 다음과 같다.

$$\boldsymbol{u} = e^{At}\boldsymbol{u}_0 = \sum_{k=0}^{\infty} \frac{1}{k!} A^k \begin{bmatrix} 1 \\ 0 \end{bmatrix} t^k \tag{7.7.191}$$

각 $m(=0,1,\cdots)$에 대해서 다음 식들이 성립한다.

$$A^{4m} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad A^{4m+1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{7.7.192}$$

$$A^{4m+2} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \qquad A^{4m+3} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \tag{7.7.193}$$

식 (7.7.192)와 식 (7.7.193)을 식 (7.7.191)에 대입하면, 다음 식들을 얻는다.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 - \frac{1}{2!}t^2 + \frac{1}{4!}t^4 - \frac{1}{6!}t^6 + \cdots \\ t - \frac{1}{3!}t^3 + \frac{1}{5!}t^5 - \frac{1}{7!}t^7 + \cdots \end{bmatrix} = \begin{bmatrix} \cos t \\ \sin t \end{bmatrix} \tag{7.7.194}$$

여덟 번째 그룹의 두 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = -\frac{3}{2}x + \frac{1}{2}y, \quad \dot{y} = x - y\,, \quad x(0) = 5\,, \;\; y(0) = 4 \tag{7.7.195}$$

초기값문제 (7.7.195)를 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 5 \\ 4 \end{bmatrix} \tag{7.7.196}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x \\ y \end{bmatrix}\,, \quad A = \begin{bmatrix} -\frac{3}{2} & \frac{1}{2} \\ 1 & -1 \end{bmatrix} \tag{7.7.197}$$

초기값문제 (7.7.196)의 해는 다음과 같다.

$$\boldsymbol{u} = e^{At}\boldsymbol{u}_0 \tag{7.7.198}$$

행렬 A를 다음과 같이 대각화할 수 있다.

$$A = P\Lambda P^{-1} \tag{7.7.199}$$

여기서 행렬 Λ, P와 P^{-1}는 각각 다음과 같다.

$$\Lambda = \begin{bmatrix} -2 & 0 \\ 0 & -0.5 \end{bmatrix}, \quad P = \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{2}} & -\frac{2}{\sqrt{5}} \end{bmatrix}, \quad P^{-1} = \begin{bmatrix} -\frac{2\sqrt{2}}{3} & \frac{\sqrt{2}}{3} \\ -\frac{\sqrt{5}}{3} & -\frac{\sqrt{5}}{3} \end{bmatrix} \tag{7.7.200}$$

따라서 초기값문제 (7.7.196)의 해는 다음과 같다.

$$\begin{aligned} \boldsymbol{u} &= Pe^{\Lambda t}P^{-1}\boldsymbol{u}_0 \\ &= \begin{bmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{2}} & -\frac{2}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} e^{-2t} & 1 \\ 1 & e^{-0.5t} \end{bmatrix} \begin{bmatrix} -\frac{2\sqrt{2}}{3} & \frac{\sqrt{2}}{3} \\ -\frac{\sqrt{5}}{3} & -\frac{\sqrt{5}}{3} \end{bmatrix} \begin{bmatrix} 5 \\ 4 \end{bmatrix} \end{aligned} \tag{7.7.201}$$

식 (7.7.201)을 정리하면 다음과 같다.

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 3e^{-0.5t} + 2e^{-2t} \\ 6e^{-0.5t} - 2e^{-2t} \end{bmatrix} \tag{7.7.202}$$

여덟 번째 그룹의 세 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = 4x - 5y, \quad \dot{y} = -2x + y\,, \quad x(0) = 2.9\,, \;\; y(0) = 2.6 \tag{7.7.203}$$

초기값문제 (7.7.203)을 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 2.9 \\ 2.6 \end{bmatrix} \tag{7.7.204}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 각각 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x \\ y \end{bmatrix}\,, \quad A = \begin{bmatrix} 4 & -5 \\ -2 & 1 \end{bmatrix} \tag{7.7.205}$$

초기값문제 (7.7.204)의 해는 다음과 같다.

$$\boldsymbol{u} = e^{At}\boldsymbol{u}_0 \tag{7.7.206}$$

행렬 A의 특성값들은 $\lambda_1 = 6$과 $\lambda_2 = -1$이고 각 특성값에 해당하는 특성벡터는 다음과 같다.

$$\boldsymbol{v}_1 = \begin{bmatrix} -5 \\ 2 \end{bmatrix}\,, \quad \boldsymbol{v}_2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{7.7.207}$$

각 $k(= 1,\, 2)$에 대해서 벡터 $\boldsymbol{u}^{(k)}(t) \doteq \exp(\lambda_k t)\boldsymbol{v}_k$는 다음 식들을 만족한다.

$$\dot{\boldsymbol{u}}^{(k)}(t) = \exp(\lambda_k t)\lambda_k \boldsymbol{v}_k = \exp(\lambda_k t) A\boldsymbol{v}_k = A\boldsymbol{u}^{(k)}(t) \tag{7.7.208}$$

즉, $\boldsymbol{u}^{(i)}(t)$는 식 (7.7.204)의 미분방정식을 만족한다. 만약 $\lambda_1 \neq \lambda_2$이면, $\boldsymbol{u}^{(1)}(t)$와 $\boldsymbol{u}^{(2)}(t)$는 서로 독립이다. 따라서, 이 미분방정식의 일반해는 다음과 같다.

$$\boldsymbol{u}(t) = c_1 \exp(\lambda_1 t)\boldsymbol{v}_1 + c_2 \exp(\lambda_2 t)\boldsymbol{v}_2 = c_1 e^{6t} \begin{bmatrix} -5 \\ 2 \end{bmatrix} + c_2 e^{-t} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{7.7.209}$$

식 (7.7.209) 와 초기조건에서 알 수 있듯이, 다음 식이 성립한다.

$$c_1 \begin{bmatrix} -5 \\ 2 \end{bmatrix} + c_2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2.9 \\ 2.6 \end{bmatrix} \tag{7.7.210}$$

이 연립방정식을 풀면 $c_1 = \frac{-3}{70}$ 이고 $c_2 = \frac{188}{70}$ 이다. 식 (7.7.209) 에서 알 수 있듯이, 이 미분방정식의 해는 다음과 같다.

$$\boldsymbol{u}(t) = \frac{-3}{70} e^{6t} \begin{bmatrix} -5 \\ 2 \end{bmatrix} + \frac{188}{70} e^{-t} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{7.7.211}$$

즉, 다음 식이 성립한다.

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} \frac{3}{14} e^{t} + \frac{94}{35} e^{-t} \\ -\frac{3}{35} e^{t} + \frac{94}{35} e^{-t} \end{bmatrix} \tag{7.7.212}$$

여덟 번째 그룹의 네 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = -2x - 2.5y, \quad \dot{y} = 10x - 2y\,, \quad x(0) = 3\,, \quad y(0) = 3 \tag{7.7.213}$$

초기값문제 (7.7.213) 을 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \tag{7.7.214}$$

여기서 벡터 $\boldsymbol{u}$ 와 행렬 A 는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x \\ y \end{bmatrix}\,, \quad A = \begin{bmatrix} -2 & -2.5 \\ 10 & -2 \end{bmatrix} \tag{7.7.215}$$

초기값문제 (7.7.214) 의 해는 다음과 같다.

$$\boldsymbol{u} = e^{At}\boldsymbol{u}_0 \tag{7.7.216}$$

행렬 A 의 특성값들은 $\lambda_1 = -2 + 5i$ 과 $\lambda_2 = -2 - 5i$ 이고 각각에 해당하는 특성벡터는 다음과

같다.

$$\boldsymbol{v}_1 = -\frac{1}{\sqrt{5}}\begin{bmatrix} i \\ 2 \end{bmatrix}, \quad \boldsymbol{v}_2 = -\frac{1}{\sqrt{5}}\begin{bmatrix} -i \\ 2 \end{bmatrix} \tag{7.7.217}$$

식 (7.7.208)에서 알 수 있듯이, 이 미분방정식의 일반해는 다음과 같다.

$$\boldsymbol{u}(t) = c_1 \exp(\lambda_1 t)\boldsymbol{v}_1 + c_2 \exp(\lambda_2 t)\boldsymbol{v}_2 \tag{7.7.218}$$

다음 식들이 성립한다.

$$\overline{\exp(\lambda_2 t)\boldsymbol{v}_2} = \exp(\overline{\lambda_2} t)\,\overline{\boldsymbol{v}_2} = \exp(\lambda_1 t)\boldsymbol{v}_1 \tag{7.7.219}$$

따라서, 다음 식들이 성립한다.

$$\Re\{\exp(\lambda_1 t)\boldsymbol{v}_1\} = \frac{\exp(\lambda_1 t)\boldsymbol{v}_1 + \overline{\exp(\lambda_1 t)\boldsymbol{v}_1}}{2} \tag{7.7.220}$$

$$\Im\{\exp(\lambda_1 t)\boldsymbol{v}_1\} = \frac{\exp(\lambda_1 t)\boldsymbol{v}_1 - \overline{\exp(\lambda_1 t)\boldsymbol{v}_1}}{2i} \tag{7.7.221}$$

식 (7.7.218)∼식 (7.7.221)에서 알 수 있듯이, 다음과 같이 정의되는 벡터들이 이 미분방정식의 해이다.

$$\boldsymbol{u}_1(t) = [\Re\{\boldsymbol{v}_1\}\cos(\Im\{\lambda_1\}t) - \Im\{\boldsymbol{v}_1\}\sin(\Im\{\lambda_1\}t)]\exp(\Re\{\lambda_1\}t) \tag{7.7.222}$$

$$\boldsymbol{u}_2(t) = [\Re\{\boldsymbol{v}_1\}\sin(\Im\{\lambda_1\}t) + \Im\{\boldsymbol{v}_1\}\cos(\Im\{\lambda_1\}t)]\exp(\Re\{\lambda_1\}t) \tag{7.7.223}$$

여기서 $\Im\{\lambda_1\} \neq 0$이면, $\boldsymbol{u}_1(t)$와 $\boldsymbol{u}_2(t)$는 서로 독립이다. 즉, 이 연립미분방정식의 일반해는 다음과 같다.

$$\boldsymbol{u}(t) = c_3\boldsymbol{u}_1(t) + c_4\boldsymbol{u}_2(t) \tag{7.7.224}$$

따라서 초기값문제 (7.7.213)의 해는 다음과 같다.

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} e^{-2t}[-1.5\sin 5t + 3\cos 5t] \\ e^{-2t}[6\sin 5t + 3\cos 5t] \end{bmatrix} \tag{7.7.225}$$

예제 7.7.15 MATLAB을 이용해서 지금까지 다룬 여덟 번째 그룹의 문제들을 살펴보기 위해서, 다음 MATLAB프로그램 ODE108.m을 실행해보자.

```
% ------------------------------------------------------------------------
% Filename ODE108.m
% Simultaneous Ordinary Differential Equations
% Programmed by CBS
% ------------------------------------------------------------------------
clear, close all
diary ODE108.txt
% Solving ODEs
tt = -2*pi:0.01:2*pi;
sol31 = dsolve('Dx = -y','Dy = x','x(0)=1','y(0)=0')
sol31.x, sol31.y
x31 = double(subs(sol31.x,'t',tt));
y31 = double(subs(sol31.y,'t',tt));
tt = 0:0.01:2*pi;
sol32 = dsolve('Dx = -1.5*x+0.5*y','Dy =x-y','x(0)=5','y(0)=4')
sol32.x, sol32.y
x32 = double(subs(sol32.x,'t',tt));
y32 = double(subs(sol32.y,'t',tt));
sol33 = dsolve('Dx = 4*x - 5*y','Dy = -2*x + y','x(0)=2.9','y(0)=2.6')
tt = -1:0.01:0.5;
sol33.x, sol33.y
x33 = double(subs(sol33.x,'t',tt));
y33 = double(subs(sol33.y,'t',tt));
tt = 0:0.01:10;
sol34 = dsolve('Dx = -2*x - 2.5*y','Dy = 10*x - 2*y','x(0)=3','y(0)=3')
sol34.x, sol34.y
x34 = double(subs(sol34.x,'t',tt));
y34 = double(subs(sol34.y,'t',tt));
% Plotting
subplot(2,2,1)
plot(x31,y31,'r-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
axis equal
subplot(2,2,2)
plot(x32,y32,'g-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
subplot(2,2,3)
plot(x33,y33,'b-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
subplot(2,2,4)
plot(x34,y34,'k-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12)
ylabel('\bf y','fontsize',12,'rotation',0)
saveas(gcf,'ODE108','jpg')
% diary off
% End oh Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 앞에서 기술한 내용을 확인할 수 있다. 또한 그림 7.7.8이 그려진다. ■

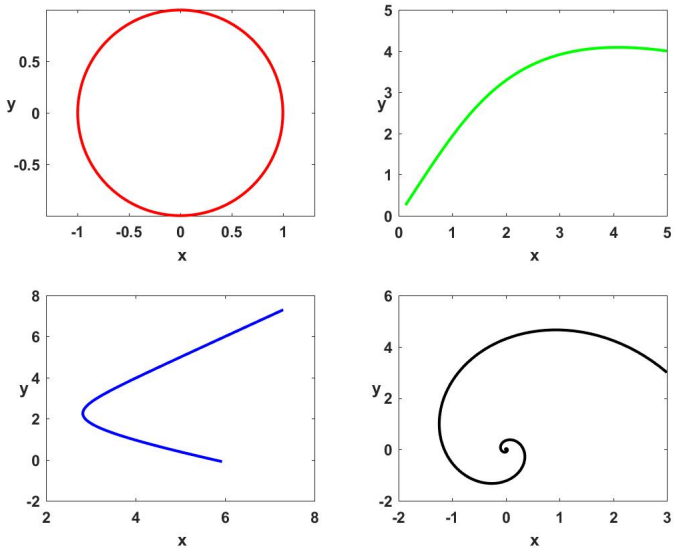


그림 7.7.8. 미분방정식의 해석해 VIII

예제 7.7.16 Python을 사용해서 예제 7.7.15을 다시 다루기 위해서, 다음 Python프로그램 ODE108.Py를 실행해 보자.

```
"""
% Filename ODE108.Py
% Ordinary Differential Equation 104
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

# Example 11
x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t), t), -y(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), x(t))
display(eq2)
sol11 = sy.dsolve((eq1, eq2))
display(sol11)
## Initial condition
C1, C2 = sy.symbols("C1, C2")             # generic constants
constants = sy.solve((sol11[0].subs(t,0).subs(x(0),1), \
                      sol11[1].subs(t,0).subs(y(0),0)),{C1,C2})
display(constants)
## Substitute back into solution
xsol11 = sy.expand(sol11[0].rhs.subs(constants))
ysol11 = sy.simplify(sol11[1].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol11))
```

```
display(sy.Eq(y(t),ysol11))
## Double checking
display(sy.simplify(eq1.subs(x(t),xsol11).subs(y(t),ysol11)))
display(sy.simplify(eq2.subs(x(t),xsol11).subs(y(t),ysol11)))
display(sy.Eq(sy.simplify(sy.diff(xsol11,t))+sy.simplify(ysol11)))
display(sy.Eq(sy.simplify(sy.diff(ysol11,t))-sy.simplify(xsol11)))
## plt.subplot(2,2,1)
fig = plt.figure()
plt.subplot(2,2,1)
sol11xLam = sy.lambdify(t,xsol11, modules=['numpy'])
sol11yLam = sy.lambdify(t,ysol11, modules=['numpy'])
tt = np.linspace(-2*np.pi,2*np.pi,201)
xx = sol11xLam(tt)
yy = sol11yLam(tt)
plt.plot(xx,yy,'r-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)
plt.axis('equal')

# Example 12
x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t), t), -1.5*x(t)+0.5*y(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), x(t)-y(t))
display(eq2)
sol12 = sy.dsolve((eq1, eq2))
display(sol12)
## Initial condition
C1, C2 = sy.symbols("C1, C2")            # generic constants
constants = sy.solve((sol12[0].subs(t,0).subs(x(0),5), \
                      sol12[1].subs(t,0).subs(y(0),4)),{C1,C2})
display(constants)
## Substitute back into solution
xsol12 = sy.expand(sol12[0].rhs.subs(constants))
ysol12 = sy.simplify(sol12[1].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol12))
display(sy.Eq(y(t),ysol12))
## Double checking
display(sy.Eq(sy.simplify(sy.diff(xsol12,t))\
              +1.5*sy.simplify(xsol12)-0.5*sy.simplify(ysol12)))
display(sy.Eq(sy.simplify(sy.diff(ysol12,t)) \
              -sy.simplify(xsol12)+sy.simplify(ysol12)))
## plt.subplot(2,2,2)
plt.subplot(2,2,2)
sol12xLam = sy.lambdify(t,xsol12, modules=['numpy'])
sol12yLam = sy.lambdify(t,ysol12, modules=['numpy'])
tt = np.linspace(0,2*np.pi,201)
xx = sol12xLam(tt)
yy = sol12yLam(tt)
plt.plot(xx,yy,'g-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)

# Example 13
x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t),t), 4*x(t)-5*y(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), -2*x(t)+y(t))
```

```
display(eq2)
sol13 = sy.dsolve((eq1, eq2))
display(sol13)
## Initial condition
C1, C2 = sy.symbols("C1, C2")            # generic constants
constants = sy.solve((sol13[0].subs(t,0).subs(x(0),2.9), \
                      sol13[1].subs(t,0).subs(y(0),2.6)),{C1,C2})
display(constants)
## Substitute back into solution
xsol13 = sy.expand(sol13[0].rhs.subs(constants))
ysol13 = sy.simplify(sol13[1].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol13))
display(sy.Eq(y(t),ysol13))
## Double checking
dumEq1 = sy.diff(xsol13,t)-4*xsol13+5*ysol13; display(dumEq1)
dumEq1 = sy.expand(dumEq1); display(dumEq1)
dumEq2 = sy.diff(ysol13,t)+2*xsol13-ysol13; display(dumEq2)
dumEq2 = sy.expand(dumEq2); display(dumEq2)
## plt.subplot(2,2,3)
plt.subplot(2,2,3)
sol13xLam = sy.lambdify(t,xsol13, modules=['numpy'])
sol13yLam = sy.lambdify(t,ysol13, modules=['numpy'])
tt = np.linspace(-1,0.5,201)
xx = sol13xLam(tt)
yy = sol13yLam(tt)
plt.plot(xx,yy,'b-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)

# Example 14
x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t),t), -2*x(t)-2.5*y(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), 10*x(t)-2*y(t))
display(eq2)
sol14 = sy.dsolve((eq1, eq2))
display(sol14)
## Initial condition
C1, C2 = sy.symbols("C1, C2")            # generic constants
constants = sy.solve((sol14[0].subs(t,0).subs(x(0),3), \
                      sol14[1].subs(t,0).subs(y(0),3)),{C1,C2})
display(constants)
## Substitute back into solution
xsol14 = sy.expand(sol14[0].rhs.subs(constants))
ysol14 = sy.simplify(sol14[1].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol14))
display(sy.Eq(y(t),ysol14))
## Double checking
dumEq1 = sy.diff(xsol14,t)+2*xsol14+2.5*ysol14; display(dumEq1)
dumEq1 = sy.expand(dumEq1); display(dumEq1)
dumEq2 = sy.diff(ysol14,t)-10*xsol14+2*ysol14; display(dumEq2)
dumEq2 = sy.expand(dumEq2); display(dumEq2)
## plt.subplot(2,2,4)
plt.subplot(2,2,4)
sol14xLam = sy.lambdify(t,xsol14, modules=['numpy'])
sol14yLam = sy.lambdify(t,ysol14, modules=['numpy'])
tt = np.linspace(0,10,201)
xx = sol14xLam(tt)
```

```
yy = sol14yLam(tt)
plt.plot(xx,yy,'k-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)

plt.show()
fig.savefig('ODE108Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.7.15의 결과와 같다. ■

아홉 번째 그룹의 첫 번째 미분방정식과 초기조건은 다음과 같다.

$$\frac{d^5x}{dt^5} + 5\frac{d^4x}{dt^4} + 12\frac{d^3x}{dt^3} + 16\frac{d^2x}{dt^2} + 12\frac{dx}{dt} + 4x = 0 \tag{7.7.226}$$

$$x(0) = 1\,, \quad \frac{dx(0)}{dt} = -1\,, \quad \frac{d^2x(0)}{dt^2} = -1\,, \quad \frac{d^3x(0)}{d^3t} = 1\,, \quad \frac{d^4x(0)}{dt^4} = -1 \tag{7.7.227}$$

미분방정식 (7.7.226)에 해당하는 특성방정식은 다음과 같다.

$$\lambda^5 + 5\lambda^4 + 12\lambda^3 + 16\lambda^2 + 12\lambda + 4 = \left[\lambda^2 + 2\lambda + 2\right]^2 [\lambda + 1] = 0 \tag{7.7.228}$$

즉, 특성근들은 다음과 같다.

$$\lambda_1 = -1, \quad \lambda_2 = -1 + i, \quad \lambda_3 = -1 + i, \quad \lambda_4 = -1 - i, \quad \lambda_5 = -1 - i \tag{7.7.229}$$

따라서 미분방정식 (7.7.226)의 일반해는 다음과 같다.

$$x = C_1 e^{-t} + [C_2 + C_3 t]\, e^{-t} \cos t + [C_4 + C_5 t]\, e^{-t} \sin t \tag{7.7.230}$$

식 (7.7.230)과 초기조건 (7.7.227)에서 알 수 있듯이, 이 초기값문제의 해는 다음과 같다.

$$x = -9e^{-t} + [10 + 2t]\, e^{-t} \cos t + [-2 + 4t]\, e^{-t} \sin t \tag{7.7.231}$$

아홉 번째 그룹의 두 번째 미분방정식과 초기조건은 다음과 같다.

$$\frac{d^5x}{dt^5} + 5\frac{d^4x}{dt^4} + 12\frac{d^3x}{dt^3} + 16\frac{d^2x}{dt^2} + 12\frac{dx}{dt} + 4x = t^2 + t - 1 \tag{7.7.232}$$

$$x(0) = 1\,, \quad \frac{dx(0)}{dt} = -1\,, \quad \frac{d^2x(0)}{dt^2} = -1\,, \quad \frac{d^3x(0)}{d^3t} = 1\,, \quad \frac{d^4x(0)}{dt^4} = -1 \tag{7.7.233}$$

비동치미분방정식 (7.7.232)의 특수해가 다음과 같다고 하자.

$$x_p(t) = At^2 + Bt + C \tag{7.7.234}$$

식 (7.7.234)를 비동치미분방정식 (7.7.232)에 대입하면, 다음 식을 얻는다.

$$4At^2 + [24A + 4B]t + [32A + 12B + 40C] \equiv t^2 + t - 1 \tag{7.7.235}$$

따라서 다음 식들이 성립한다.

$$A = \frac{1}{4}, \quad B = -\frac{5}{4}, \quad C = \frac{3}{2} \tag{7.7.236}$$

식 (7.7.230)에서 알 수 있듯이, 비동치미분방정식 (7.7.232)에 해당하는 동치방정식 (7.7.226)의 일반해는 다음과 같다.

$$x_h = C_1 e^{-t} + [C_2 + C_3 t]\, e^{-t} \cos t + [C_4 + C_5 t]\, e^{-t} \sin t \tag{7.7.237}$$

즉, 비동치미분방정식 (7.7.232)의 일반해는 다음과 같다.

$$x_g = C_1 e^{-t} + [C_2 + C_3 t]\, e^{-t} \cos t + [C_4 + C_5 t]\, e^{-t} \sin t + \frac{1}{4}t^2 - \frac{5}{4}t + \frac{3}{2} \tag{7.7.238}$$

식 (7.7.239)과 초기조건 (7.7.233)에서 알 수 있듯이, 이 초기값문제의 해는 다음과 같다.

$$x_g = -9e^{-t} + \left[\frac{17}{2} + \frac{7}{4}t\right] e^{-t} \cos t + \left[-2 + \frac{7}{2}t\right] e^{-t} \sin t + \frac{3}{2} - \frac{5}{4}t + \frac{1}{4}t^2 \tag{7.7.239}$$

아홉 번째 그룹의 세 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = -x + y + z, \quad \dot{y} = x - y + z\,, \quad \dot{z} = x + y + z \tag{7.7.240}$$

$$x(0) = 0\,, \;\; y(0) = -1 \;\; z(0) = 1 \tag{7.7.241}$$

미분방정식 (7.7.240)을 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \tag{7.7.242}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad A = \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{7.7.243}$$

이 초기값문제의 해는 다음과 같다.

$$\boldsymbol{u} = e^{At}\boldsymbol{u}_0 \tag{7.7.244}$$

행렬 A를 다음과 같이 대각화할 수 있다.

$$A = P\Lambda P^{-1} \tag{7.7.245}$$

여기서 행렬들 Λ, P와 P^{-1} 는 각각 다음과 같다.

$$\Lambda = \begin{bmatrix} -2 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \quad P = \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{6}} \\ 0 & -\frac{1}{\sqrt{3}} & \frac{2}{\sqrt{6}} \end{bmatrix}, \quad P^{-1} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \end{bmatrix} \tag{7.7.246}$$

초기조건 (7.7.241)과 식 (7.7.246)에서 알 수 있듯이, 이 미분방정식 (7.7.240)과 초기조건 (7.7.241)로 이루어진 초기값문제의 해는 다음과 같다.

$$\begin{aligned} \boldsymbol{u} &= Pe^{\Lambda t}P^{-1}\boldsymbol{u}_0 \\ &= \begin{bmatrix} -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{6}} \\ 0 & -\frac{1}{\sqrt{3}} & \frac{2}{\sqrt{6}} \end{bmatrix} \begin{bmatrix} e^{-2t} & 1 & 1 \\ 1 & e^{-t} & 1 \\ 1 & 1 & e^{2t} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} \end{aligned} \tag{7.7.247}$$

식 (7.7.247)을 정리하면 다음과 같다.

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} \frac{1}{6}e^{2t} - \frac{2}{3}e^{-t} + \frac{1}{2}e^{-2t} \\ \frac{1}{6}e^{2t} - \frac{2}{3}e^{-t} - \frac{1}{2}e^{-2t} \\ \frac{1}{3}e^{2t} + \frac{2}{3}e^{-t} \end{bmatrix} \tag{7.7.248}$$

아홉 번째 그룹의 네 번째 미분방정식과 초기조건은 다음과 같다.

$$\dot{x} = -x + y + z + t, \quad \dot{y} = x - y + z + t - 1\,, \quad \dot{z} = x + y + z + t^2 + 1 \tag{7.7.249}$$

$$x(0) = 0\,, \;\; y(0) = -1 \;\; z(0) = 1 \tag{7.7.250}$$

미분방정식 (7.7.249)를 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u} + \boldsymbol{b}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 0, & -1, & 1 \end{bmatrix}^t \tag{7.7.251}$$

여기서 벡터들 $\boldsymbol{u}$와 $\boldsymbol{b}$ 그리고 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}\,, \quad \boldsymbol{b} = \begin{bmatrix} t \\ t-1 \\ t^2+1 \end{bmatrix}\,, \quad A = \begin{bmatrix} -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{7.7.252}$$

비동치미분방정식 (7.7.249)의 특수해가 다음과 같다고 하자.

$$x_p(t) = p_{11}t^2 + p_{12}t + p_{13}, \;\; y_p(t) = p_{21}t^2 + p_{22}t + p_{23}, \;\; z_p(t) = p_{31}t^2 + p_{32}t + p_{33} \tag{7.7.253}$$

식 (7.7.253)을 비동치미분방정식 (7.7.249)에 대입하면, 다음 식을 얻는다.

$$x_p(t) = \frac{1}{2}t^2 + \frac{1}{2}, \;\; y_p(t) = \frac{1}{2}t^2 - t + 1, \;\; z_p(t) = t - \frac{1}{2} \tag{7.7.254}$$

식 (7.7.248)에서 알 수 있듯이, 비동치미분방정식 (7.7.249)에 해당하는 동치방정식의 해는 다음과 같다.

$$\begin{bmatrix} x_h(t) \\ y_h(t) \\ z_h(t) \end{bmatrix} = \begin{bmatrix} h_{1,1}e^{2t} + h_{1,2}e^{-t} + h_{1,3}e^{-2t} \\ h_{2,1}e^{2t} + h_{2,2}e^{-t} + h_{2,3}e^{-2t} \\ h_{3,1}e^{2t} + h_{3,2}e^{-t} + h_{3,3}e^{-2t} \end{bmatrix} \tag{7.7.255}$$

식 (7.7.254)과 식 (7.7.255)에서 알 수 있듯이, 이 초기값문제의 일반해는 다음과 같다.

$$\begin{bmatrix} x_g(t) \\ y_g(t) \\ z_g(t) \end{bmatrix} = \begin{bmatrix} h_{1,1}e^{2t} + h_{1,2}e^{-t} + h_{1,3}e^{-2t} + \frac{1}{2}t^2 + \frac{1}{2} \\ h_{2,1}e^{2t} + h_{2,2}e^{-t} + h_{2,3}e^{-2t} + \frac{1}{2}t^2 - t + 1 \\ h_{3,1}e^{2t} + h_{3,2}e^{-t} + h_{3,3}e^{-2t} + t - \frac{1}{2} \end{bmatrix} \tag{7.7.256}$$

식 (7.7.256)과 초기조건 (7.7.250)에서 알 수 있듯이, 이 초기값문제의 해는 다음과 같다.

$$\begin{bmatrix} x_p(t) \\ y_p(t) \\ z_p(t) \end{bmatrix} = \begin{bmatrix} \frac{1}{12}e^{2t} - \frac{4}{3}e^{-t} + \frac{3}{4}e^{-2t} + \frac{1}{2}t^2 + \frac{1}{2} \\ \frac{1}{12}e^{2t} - \frac{4}{3}e^{-t} - \frac{3}{4}e^{-2t} + \frac{1}{2}t^2 - t + 1 \\ \frac{1}{6}e^{2t} + \frac{4}{3}e^{-t} + t - \frac{1}{2} \end{bmatrix} \tag{7.7.257}$$

예제 7.7.17 MATLAB을 이용해서 지금까지 다룬 아홉 번째 그룹의 문제들을 살펴보기 위해서, 다음 MATLAB프로그램 ODE109.m을 실행해보자.

```
% ------------------------------------------------------------------
% Filename ODE109.m
% Sydsaester, Hammond, Seierstad, and Strom (2008)
% Further Mathematics for Economic Analysis, Chapter 7
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary ODE109.txt
SHSS711g = dsolve('D5x+5.0*D4x+12.0*D3x+16.0*D2x+12.0*Dx+4*x=0','t')
SHSS711 = dsolve('D5x+5.0*D4x+12.0*D3x+16.0*D2x+12.0*Dx+4*x=0', ...
            'x(0)=1','Dx(0)=-1','D2x(0)=-1','D3x(0)=1','D4x(0)=-1','t')
SHSS712g = dsolve('D5x+5.0*D4x+12.0*D3x+16.0*D2x+12.0*Dx+4*x=t^2+t-1','t')
SHSS712 = dsolve('D5x+5.0*D4x+12.0*D3x+16.0*D2x+12.0*Dx+4*x=t^2+t-1', ...
            'x(0)=1','Dx(0)=-1','D2x(0)=-1','D3x(0)=1','D4x(0)=-1','t')
SHSS713g = dsolve('Dx=-x+y+z','Dy=x-y+z','Dz=x+y+z','t')
SHSS713g.x, SHSS713g.y, SHSS713g.z
SHSS713 = dsolve('Dx=-x+y+z','Dy=x-y+z','Dz=x+y+z', ...
                  'x(0)=0','y(0)=-1','z(0)=1','t')
SHSS713.x, SHSS713.y, SHSS713.z
SHSS714g = dsolve('Dx=-x+y+z+t','Dy=x-y+z+t-1','Dz=x+y+z+t^2 +1','t')
SHSS714g.x, SHSS714g.y, SHSS714g.z
Eqn714 = 'Dx=-x+y+z+t,Dy=x-y+z-t,Dz=x+y+z-t^2'
Init714 = 'x(0)=0, y(0)=-1, z(0)=1'
SHSS714 = dsolve(Eqn714,Init714,'t')
x4 = SHSS714.x; y4 = SHSS714.y; z4 = SHSS714.z;
x4 = simplify(x4), y4 = simplify(y4), z4 =simplify(z4)
% Plotting
subplot(2,2,1)
h711 = ezplot(SHSS711, 't', [-1.2,1.2])
set(h711,'LineWidth',2,'color',[1, 0, 0])
set(gca,'fontsize',11,'fontweigh','bold')
title('Homogeneous')
xlabel('\bf x','fontsize',12)
ylabel('\bf t','fontsize',12,'rotation',0)
subplot(2,2,2)
h712 = ezplot(SHSS712, 't', [-1.2,1.2])
set(h712,'LineWidth',2,'color',[0, 1, 0])
set(gca,'fontsize',11,'fontweigh','bold')
title('Non-Homogeneous')
xlabel('\bf x','fontsize',12)
ylabel('\bf t','fontsize',12,'rotation',0)
subplot(2,2,3)
tt = linspace(-2,2,201);
x3 = double(subs(SHSS713.x,'t',tt));
y3 = double(subs(SHSS713.y,'t',tt));
```

```
z3 = double(subs(SHSS713.z,'t',tt));
plot3(x3,y3,z3,'b-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12), ylabel('y','fontsize',12)
zlabel('z','fontsize',12)
subplot(2,2,4)
t = linspace(-2,2,201);
xx4 = eval(vectorize(x4));
yy4 = eval(vectorize(y4));
zz4 = eval(vectorize(z4));
plot(t,xx4,'r--',t,yy4,'g-',t,zz4,'b-.','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf t','fontsize',12)
legend('\bf x','\bf y','\bf z','location','SE')
saveas(gcf,'ODE109.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 앞에서 기술한 내용을 확인할 수 있다. 또한 그림 7.7.9가 그려진다. ■

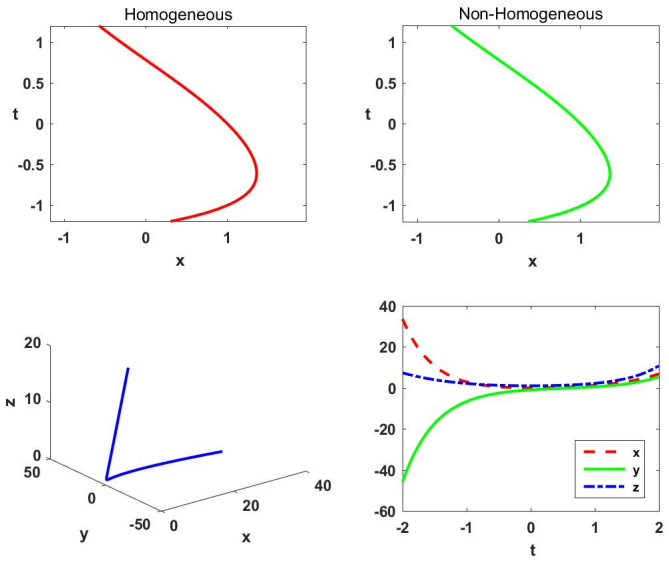


그림 7.7.9. 미분방정식의 해석해 IX

예제 7.7.18 Python을 사용해서 예제 7.7.17을 다시 다루기 위해서, 다음 Python프로그램 ODE109.Py를 실행해 보자.

```
"""
% Filename ODE109.Py
% Ordinary Differential Equation 109
% Programmed by CBS
```

```
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

# Example 11
x = sy.Function('x')
t = sy.symbols('t')
de0 = x(t)
de1 = sy.diff(x(t),t,1)
de2 = sy.diff(x(t),t,2)
de3 = sy.diff(x(t),t,3)
de4 = sy.diff(x(t),t,4)
de5 = sy.diff(x(t),t,5)
eq1 = sy.Eq(de5+5*de4+12*de3+16*de2+12*de1+4*de0,0)
display(eq1)
sol11 = sy.dsolve(eq1).rhs
display(sol11)
## Initial condition
C1, C2, C3, C4, C5 = sy.symbols("C1, C2, C3, C4, C5") # generic constants
cnd0 = sy.Eq(sol11.subs(t,0), 1)          # x(0)=1
display(cnd0)
cnd1 = sy.Eq(sol11.diff(t).subs(t,0), -1) # x'(0)=-1
display(cnd1)
cnd2 = sy.Eq(sol11.diff(t,2).subs(t,0), -1) # x''(0)=-1
display(cnd2)
cnd3 = sy.Eq(sol11.diff(t,3).subs(t,0), 1) # x'''(0)=1
display(cnd3)
cnd4 = sy.Eq(sol11.diff(t,4).subs(t,0), -1) # x'(0)=-1
display(cnd4)
##  Solve for C1, C2, C3 C4, C5
const = sy.solve([cnd0,cnd1,cnd2,cnd3,cnd4],(C1,C2,C3,C4,C5))
display(const)
## Substitute back into solution
sol11s = sy.expand(sol11.subs(const))
print("Solution with initial conditions:")
display(sy.Eq(x(t),sol11s))
## Double checking
dum0 = sy.simplify(sol11s)
display(dum0)
dum1 = sy.simplify(sol11s.diff(t))
display(dum1)
dum2 = sy.simplify(sol11s.diff(t,2))
display(dum2)
dum3 = sy.simplify(sol11s.diff(t,3))
display(dum3)
dum4 = sy.simplify(sol11s.diff(t,4))
display(dum4)
dum5 = sy.simplify(sol11s.diff(t,5))
display(dum5)
dumTot = sy.simplify(dum5+5*dum4+12*dum3+16*dum2+12*dum1+4*dum0)
display(dumTot)
## plt.subplot(2,2,1)
fig = plt.figure()
plt.subplot(2,2,1)
sol11sLam = sy.lambdify(t,sol11s, modules=['numpy'])
tt = np.linspace(-1.2,1.2,201)
xx = sol11sLam(tt)
```

```
plt.plot(tt,xx,'r-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.ylabel('x',fontsize=12)

# Example 12
x = sy.Function('x')
t = sy.symbols('t')
de0 = x(t)
de1 = sy.diff(x(t),t,1)
de2 = sy.diff(x(t),t,2)
de3 = sy.diff(x(t),t,3)
de4 = sy.diff(x(t),t,4)
de5 = sy.diff(x(t),t,5)
eq1 = sy.Eq(de5+5*de4+12*de3+16*de2+12*de1+4*de0,t**2+t-1)
display(eq1)
sol12 = sy.dsolve(eq1).rhs
display(sol12)
## Initial condition
C1, C2, C3, C4, C5 = sy.symbols("C1, C2, C3, C4, C5") # generic constants
cnd0 = sy.Eq(sol12.subs(t,0), 1)          # x(0)=1
display(cnd0)
cnd1 = sy.Eq(sol12.diff(t).subs(t,0), -1) # x'(0)=-1
display(cnd1)
cnd2 = sy.Eq(sol12.diff(t,2).subs(t,0), -1) # x''(0)=-1
display(cnd2)
cnd3 = sy.Eq(sol12.diff(t,3).subs(t,0), 1) # x'''(0)=1
display(cnd3)
cnd4 = sy.Eq(sol12.diff(t,4).subs(t,0), -1) # x'(0)=-1
display(cnd4)
##  Solve for C1, C2, C3 C4, C5
const = sy.solve([cnd0,cnd1,cnd2,cnd3,cnd4],(C1,C2,C3,C4,C5))
display(const)
## Substitute back into solution
sol12s = sy.expand(sol12.subs(const))
print("Solution with initial conditions:")
display(sy.Eq(x(t),sol12s))
## Double checking
dum0 = sy.simplify(sol12s)
display(dum0)
dum1 = sy.simplify(sol12s.diff(t))
display(dum1)
dum2 = sy.simplify(sol12s.diff(t,2))
display(dum2)
dum3 = sy.simplify(sol12s.diff(t,3))
display(dum3)
dum4 = sy.simplify(sol12s.diff(t,4))
display(dum4)
dum5 = sy.simplify(sol12s.diff(t,5))
display(dum5)
dumTot = sy.simplify(dum5+5*dum4+12*dum3+16*dum2+12*dum1+4*dum0 \
                     -t**2-t+1)
display(dumTot)
## plt.subplot(2,2,2)
plt.subplot(2,2,2)
sol11sLam = sy.lambdify(t,sol12s, modules=['numpy'])
tt = np.linspace(-1.2,1.2,201)
xx = sol11sLam(tt)
plt.plot(tt,xx,'g-',lw=2.0)
plt.xlabel('t',fontsize=12), plt.ylabel('x',fontsize=12)
plt.show()

# Example 13
```

```
x = sy.Function('x')
y = sy.Function('y')
z = sy.Function('z')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t),t), -x(t)+y(t)+z(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), x(t)-y(t)+z(t))
display(eq2)
eq3 = sy.Eq(sy.diff(z(t),t), x(t)+y(t)+z(t))
display(eq3)
sol13 = sy.dsolve((eq1, eq2, eq3))
display(sol13)
## Initial condition
C1, C2, C3 = sy.symbols("C1, C2, C3")            # generic constants
cond0 = sy.Eq(sol13[0].rhs.subs(t,0),0)
display(cond0)
cond1 = sy.Eq(sol13[1].rhs.subs(t,0),-1)
display(cond1)
cond2 = sy.Eq(sol13[2].rhs.subs(t,0),1)
display(cond2)
constants = sy.solve((cond0,cond1,cond2),{C1,C2,C3})
display(constants)
## Substitute back into solution
xsol13 = sy.expand(sol13[0].rhs.subs(constants))
ysol13 = sy.expand(sol13[1].rhs.subs(constants))
zsol13 = sy.expand(sol13[2].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol13))
display(sy.Eq(y(t),ysol13))
display(sy.Eq(z(t),zsol13))
## Double checking
dumx = sy.diff(xsol13,t)+xsol13-ysol13-zsol13;
print(dumx)
dumy = sy.diff(ysol13,t)-xsol13+ysol13-zsol13;
print(dumy)
dumz = sy.diff(zsol13,t)-xsol13-ysol13-zsol13;
print(dumz)
## plt.subplot(2,2,3)
sol13xLam = sy.lambdify(t,xsol13, modules=['numpy'])
sol13yLam = sy.lambdify(t,ysol13, modules=['numpy'])
sol13zLam = sy.lambdify(t,zsol13, modules=['numpy'])
tt = np.linspace(-2,2,201)
xx = sol13xLam(tt)
yy = sol13yLam(tt)
zz = sol13zLam(tt)
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(223)
ax = Axes3D(fig)
ax.scatter(xx,yy,zz,color='b',marker='*')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
fig.savefig('ODE109Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.7.17의 결과와 같다. ■

제7.8절 미분방정식과 Jordan표준형

7.8.1 Jordan표준형

제7.7절에서는 행렬 A의 특성벡터들이 서로 독립인 경우를 주로 다루었다. 이 절에서는 행렬 A의 특성벡터들이 서로 독립이 아닌 경우를 다루기로 하자.

정의 7.8.1: 행렬의 대각화

정방행렬들 A와 B에 대해서 다음 식을 만족하는 정칙행렬 P가 존재하면, 이 행렬들은 상사적(similar)이라 한다.

$$A = PBP^{-1}$$

이러한 상사성을 식 $A \sim B$로 표기한다. 만약 정방행렬 A가 대각행렬과 상사적이면, 행렬 A는 대각화가능이라 한다.

다음 정리의 증명은 선형대수학 책을 참조하라.

정리 7.8.1

정방행렬 A가 대각화가능하기 위한 필요충분조건은 행렬 A의 특성벡터들이 선형독립이다.

정방행렬 A의 서로 다른 특성값들 λ_1과 λ_2에 대한 특성벡터들을 각각 $\boldsymbol{x}_1$과 $\boldsymbol{x}_2$라 하자. 식 $c_1\boldsymbol{x}_1 + c_2\boldsymbol{x}_2 = \boldsymbol{0}$가 성립한다고 하자. 만약 식 $c_1 \neq 0$이면, 식 $\boldsymbol{x}_1 = -c_2\lambda_2/[c_1\lambda_1]\boldsymbol{x}_2$이다. 이는 $\boldsymbol{x}_2$가 특성값 λ_1에 대한 특성벡터라는 뜻이다. 즉, 다음 식들이 성립한다.

$$\lambda_2\boldsymbol{x}_2 = A\boldsymbol{x}_2 = \lambda_1\boldsymbol{x}_2. \tag{7.8.1}$$

즉, 다음 식이 성립한다.

$$[\lambda_1 - \lambda_2]\,\boldsymbol{x}_2 = \boldsymbol{0}. \tag{7.8.2}$$

즉, $c_1 = c_2 = 0$이 성립한다. 따라서, 특성벡터들 $\boldsymbol{x}_1$과 $\boldsymbol{x}_2$는 독립이다. 따라서, 행렬 A의 특성근이 서로 다르면, 행렬 A는 대각화가능하다. 즉, 대각화가 불가능한 행렬 A는 중복도

(multiplicity)가 2 이상인 특성근을 적어도 한 개 갖는다. 이 절에서는 이러한 경우를 다루기로 하자.

차원이 $n \times n$ 행렬 A의 특성방정식 $f(\lambda) = 0$의 특성값 λ_1의 중복도가 k_1이라 하자. 즉, 특성함수 $f(\lambda)$를 $[\lambda - \lambda_1]^{k_1}$로 나눌 수 있으나 $[\lambda - \lambda_1]^{k_1+1}$로 나눌 수 없다고 하자. 이 경우에 특성값 λ_1의 대수중복도가 k_1이라 한다. 특성값 λ_1에 해당하는 특성벡터들이 이루는 부분공간의 차원을 기하중복도(geometric multiplicity)라 하고 s_1으로 표기하자. 다음 식이 성립한다.

$$s_1 = n - \operatorname{rank}(A - \lambda_1 I) \tag{7.8.3}$$

어떤 특성값의 대수중복도가 2 이상이라고 해서 대각화가 반드시 불가능한 것은 아니다. 다음 예제들을 살펴보자.

예제 7.8.1 다음과 같은 행렬 A를 살펴보자.

$$A = \begin{bmatrix} 5 & 4 & 2 \\ 4 & 5 & 2 \\ 2 & 2 & 2 \end{bmatrix} \tag{1}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 5-\lambda & 4 & 2 \\ 4 & 5-\lambda & 2 \\ 2 & 2 & 2-\lambda \end{vmatrix}$$

$$= -\lambda^3 + 12\lambda^2 - 21\lambda + 10 = -[\lambda - 1]^2[\lambda - 10] = 0 \tag{2}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = 1\,, \quad \lambda_2 = 1\,, \quad \lambda_3 = 10 \tag{3}$$

특성값 $\lambda_1 = 1$에 대해서 다음 식들이 성립한다.

$$\begin{bmatrix} 5-\lambda & 4 & 2 \\ 4 & 5-\lambda & 2 \\ 2 & 2 & 2-\lambda \end{bmatrix}_{\lambda=1} = \begin{bmatrix} 4 & 4 & 2 \\ 4 & 4 & 2 \\ 2 & 2 & 1 \end{bmatrix} \sim \begin{bmatrix} 2 & 2 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{4}$$

따라서 이에 해당하는 특성벡터들은 다음과 같다.

$$\boldsymbol{v}_1 = \begin{bmatrix} -1, & 0, & 2 \end{bmatrix}^t, \quad \boldsymbol{v}_2 = \begin{bmatrix} -1, & 1, & 0 \end{bmatrix}^t \tag{5}$$

이 특성값 $\lambda_1 = 1$의 기하중복도는 다음과 같다.

$$s_1 = n - \operatorname{rank}(A - \lambda_1 I) = 3 - 1 = 2. \tag{6}$$

특성값 $\lambda_2 = 10$에 대해서 다음 식들이 성립한다.

$$\begin{bmatrix} 5-\lambda & 4 & 2 \\ 4 & 5-\lambda & 2 \\ 2 & 2 & 2-\lambda \end{bmatrix}_{\lambda=10} = \begin{bmatrix} -5 & 4 & 2 \\ 4 & -5 & 2 \\ 2 & 2 & -8 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & -2 \\ 0 & 0 & 0 \end{bmatrix} \tag{7}$$

따라서 이에 해당하는 특성벡터는 다음과 같다.

$$\boldsymbol{v}_3 = \begin{bmatrix} 2, & 2, & 1 \end{bmatrix}^t \tag{8}$$

이 특성값 $\lambda_2 = 10$의 기하중복도는 다음과 같다.

$$s_1 = n - \operatorname{rank}(A - \lambda_2 I) = 3 - 2 = 1. \tag{9}$$

다음과 같이 대각행렬 D와 행렬 P를 정의하자.

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 10 \end{bmatrix}, \quad P = \begin{bmatrix} \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3 \end{bmatrix} = \begin{bmatrix} -1 & -1 & 2 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix} \tag{10}$$

다음 식이 성립한다.

$$\det(P) = -9 \tag{9}$$

즉, 특성벡터들 $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$는 선형독립이다. 따라서 대각화가 가능하다. 다음 식이 성립함을 쉽게 확인할 수 있다.

$$AP = \begin{bmatrix} -1 & -1 & 20 \\ 0 & 1 & 20 \\ 2 & 0 & 10 \end{bmatrix} = PD \tag{12}$$

즉, 행렬 A를 다음과 같이 대각화할 수 있다.

$$A = PDP^{-1} \tag{13}$$

■

예제 7.8.2 다음과 같은 행렬 A를 살펴보자.

$$A = \begin{bmatrix} 2 & 0 & 1 & -3 \\ 0 & 2 & 10 & 4 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \tag{1}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 2-\lambda & 0 & 1 & -3 \\ 0 & 2-\lambda & 10 & 4 \\ 0 & 0 & 2-\lambda & 0 \\ 0 & 0 & 0 & 3-\lambda \end{vmatrix}$$

$$= \lambda^4 - 9\lambda^3 + 30\lambda^2 - 44\lambda + 24 = -[\lambda - 2]^3[\lambda - 3] = 0 \tag{2}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = 2\,, \quad \lambda_2 = 2\,, \quad \lambda_3 = 2\,, \quad \lambda_4 = 3 \tag{3}$$

예제 7.8.1과 같은 방법을 적용하면, 특성값 $\lambda_1 = 2$에 대한 특성벡터들이 다음과 같음을

알 수 있다.

$$\boldsymbol{v}_1 = \begin{bmatrix} 1, & 0, & 0, & 0 \end{bmatrix}^t, \quad \boldsymbol{v}_2 = \begin{bmatrix} 1, & 10, & 0, & 0 \end{bmatrix}^t \tag{4}$$

또한, 특성값 $\lambda_4 = 3$에 대한 특성벡터들이 다음과 같음을 알 수 있다.

$$\boldsymbol{v}_4 = \begin{bmatrix} -3, & 4, & 0, & 1 \end{bmatrix}^t \tag{5}$$

즉, 행렬 A는 선형독립인 특성벡터들을 3개만 갖는다. 따라서 행렬 A는 대각화가 가능하지 않다. 반면 다음 식이 성립한다.

$$\begin{bmatrix} 2 & 0 & 1 & -3 \\ 0 & 2 & 10 & 4 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \sim \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \tag{6}$$

행렬 A를 대각화할 수는 없다. 그러나 행렬 A가 대각행렬과 비슷한 행렬과 상사적임을 알 수 있다. ■

정의 7.8.2: Jordan블록

특성값 λ에 대한 Jordan블록은 대각원소는 λ이고, 대각원소 바로 위의 원소는 1이고, 나머지 원소는 0인 상삼각행렬이다. 즉, 특성값 λ에 대한 Jordan블록은 다음과 같다.

$$J = \begin{bmatrix} \lambda & 1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda & 1 & \cdots & 0 & 0 \\ 0 & 0 & \lambda & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda & 1 \\ 0 & 0 & 0 & 0 & 0 & \lambda \end{bmatrix}$$

정의 7.8.3: Jordan행렬

각 대각부행렬(diagonal submatrix)이 Jordan블록이고 나머지 원소들이 0인 행렬을 Jordan행렬이라 하고 Jordan표준형(canonical form)을 취한다고 한다.

예제 7.8.3 행렬 A가 대수중복도가 2인 특성근 λ_1, 대수중복도가 1인 특성근 λ_2, 그리고 대수중복도가 3인 특성근 λ_3를 갖는다고 하자. 만약 각 특성근의 기하중복도가 1이면, 이에 해당하는 Jordan행렬 J는 다음과 같다.

$$J = \begin{bmatrix} \lambda_1 & 1 & 0 & 0 & 0 & 0 \\ 0 & \lambda_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_3 & 1 & 0 \\ 0 & 0 & 0 & 0 & \lambda_3 & 1 \\ 0 & 0 & 0 & 0 & 0 & \lambda_3 \end{bmatrix}$$

■

정리 7.8.2: Jordan분해

임의의 정방행렬은 Jordan행렬과 상사적이다. 즉, 임의의 정방행렬 A에 대해서 다음 식을 만족하는 Jordan행렬 J와 비정칙행렬 H가 존재한다.

$$A = HJH^{-1}$$

특성값 λ에 해당하는 특성벡터가 $\boldsymbol{v}_1$이고 Jordan블록의 차수가 $k \times k$라고 하면, 다음 식이 성립한다.

$$[A - \lambda I]\,\boldsymbol{v}_1 = \boldsymbol{0} \tag{7.8.4}$$

일반화특성벡터(generalized eigenvector) $\boldsymbol{v}_m (\neq \boldsymbol{0})$을 다음과 같이 정의하자.

$$[A - \lambda I]\,\boldsymbol{v}_m = \boldsymbol{v}_{m-1}, \quad (m = 2, 3, \cdots, k) \tag{7.8.5}$$

식 (7.8.5)에서 알 수 있듯이, 다음 식들이 성립한다.

$$[A-\lambda I]^m \boldsymbol{v}_m = \mathbf{0}, \quad (m=1,2,\cdots,k) \tag{7.8.6}$$

다음 식을 만족하는 스칼라들 $c_1, c_2, \cdots, c_k$를 살펴보자.

$$c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_k\boldsymbol{v}_k = \mathbf{0} \tag{7.8.7}$$

식 (7.8.6)과 식 (7.8.7)에서 알 수 있듯이, 다음 식들이 성립한다.

$$[A-\lambda I]^{k-1}\left[c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_k\boldsymbol{v}_k\right] = c_k\boldsymbol{v}_1 = \mathbf{0} \tag{7.8.8}$$

즉, $c_k = 0$이다. 또한, 다음 식들이 성립함을 증명할 수 있다.

$$[A-\lambda I]^{k-2}\left[c_1\boldsymbol{v}_1 + c_2\boldsymbol{v}_2 + \cdots + c_k\boldsymbol{v}_k\right] = c_{k-1}\boldsymbol{v}_1 + c_k\boldsymbol{v}_2 = \mathbf{0} \tag{7.8.9}$$

즉, $c_{k-1} = 0$이다. 같은 방법을 적용해서, 다음 식들이 성립함을 증명할 수 있다.

$$c_1 = c_2 = \cdots = c_k = 0 \tag{7.8.10}$$

따라서, 벡터들의 집합 $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_k\}$은 서로 독립이다. 이 집합을 길이가 k인 Jordan 체인이라 부른다.

예제 7.8.4 예제 7.8.2를 다시 생각해보자. 행렬 A는 다음과 같다.

$$A = \begin{bmatrix} 2 & 0 & 1 & -3 \\ 0 & 2 & 10 & 4 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \tag{1}$$

행렬 A의 특성방정식이 $[\lambda-2]^3[\lambda-3]=0$이므로 특성값 $\lambda_1 = 2$에 대한 특성벡터들이 3개 존재해야 할 것 같다. 그러나, 이 특성값에 대한 특성벡터들은 다음과 같다.

$$\boldsymbol{v}_1 = \begin{bmatrix} 1, & 0, & 0, & 0 \end{bmatrix}^t, \quad \boldsymbol{v}_2 = \begin{bmatrix} 1, & 10, & 0, & 0 \end{bmatrix}^t \tag{2}$$

다음 방정식을 만족하는 벡터 $\boldsymbol{v}_3$를 구해보자.

$$\begin{bmatrix} 0 & 0 & 1 & -3 \\ 0 & 0 & 10 & 4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \boldsymbol{v}_3 = [A - 2I]\boldsymbol{v}_3 = \boldsymbol{v}_2 = \begin{bmatrix} 1 \\ 10 \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

벡터 $\boldsymbol{v}_3 = \begin{bmatrix} 0, & 0, & 1, & 0 \end{bmatrix}^t$는 식 (3)을 만족한다. 또한 다음 식들을 만족한다.

$$[A - 2I]\boldsymbol{v}_3 = \boldsymbol{v}_2 \neq \mathbf{0}, \quad [A - 2I]^2\boldsymbol{v}_3 = [A - 2I]\boldsymbol{v}_2 = \mathbf{0} \tag{4}$$

따라서, $\boldsymbol{v}_3$는 특성값 $\lambda_1 = 2$에 대한 랭크가 2인 일반화특성벡터이다.

다음 행렬을 정의하자.

$$H = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, \boldsymbol{v}_4] = \begin{bmatrix} 1 & 1 & 0 & -3 \\ 0 & 10 & 0 & 4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

여기서 $\boldsymbol{v}_4$는 특성값 $\lambda_4 = 3$에 해당하는 특성벡터이다. 다음 식이 성립한다.

$$\det(H) = 1 \tag{6}$$

즉, 벡터들 $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, \boldsymbol{v}_4$는 선형독립이다. 따라서, 벡터들의 집합 $\{\boldsymbol{v}_2, \boldsymbol{v}_3\}$는 서로 독립이며, 길이가 $k = 2$인 Jordan체인이다. 따라서, Jordan분해는 다음과 같다.

$$A = H \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} H^{-1} \tag{7}$$

■

차원이 2×2인 행렬의 Jordan분해에 대해서 살펴보자.

예제 7.8.5 다음과 같은 행렬 A를 살펴보자.

$$A = \begin{bmatrix} 0 & 1 \\ -1 & -2 \end{bmatrix} \tag{1}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 0-\lambda & 1 \\ -1 & -2-\lambda \end{vmatrix} = \lambda^2 + 2\lambda + 1 = [\lambda + 1]^2 = 0 \tag{2}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = -1\,, \quad \lambda_2 = -1 \tag{3}$$

특성값 $\lambda_1 = -1$의 대수중복도는 $k_1 = 1$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{11},\, v_{21}\,]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} 0+1 & 1 \\ -1 & -2+1 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4}$$

즉, 다음 식이 성립한다.

$$v_{11} + v_{21} = 0 \tag{5}$$

따라서 $\boldsymbol{v}_1 = [1, -1]^t$은 특성값 $\lambda_1 = -1$에 해당하는 특성벡터이고, 특성값 $\lambda_1 = -1$의 기하중복도는 다음과 같다.

$$s_1 = n - \operatorname{rank}(A - \lambda_1 I) = 2 - 1 = 1 \tag{6}$$

따라서, Jordan행렬은 다음과 같다.

$$J = \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix} \tag{7}$$

행렬 A의 Jordan표준형이 다음과 같다고 하자.

$$A = HJH^{-1} \tag{8}$$

행렬 H 를 다음과 같이 표기하자.

$$H = [\boldsymbol{v}_1, \boldsymbol{v}_2] \tag{9}$$

즉, $\boldsymbol{v}_1$ 과 $\boldsymbol{v}_2$ 가 행렬 H 의 열벡터들이라고 하자. 다음 식들이 성립한다.

$$A\,[\boldsymbol{v}_1, \boldsymbol{v}_2] = AH = HJ = [\boldsymbol{v}_1, \boldsymbol{v}_2] \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix} \tag{10}$$

따라서, 다음 식들이 성립한다.

$$A\boldsymbol{v}_1 = -\boldsymbol{v}_1 \tag{11}$$

$$A\boldsymbol{v}_2 = \boldsymbol{v}_1 - \boldsymbol{v}_2 \tag{12}$$

식 (11) 과 식 (12) 에서 알 수 있듯이, 다음 식들이 성립한다.

$$[A + I]^2 \boldsymbol{v}_2 = [A + I]\,\boldsymbol{v}_1 = \mathbf{0} \tag{13}$$

식 (13) 을 만족하는 일반화특성벡터 $\boldsymbol{v}_2$ 를 다음과 같이 선택하자.

$$\boldsymbol{v}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{14}$$

식 (14) 를 식 (12) 에 대입하면, 다음 식들이 성립함을 알 수 있다.

$$\boldsymbol{v}_1 = [A + I]\boldsymbol{v}_2 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{15}$$

다음 식이 성립함을 확인할 수 있다.

$$[A + I]\boldsymbol{v}_1 = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{16}$$

즉, 식 (11) 이 성립한다. 식 (14) 와 식 (15) 에서 알 수 있듯이, 행렬 H 는 다음과 같다.

$$H = [\boldsymbol{v}_1, \boldsymbol{v}_2] = \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix} \tag{17}$$

식 $\det(H) = 1$이 성립하므로, 벡터들 $\boldsymbol{v}_1$과 $\boldsymbol{v}_2$는 선형독립이다. ■

차원이 3×3인 행렬의 Jordan분해에 대해서 살펴보자.

예제 7.8.6 다음과 같은 행렬 A를 살펴보자.

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 2 & 0 \\ 1 & -2 & 3 \end{bmatrix} \tag{1}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 1-\lambda & 2 & -1 \\ 0 & 2-\lambda & 0 \\ 1 & -2 & 3-\lambda \end{vmatrix}$$

$$= -\lambda^3 + 6\lambda^2 - 12\lambda + 8 = -[\lambda - 2]^3 = 0 \tag{2}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = 2\,, \quad \lambda_2 = 2, \quad \lambda_3 = 2 \tag{3}$$

특성값 $\lambda_1 = 2$의 대수중복도는 $k_1 = 3$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{11},\, v_{21},\, v_{31}\,]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} 1-2 & 2 & -1 \\ 0 & 2-2 & 0 \\ 1 & -2 & 3-2 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \\ v_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

즉, 다음 식이 성립한다.

$$v_{11} - 2v_{21} + v_{31} = 0 \tag{5}$$

특성값 $\lambda_1 = 2$에 해당하며 서로 독립인 특성벡터들이 2개 존재하므로 기하중복도는 2이다. 이를 다음 식에서 확인할 수 있다.

$$s_1 = n - \operatorname{rank}(A - \lambda_1 I) = 3 - 1 = 2 \tag{6}$$

따라서, Jordan 행렬은 다음과 같다.

$$J = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \tag{7}$$

행렬 A의 Jordan 표준형이 다음과 같다고 하자.

$$A = HJH^{-1} \tag{8}$$

다음과 같이 벡터들 $\boldsymbol{v}_1$, $\boldsymbol{v}_2$와 $\boldsymbol{v}_3$를 정의하자.

$$H = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3] \tag{9}$$

다음 식들이 성립한다.

$$A\,[\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\,] = AH = HJ = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\,] \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \tag{10}$$

즉, 다음 식들이 성립한다.

$$A\boldsymbol{v}_1 = 2\boldsymbol{v}_1 \tag{11}$$

$$A\boldsymbol{v}_2 = \boldsymbol{v}_1 + 2\boldsymbol{v}_2 \tag{12}$$

$$A\boldsymbol{v}_3 = 2\boldsymbol{v}_3 \tag{13}$$

식 (11)과 식 (12)에서 알 수 있듯이, 다음 식들이 성립한다.

$$[A - 2I]^2\,\boldsymbol{v}_2 = [A - 2I]\,\boldsymbol{v}_1 = \boldsymbol{0} \tag{14}$$

식 (14)를 만족하는 벡터 $\boldsymbol{v}_2$를 다음과 같이 선택하자.

$$\boldsymbol{v}_2 = \begin{bmatrix} 1, & 0, & 0 \end{bmatrix}^t \tag{15}$$

식 (15)를 식 (12)에 대입하면, 다음 식들이 성립함을 알 수 있다.

$$\boldsymbol{v}_1 = [A - 2I]\boldsymbol{v}_2 = \begin{bmatrix} -1 & 2 & -1 \\ 0 & 0 & 0 \\ 1 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{16}$$

명백하게 벡터 $\boldsymbol{v}_1$과 벡터 $\boldsymbol{v}_2$는 독립이다. 식 (4)로 표현되는 평면 상의 벡터로서 벡터들 $\boldsymbol{v}_1$ 그리고 $\boldsymbol{v}_2$에 수직인 벡터 $\boldsymbol{v}_3$를 다음과 같이 선택하자.

$$\boldsymbol{v}_3 = \begin{bmatrix} 0, & 1, & 0 \end{bmatrix}^t \tag{17}$$

행렬 H는 다음과 같다.

$$H = \begin{bmatrix} -1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{18}$$

식 $\det(H) = 1$이 성립하므로, 벡터들 $\boldsymbol{v}_1$, $\boldsymbol{v}_2$와 $\boldsymbol{v}_3$는 선형독립이다. ■

예제 7.8.7 다음과 같은 행렬 A를 살펴보자.

$$A = \begin{bmatrix} 2 & -1 & 1 \\ 1 & 5 & -2 \\ 0 & 1 & 2 \end{bmatrix} \tag{1}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 2-\lambda & -1 & 1 \\ 1 & 5-\lambda & -2 \\ 0 & 1 & 2-\lambda \end{vmatrix} = -[\lambda - 3]^3 = 0 \tag{2}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = 3\,, \quad \lambda_2 = 3, \quad \lambda_3 = 3 \tag{3}$$

특성값 $\lambda_1 = 3$의 대수중복도는 $k_1 = 3$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{11},\, v_{21},\, v_{31}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} -1 & -1 & 1 \\ 1 & 2 & -2 \\ 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \\ v_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

즉, 다음 식이 성립한다.

$$v_{11} = 0\,, \quad 0v_{11} - v_{21} + v_{31} = 0 \tag{5}$$

따라서 특성값 $\lambda_1 = 3$에 해당하며 서로 독립인 특성벡터가 단 1개 존재하고, 또한 특성값 $\lambda_1 = 3$의 기하중복도는 다음과 같다.

$$s_1 = n - \operatorname{rank}(A - \lambda_1 I) = 3 - 2 = 1 \tag{6}$$

따라서, Jordan행렬은 다음과 같다.

$$J = \begin{bmatrix} 3 & 1 & 0 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{bmatrix} \tag{7}$$

행렬 A의 Jordan표준형이 다음과 같다고 하자.

$$A = HJH^{-1} \tag{8}$$

다음과 같이 벡터들 $\boldsymbol{v}_1$, $\boldsymbol{v}_2$와 $\boldsymbol{v}_3$를 정의하자.

$$H = [\boldsymbol{v}_1,\, \boldsymbol{v}_2,\, \boldsymbol{v}_3] \tag{9}$$

다음 식들이 성립한다.

$$A\,[\boldsymbol{v}_1,\, \boldsymbol{v}_2,\, \boldsymbol{v}_3\,] = AH = HJ = [\boldsymbol{v}_1,\, \boldsymbol{v}_2,\, \boldsymbol{v}_3\,] \begin{bmatrix} 3 & 1 & 0 \\ 0 & 3 & 1 \\ 0 & 0 & 3 \end{bmatrix} \tag{10}$$

따라서, 다음 식들이 성립한다.

$$A\boldsymbol{v}_1 = 3\boldsymbol{v}_1 \tag{11}$$

$$A\boldsymbol{v}_2 = \boldsymbol{v}_1 + 3\boldsymbol{v}_2 \tag{12}$$

$$A\boldsymbol{v}_3 = \boldsymbol{v}_2 + 3\boldsymbol{v}_3 \tag{13}$$

식 (11)∼식 (13)에서 알 수 있듯이, 다음 식들이 성립한다.

$$[A-3I]^3\boldsymbol{v}_3 = [A-3I]^2\boldsymbol{v}_2 = [A-3I]\boldsymbol{v}_1 = \mathbf{0} \tag{14}$$

즉, 다음 식들이 성립한다.

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}\boldsymbol{v}_3 = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}\boldsymbol{v}_2 = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 2 & -2 \\ 0 & 1 & -1 \end{bmatrix}\boldsymbol{v}_1 = \mathbf{0} \tag{15}$$

예제 7.8.6에서와는 달리, 벡터 $\boldsymbol{v}_1$, 벡터 $\boldsymbol{v}_2$ 그리고 벡터 $\boldsymbol{v}_3$ 순으로 이 벡터들을 구하기로 하자. 식 (15)를 만족하는 특성벡터 $\boldsymbol{v}_1$을 다음과 같이 선택하자.

$$\boldsymbol{v}_1 = \begin{bmatrix} 0, & 1, & 1 \end{bmatrix}^t \tag{16}$$

식 (12)에서 알 수 있듯이, 벡터 $\boldsymbol{v}_2 = [v_{12},\, v_{22},\, v_{32}]^t$는 다음 식들을 만족한다.

$$[A-3I]\boldsymbol{v}_2 = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 2 & -2 \\ 0 & 1 & -1 \end{bmatrix}\begin{bmatrix} v_{12} \\ v_{22} \\ v_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \boldsymbol{v}_1 \tag{17}$$

다음과 같이 $\boldsymbol{v}_2$를 선택하자.

$$\boldsymbol{v}_2 = \begin{bmatrix} -1, & 1, & 0 \end{bmatrix}^t \tag{18}$$

식 (13)에서 알 수 있듯이, 다음 식들이 성립한다.

$$[A-3I]\boldsymbol{v}_3 = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 2 & -2 \\ 0 & 1 & -1 \end{bmatrix}\begin{bmatrix} v_{13} \\ v_{23} \\ v_{33} \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} = \boldsymbol{v}_2 \tag{19}$$

다음과 같이 $\boldsymbol{v}_3$를 선택하자.

$$\boldsymbol{v}_3 = \begin{bmatrix} 1, & 0, & 0 \end{bmatrix}^t \tag{20}$$

따라서, 행렬 H는 다음과 같다.

$$H = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3] = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{21}$$

식 $\det(H) = -1$이 성립하므로, 벡터들 $\boldsymbol{v}_1$, $\boldsymbol{v}_2$와 $\boldsymbol{v}_3$는 선형독립이다. ■

예제 7.8.8 다음과 같은 행렬 A의 Jordan표준형을 구해보자.

$$A = \begin{bmatrix} -1 & -18 & -7 \\ 1 & -13 & -4 \\ -1 & 25 & 8 \end{bmatrix} \tag{1}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = -[\lambda + 2]^3 = 0 \tag{2}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = -2\,, \quad \lambda_2 = -2\,, \quad \lambda_3 = -2 \tag{3}$$

다음 식이 성립한다.

$$A + 2I = \begin{bmatrix} 1 & -18 & -7 \\ 1 & -11 & -4 \\ -1 & 25 & 10 \end{bmatrix} \tag{4}$$

특성값 $\lambda_1 = -2$에 대한 특성벡터가 다음과 같음을 알 수 있다.

$$\boldsymbol{v}_{11} = \begin{bmatrix} -5, & -3, & 7 \end{bmatrix}^t \tag{5}$$

이 $\boldsymbol{v}_{11}$은 행렬 $A+2I$의 커넬의 기저(basis)이다. 또한, 랭크가 1인 일반화특성벡터는 $\boldsymbol{v}_{11}$의 스칼라 배이다. 행렬 $A+2I$의 랭크는 2이다. 따라서, $\lambda_1=-2$의 기하중복도는 다음과 같다.

$$s_1 = n - \operatorname{rank}(A-\lambda_1 I) = 3-2 = 1 \tag{6}$$

다음 식이 성립한다.

$$[A+2I]^2 = \begin{bmatrix} -10 & 5 & -5 \\ -6 & 3 & -3 \\ 14 & -7 & 7 \end{bmatrix} \tag{7}$$

다음 벡터들을 정의하자.

$$\boldsymbol{v}_{21} = \begin{bmatrix} -1 & 0 & 2 \end{bmatrix}^t, \quad \boldsymbol{v}_{22} = \begin{bmatrix} 1 & 2 & 0 \end{bmatrix}^t \tag{8}$$

벡터들의 집합 $\{\boldsymbol{v}_{21}, \boldsymbol{v}_{22}\}$는 행렬 $[A+2I]^2$의 커넬의 기저이다. 랭크가 2인 일반화특성벡터는 이 기저에 의해서 생성된 부분공간 $\mathrm{Sp}(\{\boldsymbol{v}_{21}, \boldsymbol{v}_{22}\})$에 속한다. 특성벡터 $\boldsymbol{v}_{11}$도 행렬 $[A+2I]^2$의 커넬에 속한다.

다음 식이 성립한다.

$$[A+2I]^3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{9}$$

다음 벡터들을 정의하자.

$$\boldsymbol{v}_{31} = \boldsymbol{e}_1, \quad \boldsymbol{v}_{32} = \boldsymbol{e}_2, \quad \boldsymbol{v}_{33} = \boldsymbol{e}_3 \tag{10}$$

벡터들의 집합 $\{\boldsymbol{v}_{31}, \boldsymbol{v}_{32}, \boldsymbol{v}_{33}\}$은 행렬 $[A+2I]^3$의 커넬의 기저이다. 랭크가 3인 일반화특성벡터는 이 기저에 의해서 생성된 부분공간 $\mathrm{Sp}(\{\boldsymbol{v}_{31}, \boldsymbol{v}_{32}, \boldsymbol{v}_{33}\})$에 속한다. 또한 벡터들 $\boldsymbol{v}_{11}, \boldsymbol{v}_{21}, \boldsymbol{v}_{22}$ 이 행렬 $[A+2I]^3$의 커넬에 속하는 것은 당연하다.

앞에서 언급했듯이, 벡터 $\boldsymbol{x}_3=\boldsymbol{e}_1$은 랭크가 3인 일반화특성벡터이다. 이 벡터 $\boldsymbol{x}_3$를 씨앗(seed)로 하는 Jordan체인을 구축하자. 다음 식들이 성립한다.

$$\boldsymbol{x}_2 = [A+2I]\,\boldsymbol{x}_3 = \begin{bmatrix} 1, & 1, & -1 \end{bmatrix}^t \tag{11}$$

$$\boldsymbol{x}_1 = [A+2I_4]\,\boldsymbol{x}_2 = [A+2I_4]^2\,\boldsymbol{x}_3 = 2\begin{bmatrix} -5, & -3, & 7 \end{bmatrix}^t \tag{12}$$

식 (5)와 식 (12)에서 알 수 있듯이, $\boldsymbol{x}_1$은 특성값 $\lambda_1 = -2$에 대한 특성벡터이다. 또한 벡터들의 집합 $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$는 서로 독립이며, 길이가 3인 Jordan체인이다. 다음과 같이 행렬 H와 행렬 J를 정의하자.

$$H = \begin{bmatrix} -10 & 1 & 1 \\ -6 & 1 & 0 \\ 14 & -1 & 0 \end{bmatrix}, \quad J = \begin{bmatrix} -2 & 1 & 0 \\ 0 & -2 & 1 \\ 0 & 0 & -2 \end{bmatrix} \tag{13}$$

결론적으로 행렬 A를 다음과 같은 Jordan표준형으로 나타낼 수 있다.

$$A = HJH^{-1} \tag{14}$$

■

차원이 4×4인 행렬의 Jordan분해에 대해서 살펴보자.

예제 7.8.9 다음과 같은 행렬 A를 살펴보자.

$$A = \begin{bmatrix} 4 & -4 & -11 & 11 \\ 7 & -16 & -48 & 46 \\ -6 & 16 & 43 & -38 \\ -3 & 9 & 23 & -19 \end{bmatrix} \tag{1}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 4-\lambda & -4 & -11 & 11 \\ 7 & -16-\lambda & -48 & 46 \\ -6 & 16 & 43-\lambda & -38 \\ -3 & 9 & 23 & -19-\lambda \end{vmatrix}$$

$$= \lambda^4 - 12\lambda^3 + 54\lambda^2 - 108\lambda + 81 = -[\lambda - 3]^4 = 0 \tag{2}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = 3, \quad \lambda_2 = 3, \quad \lambda_3 = 3, \quad \lambda_4 = 3 \tag{3}$$

특성값 $\lambda_1 = 3$에 대한 특성벡터 $\boldsymbol{v}_{11}$은 다음 식들을 만족한다.

$$[A-3I]\boldsymbol{v}_{11} = \begin{bmatrix} 1 & -4 & -11 & 11 \\ 7 & -19 & -48 & 46 \\ -6 & 16 & 40 & -38 \\ -3 & 9 & 23 & -22 \end{bmatrix} \boldsymbol{v}_{11} = \mathbf{0} \tag{4}$$

식 (4)를 만족하는 다음과 같은 특성벡터 $\boldsymbol{v}_{11}$을 선택하자.

$$\boldsymbol{v}_{11} = \begin{bmatrix} -1, & -3, & 2, & 1 \end{bmatrix}^t \tag{5}$$

즉, $\boldsymbol{v}_{11}$은 행렬 $A-3I$의 커넬(kernel 또는 null space)의 기저(basis)이다. 행렬 $A-3I$의 랭크는 3이다. 따라서, $\lambda_1 = 3$의 기하중복도는 다음과 같다.

$$s_1 = n - \operatorname{rank}(A-\lambda_1 I) = 4-3 = 1 \tag{6}$$

다음 식이 성립한다.

$$[A-3I]^2 = \begin{bmatrix} 6 & -5 & -6 & 3 \\ 24 & -21 & -27 & 15 \\ -20 & 18 & 24 & -14 \\ -12 & 11 & 15 & -9 \end{bmatrix} \tag{7}$$

다음 벡터들을 정의하자.

$$\boldsymbol{v}_{21} = \begin{bmatrix} 2, & 3, & 0, & 1 \end{bmatrix}^t, \quad \boldsymbol{v}_{22} = \begin{bmatrix} -3, & -6, & 2, & 0 \end{bmatrix}^t \tag{8}$$

벡터들의 집합 $\{\boldsymbol{v}_{21}, \boldsymbol{v}_{22}\}$는 행렬 $[A-3I]^2$의 커넬의 기저이다. 랭크가 2인 일반화특성벡터는 이 기저에 의해서 생성된 부분공간 $\operatorname{Sp}(\{\boldsymbol{v}_{21}, \boldsymbol{v}_{22}\})$에 속한다. 식 $\boldsymbol{v}_{11} = \boldsymbol{v}_{21} + \boldsymbol{v}_{22}$에서 알 수 있듯이, 특성벡터 $\boldsymbol{v}_{11}$은 행렬 $[A-3I]^2$의 커넬에 속한다.

다음 식이 성립한다.

$$[A-3I]^3 = \begin{bmatrix} -2 & 2 & 3 & -2 \\ -6 & 6 & 9 & -6 \\ 4 & -4 & -6 & 4 \\ 2 & -2 & -3 & 2 \end{bmatrix} \tag{9}$$

다음 벡터들을 정의하자.

$$\boldsymbol{v}_{31} = \begin{bmatrix} -1, & 0, & 0, & 1 \end{bmatrix}^t, \quad \boldsymbol{v}_{32} = \begin{bmatrix} -3, & 0, & 2, & 0 \end{bmatrix}^t, \quad \boldsymbol{v}_{33} = \begin{bmatrix} 1, & 1, & 0, & 0 \end{bmatrix}^t \tag{10}$$

벡터들의 집합 $\{\boldsymbol{v}_{31}, \boldsymbol{v}_{32}, \boldsymbol{v}_{33}\}$는 행렬 $[A-3I]^3$의 커넬의 기저이다. 랭크가 3인 일반화특성벡터는 이 기저에 의해서 생성된 부분공간 $\mathrm{Sp}(\{\boldsymbol{v}_{31}, \boldsymbol{v}_{32}, \boldsymbol{v}_{33}\})$에 속한다. 또한 벡터들 $\boldsymbol{v}_{11}, \boldsymbol{v}_{21}, \boldsymbol{v}_{22}$도 행렬 $[A-3I]^3$의 커넬에 속한다.

다음 식이 성립한다.

$$[A-3I]^4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{11}$$

다음 벡터들을 정의하자.

$$\boldsymbol{v}_{41} = \boldsymbol{e}_1 = \begin{bmatrix} 1, & 0, & 0, & 0 \end{bmatrix}^t, \qquad \boldsymbol{v}_{42} = \boldsymbol{e}_2 = \begin{bmatrix} 0, & 1, & 0, & 0 \end{bmatrix}^t \tag{12}$$

$$\boldsymbol{v}_{43} = \boldsymbol{e}_3 = \begin{bmatrix} 0, & 0, & 1, & 0 \end{bmatrix}^t, \qquad \boldsymbol{v}_{44} = \boldsymbol{e}_4 = \begin{bmatrix} 0, & 0, & 0, & 1 \end{bmatrix}^t \tag{13}$$

벡터들의 집합 $\{\boldsymbol{v}_{41}, \boldsymbol{v}_{42}, \boldsymbol{v}_{43}, \boldsymbol{v}_{44}\}$는 행렬 $[A-3I]^4$의 커넬의 기저이다. 랭크가 4인 일반화특성벡터는 이 기저에 의해서 생성된 부분공간 $\mathrm{Sp}(\{\boldsymbol{v}_{41}, \boldsymbol{v}_{42}, \boldsymbol{v}_{43}, \boldsymbol{v}_{44}\})$에 속한다. 또한 벡터들 $\boldsymbol{v}_{11}, \boldsymbol{v}_{21}, \boldsymbol{v}_{22}, \boldsymbol{v}_{31}, \boldsymbol{v}_{32}, \boldsymbol{v}_{33}$가 행렬 $[A-3I]^4$의 커넬에 속하는 것은 당연하다.

앞에서 언급했듯이, 벡터 $\boldsymbol{x}_4 = \boldsymbol{e}_1$은 랭크가 4인 일반화특성벡터이다. 이 벡터 $\boldsymbol{x}_4$를 씨앗(seed)로 하는 Jordan체인을 구축하자. 다음 식들이 성립한다.

$$\boldsymbol{x}_3 = [A-3I_4]\,\boldsymbol{x}_4 = \begin{bmatrix} 1, & 7, & -6, & -3 \end{bmatrix}^t \tag{14}$$

$$\boldsymbol{x}_2 = [A-3I_4]\,\boldsymbol{x}_3 = [A-3I_4]^2\,\boldsymbol{x}_4 = \begin{bmatrix} 6, & 24, & -20, & -12 \end{bmatrix}^t \tag{15}$$

$$\boldsymbol{x}_1 = [A-3I_4]\,\boldsymbol{x}_2 = [A-3I_4]^2\,\boldsymbol{x}_3 = [A-3I_4]^3\,\boldsymbol{x}_4 = 2\begin{bmatrix} -1, & -3, & 2, & 1 \end{bmatrix}^t \tag{16}$$

식 (5)와 식 (16)에서 알 수 있듯이, $\boldsymbol{x}_1$은 특성값 $\lambda_1 = 3$에 대한 특성벡터이다. 다음과 같은 행렬을 정의하자.

$$H = [\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4] = \begin{bmatrix} -2 & 6 & 1 & 1 \\ -6 & 24 & 7 & 0 \\ 4 & -20 & -6 & 0 \\ 2 & -12 & -3 & 0 \end{bmatrix} \tag{17}$$

다음 식이 성립한다.

$$\det(H) = -16 \tag{18}$$

벡터들의 집합 $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4\}$는 서로 독립이며, 길이가 4인 Jordan체인이다.

다음 식이 성립한다.

$$H^{-1} = \begin{bmatrix} 0 & -\frac{3}{4} & -\frac{3}{4} & -\frac{1}{4} \\ 0 & 0 & \frac{1}{4} & -\frac{1}{2} \\ 0 & -\frac{1}{2} & -\frac{3}{2} & \frac{3}{2} \\ 1 & -1 & -\frac{3}{2} & 1 \end{bmatrix} \tag{19}$$

다음 식이 성립함을 쉽게 알 수 있다.

$$H^{-1}AH = \begin{bmatrix} 3 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 3 \end{bmatrix} \tag{20}$$

식 (20)의 우변은 Jordan행렬이다. 이 행렬을 J로 표기하면, 행렬 A를 다음과 같은 Jordan 표준형으로 나타낼 수 있다.

$$A = HJH^{-1} \tag{21}$$

■

예제 7.8.10 다음과 같은 행렬 A를 살펴보자.

$$A = \begin{bmatrix} 4 & -4 & -11 & 11 \\ 3 & -12 & -42 & 42 \\ -2 & 12 & 37 & -34 \\ -1 & 7 & 20 & -17 \end{bmatrix} \tag{1}$$

다음과 같은 행렬들을 정의하자.

$$H = \begin{bmatrix} 3 & 3 & 14 & 0 \\ 9 & 0 & 0 & 0 \\ -6 & 6 & 1 & 1 \\ -3 & 6 & 0 & 1 \end{bmatrix}, \quad J = \begin{bmatrix} 3 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} \tag{2}$$

예제 7.8.9의 방법을 사용해서, 행렬 A를 다음과 같은 Jordan표준형으로 나타낼 수 있다.

$$A = HJH^{-1} \tag{3}$$

■

예제 7.8.11 다음과 같은 행렬을 살펴보자.

$$A = \begin{bmatrix} 4 & -1 & -2 & 2 \\ 7 & -4 & -12 & 10 \\ -6 & 6 & 13 & -8 \\ -3 & 3 & 5 & -1 \end{bmatrix} \tag{1}$$

다음과 같은 행렬들을 정의하자.

$$H = \begin{bmatrix} 1 & 1 & 2 & 0 \\ 7 & 0 & 10 & 0 \\ -6 & 0 & -8 & 0 \\ -3 & 0 & -4 & 1 \end{bmatrix}, \quad J = \begin{bmatrix} 3 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 3 \end{bmatrix} \tag{2}$$

예제 7.8.9의 방법을 사용해서, 행렬 A를 다음과 같은 Jordan표준형으로 나타낼 수 있다.

$$A = HJH^{-1} \tag{3}$$

■

차원이 5×5인 행렬의 Jordan분해에 대해서 살펴보자.

예제 7.8.12 다음과 같은 행렬 A를 살펴보자.

$$A = \begin{bmatrix} 2 & 5 & 0 & 0 & 1 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{1}$$

다음과 같은 행렬들을 정의하자.

$$H = \begin{bmatrix} -1 & -15 & 5 & 0 & 0 \\ 0 & 0 & -3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad J = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix} \tag{2}$$

예제 7.8.9의 방법을 사용해서, 행렬 A를 다음과 같은 Jordan표준형으로 나타낼 수 있다.

$$A = HJH^{-1} \tag{3}$$

■

7.8.2 Jordan표준형과 미분방정식

다음 연립미분방정식을 생각해보자.

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) \tag{7.8.11}$$

여기서 벡터 $\boldsymbol{x}$와 행렬 A는 각각 다음과 같다.

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_p(t) \end{bmatrix}, \quad A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pp} \end{bmatrix} \tag{7.8.12}$$

연립미분방정식 (7.8.11)의 해는 다음과 같다.

$$\boldsymbol{x}(t) = e^{At}\boldsymbol{x}(0) \tag{7.8.13}$$

행렬 A의 Jordan표준형이 다음과 같다고 하자.

$$A = HJH^{-1} \tag{7.8.14}$$

다음 식들이 성립한다.

$$\begin{aligned} \boldsymbol{x}(t) = e^{At}\boldsymbol{x}(0) &= \sum_{i=0}^{\infty} \frac{1}{i!} A^i t^i \boldsymbol{x}(0) = \sum_{i=0}^{\infty} \frac{1}{i!} \left[HJH^{-1}\right]^i t^i \boldsymbol{x}(0) \\ &= \sum_{i=0}^{\infty} \frac{1}{i!} HJ^iH^{-1}t^i\boldsymbol{x}(0) = H\left[\sum_{i=0}^{\infty} \frac{1}{i!} t^i J^i\right] H^{-1}\boldsymbol{x}(0) = He^{Jt}H^{-1}\boldsymbol{x}(0) \end{aligned} \tag{7.8.15}$$

다음 Jordan블록을 살펴보자.

$$J \doteq \begin{bmatrix} \lambda & 1 & 0 & 0 & 0 \\ 0 & \lambda & 1 & 0 & 0 \\ 0 & 0 & \lambda & 1 & 0 \\ 0 & 0 & 0 & \lambda & 1 \\ 0 & 0 & 0 & 0 & \lambda \end{bmatrix} \tag{7.8.16}$$

다음 식들이 성립한다.

$$J^2 \doteq \begin{bmatrix} \binom{2}{0}\lambda^2 & \binom{2}{1}\lambda^1 & \binom{2}{2}\lambda^0 & 0 & 0 \\ 0 & \binom{2}{0}\lambda^2 & \binom{2}{1}\lambda^1 & \binom{2}{2}\lambda^0 & 0 \\ 0 & 0 & \binom{2}{0}\lambda^2 & \binom{2}{1}\lambda^1 & \binom{2}{2}\lambda^0 \\ 0 & 0 & 0 & \binom{2}{0}\lambda^2 & \binom{2}{1}\lambda^1 \\ 0 & 0 & 0 & 0 & \binom{2}{0}\lambda^2 \end{bmatrix} \tag{7.8.17}$$

$$J^3 \doteq \begin{bmatrix} \binom{3}{0}\lambda^3 & \binom{3}{1}\lambda^2 & \binom{3}{2}\lambda^1 & \binom{3}{3}\lambda^0 & 0 \\ 0 & \binom{3}{0}\lambda^3 & \binom{3}{1}\lambda^2 & \binom{3}{2}\lambda^1 & \binom{3}{3}\lambda^0 \\ 0 & 0 & \binom{3}{0}\lambda^3 & \binom{3}{1}\lambda^2 & \binom{3}{2}\lambda^1 \\ 0 & 0 & 0 & \binom{3}{0}\lambda^3 & \binom{3}{1}\lambda^2 \\ 0 & 0 & 0 & 0 & \binom{3}{0}\lambda^3 \end{bmatrix} \tag{7.8.18}$$

$$J^4 \doteq \begin{bmatrix} \binom{4}{0}\lambda^4 & \binom{4}{1}\lambda^3 & \binom{4}{2}\lambda^2 & \binom{4}{3}\lambda^1 & \binom{4}{4}\lambda^0 \\ 0 & \binom{4}{0}\lambda^4 & \binom{4}{1}\lambda^3 & \binom{4}{2}\lambda^2 & \binom{4}{3}\lambda^1 \\ 0 & 0 & \binom{4}{0}\lambda^4 & \binom{4}{1}\lambda^3 & \binom{4}{2}\lambda^2 \\ 0 & 0 & 0 & \binom{4}{0}\lambda^4 & \binom{4}{1}\lambda^3 \\ 0 & 0 & 0 & 0 & \binom{4}{0}\lambda^4 \end{bmatrix} \tag{7.8.19}$$

각 $k(=5,6,\cdots)$에 대해서 다음 식이 성립한다.

$$J^k \doteq \begin{bmatrix} \binom{k}{0}\lambda^k & \binom{k}{1}\lambda^{k-1} & \binom{k}{2}\lambda^{k-2} & \binom{k}{3}\lambda^{k-3} & \binom{k}{k}\lambda^{k-4} \\ 0 & \binom{k}{0}\lambda^k & \binom{k}{1}\lambda^{k-1} & \binom{k}{2}\lambda^{k-2} & \binom{k}{3}\lambda^{k-3} \\ 0 & 0 & \binom{k}{0}\lambda^k & \binom{k}{1}\lambda^{k-1} & \binom{k}{2}\lambda^{k-2} \\ 0 & 0 & 0 & \binom{k}{0}\lambda^k & \binom{k}{1}\lambda^{k-1} \\ 0 & 0 & 0 & 0 & \binom{k}{0}\lambda^k \end{bmatrix} \tag{7.8.20}$$

식 (7.8.16) $\sim$ 식 (7.8.20)에서 알 수 있듯이, 복소수평면에서 해석적인(analytic) 함수 $f(z)$

에 대해서 행렬함수 $f(J)$를 다음과 같이 정의한다.

$$f(J) \doteq \begin{bmatrix} f(\lambda) & \frac{1}{1!}f^{(1)}(\lambda) & \frac{1}{2!}f^{(2)}(\lambda) & \frac{1}{3!}f^{(3)}(\lambda) & \frac{1}{4!}f^{(4)}(\lambda) \\ 0 & f(\lambda) & \frac{1}{1!}f^{(1)}(\lambda) & \frac{1}{2!}f^{(2)}(\lambda) & \frac{1}{3!}f^{(3)}(\lambda) \\ 0 & 0 & f(\lambda) & \frac{1}{1!}f^{(1)}(\lambda) & \frac{1}{2!}f^{(2)}(\lambda) \\ 0 & 0 & 0 & f(\lambda) & \frac{1}{1!}f^{(1)}(\lambda) \\ 0 & 0 & 0 & 0 & f(\lambda) \end{bmatrix} \tag{7.8.21}$$

식 (7.8.16) $\sim$ 식 (7.8.21)의 각 행렬은 상삼각행렬이며 Toeplitz행렬이다.

식 (7.8.15)와 식 (7.8.21)에서 알 수 있듯이, 길이가 k인 Jordan체인에 해당하는 미분방정식 (7.8.11)의 해들은 다음과 같다.

$$\boldsymbol{x}_1(t) = e^{\lambda t}\boldsymbol{v}_1 \tag{7.8.22}$$

$$\boldsymbol{x}_2(t) = e^{\lambda t}\left[\frac{t}{1!}\boldsymbol{v}_1 + \boldsymbol{v}_2\right] \tag{7.8.23}$$

$$\boldsymbol{x}_3(t) = e^{\lambda t}\left[\frac{t^2}{2!}\boldsymbol{v}_1 + \frac{t}{1!}\boldsymbol{v}_2 + \boldsymbol{v}_3\right] \tag{7.8.24}$$

$$\vdots$$

$$\boldsymbol{x}_k(t) = e^{\lambda t}\left[\frac{t^{k-1}}{(k-1)!}\boldsymbol{v}_1 + \frac{t^{k-2}}{(k-2)!}\boldsymbol{v}_2 + \cdots + \frac{t}{1!}\boldsymbol{v}_{k-1} + \boldsymbol{v}_k\right] \tag{7.8.25}$$

앞에서도 언급했듯이, $\boldsymbol{x}_1(t), \boldsymbol{x}_2(t), \cdots, \boldsymbol{x}_k(t)$는 선형독립이다. Wronskian행렬식을 사용해서 이 성질이 성립함을 확인할 수 있다. 행렬 A의 각 Jordan블록에 해당하는 Jordan체인의 집합은 미분방정식 (7.8.11)의 서로 독립인 해들로 이들의 개수는 해당 정방행렬 A의 차원과 같다. 이러한 집합을 미분방정식 (7.8.11)의 근본적 시스템이라 부른다.

예제 7.8.13 다음 초기값문제를 살펴보자.

$$\dot{x} = 2x - 3y\,, \quad \dot{y} = -x + 4y\,, \quad x(0) = 4\,, \quad y(0) = 0 \tag{1}$$

미분방정식 (1)을 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 4 \\ 0 \end{bmatrix} \tag{2}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x \\ y \end{bmatrix}, \quad A = \begin{bmatrix} 2 & -3 \\ -1 & 4 \end{bmatrix} \tag{3}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 2-\lambda & -3 \\ -1 & 4-\lambda \end{vmatrix} = \lambda^2 - 6\lambda + 5 = 0 \tag{4}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = 1\,, \quad \lambda_2 = 5 \tag{5}$$

특성값 $\lambda_1 = 1$의 대수중복도(algebraic multiplicity)는 $k_1 = 1$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{11},\, v_{21}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} 2-1 & -3 \\ -1 & 4-1 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{6}$$

즉, 다음 식이 성립한다.

$$v_{11} - 3v_{21} = 0 \tag{7}$$

따라서 특성값 $\lambda_1 = 1$에 해당하는 특성벡터는 $\boldsymbol{v}_1 = [3,\, 1]^t$이고, 또한 특성값 $\lambda_1 = 1$의 기하중복도(geometric multiplicity)는 다음과 같다.

$$s_1 = n - \operatorname{rank}(A - \lambda_1 I) = 2 - 1 = 1 \tag{8}$$

특성값 $\lambda_2 = 5$의 대수중복도는 $k_2 = 1$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_2 = [v_{12},\, v_{22}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} 2-5 & -3 \\ -1 & 4-5 \end{bmatrix} \begin{bmatrix} v_{12} \\ v_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{9}$$

즉, 다음 식이 성립한다.

$$v_{12} + v_{22} = 0 \tag{10}$$

따라서 특성값 $\lambda_2 = 5$에 해당하는 특성벡터는 $\boldsymbol{v}_2 = [-1, 1]^t$이고, 또한 특성값 $\lambda_2 = 5$의 기하중복도는 다음과 같다.

$$s_2 = n - \text{rank}(A - \lambda_2 I) = 2 - 1 = 1 \tag{11}$$

각 특성값에 대해 단 하나의 특성벡터가 존재하고, 벡터 $\boldsymbol{v}_1$ 그리고 벡터 $\boldsymbol{v}_2$는 서로 독립이다. 따라서, 미분방정식 (1)의 일반해는 다음과 같다.

$$\boldsymbol{u}(t) = c_1 e^t \boldsymbol{v}_1 + c_2 e^{5t} \boldsymbol{v}_2 = c_1 e^t \begin{bmatrix} 3 \\ 1 \end{bmatrix} + c_2 e^{5t} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \tag{12}$$

식 (1)의 두 번째 식 $x = 4y - \dot{y}$를 첫 번째 식에 대입하면, 다음 식을 얻는다.

$$\ddot{y} - 6\dot{y} + 5y = 0 \tag{13}$$

미분방정식 (13)의 해가 식 (12)를 만족함을 쉽게 알 수 있다.

행렬 A를 다음과 같이 분해할 수 있다.

$$A = HJH^{-1} = \begin{bmatrix} 3 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 5 \end{bmatrix} \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{3}{4} \end{bmatrix} \tag{14}$$

따라서 다음 식이 성립한다.

$$\begin{aligned} \boldsymbol{u}(t) = He^{Jt}H^{-1}\boldsymbol{u}_0 &= \begin{bmatrix} 3 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \sum_{k=0}^{\infty} \frac{1}{k!} 1^k t^k & 0 \\ 0 & \sum_{k=0}^{\infty} \frac{1}{k!} 5^k t^k \end{bmatrix} \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{3}{4} \end{bmatrix} \boldsymbol{u}_0 \\ &= \begin{bmatrix} 3 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} e^t & 0 \\ 0 & e^{5t} \end{bmatrix} \begin{bmatrix} \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{3}{4} \end{bmatrix} \boldsymbol{u}_0 = \begin{bmatrix} 3 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} e^t & 0 \\ 0 & e^{5t} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \end{aligned} \tag{15}$$

즉, 식 (12)가 성립함을 확인할 수 있다.

식 (1)의 초기조건을 일반해 (12)에 대입하면, 다음 식들이 성립한다.

$$c_1 = 1, \quad c_2 = -1 \tag{16}$$

따라서 이 초기값문제의 해는 다음과 같다.

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 3e^t + e^{5t} \\ e^t - e^{5t} \end{bmatrix} \tag{17}$$

이 초기값문제를 풀기 위해서, 다음 MATLAB프로그램 Jordan101.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename Jordan101.m
% https://www.math24.net/general-solution-system-differential
%                          -equations-jordan-form/
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary JordanD101.txt
JordanD101g = dsolve('Dx=2*x-3*y','Dy=-x+4*y','t')
JordanD101g.x, JordanD101g.y
JordanD101 = dsolve('Dx=2*x-3*y','Dy=-x+4*y','x(0)=4','y(0)=0','t')
JordanD101.x, JordanD101.y
% Plotting
tt = linspace(-2,1,201);
x1 = double(subs(JordanD101.x,'t',tt));
y1 = double(subs(JordanD101.y,'t',tt));
plot(x1,y1,'r-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12), ylabel('y','fontsize',12)
saveas(gcf,'Jordan101.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 Jordan101.m에서는 심볼릭함수 dsolve.m을 사용해서 이 초기값문제를 푼다. 이 MATLAB프로그램을 실행한 결과, 식 (17)의 해가 올바름을 확인할 수 있다. 또한, 그림 7.8.1이 그려진다.

다음 MATLAB프로그램 Jordan201.m을 실행하면, MATLAB함수 eig.m을 사용해서 이 초기값문제를 푼다.

```
% ------------------------------------------------------------------
% Filename Jordan201.m
% Solve x' = A*x
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan201.txt
syms t x y
A = [ 2 -3 ; -1 4 ]
u0 = [ 4 0 ]'
% A decomposition solution
[ P D ] = eig(A)
```

```
u = P*expm(D*t)*inv(P)*u0
u = vpa(u)
% A direct solution
uu = expm(A*t)*u0
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행한 결과는 MATLAB프로그램 Jordan101.m을 실행한 결과와 같다. ■

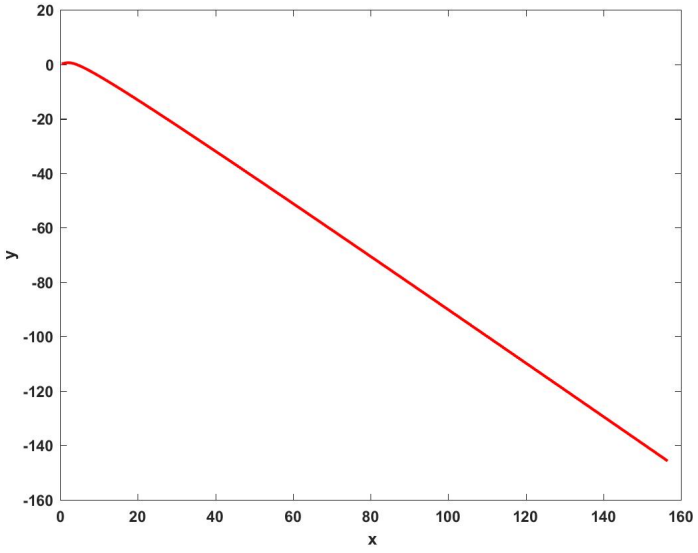


그림 7.8.1. Jordan표준형 미분방정식 I

예제 7.8.14 Python을 사용해서 예제 7.8.13을 다시 다루기 위해서, 다음 Python프로그램 Jordan101.Py를 실행해 보자.

```
"""
% Filename Jordan101.Py
% Jordan form for Ordinary Differential Equation
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t), t), 2*x(t)-3*y(t))
```

```
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), -x(t)+4*y(t))
display(eq2)
sol11 = sy.dsolve((eq1, eq2))
display(sol11)
## Initial condition
C1, C2 = sy.symbols("C1, C2")             # generic constants
constants = sy.solve((sol11[0].subs(t,0).subs(x(0),4), \
                      sol11[1].subs(t,0).subs(y(0),0)),{C1,C2})
display(constants)
## Substitute back into solution
xsol11 = sy.expand(sol11[0].rhs.subs(constants))
ysol11 = sy.simplify(sol11[1].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol11))
display(sy.Eq(y(t),ysol11))
## Double checking
display(sy.Eq(sy.simplify(sy.diff(xsol11,t))+sy.simplify(ysol11)))
display(sy.Eq(sy.simplify(sy.diff(ysol11,t))-sy.simplify(xsol11)))
## Plotting
fig = plt.figure()
sol11xLam = sy.lambdify(t,xsol11, modules=['numpy'])
sol11yLam = sy.lambdify(t,ysol11, modules=['numpy'])
tt = np.linspace(-2,1,201)
xx = sol11xLam(tt)
yy = sol11yLam(tt)
plt.plot(xx,yy,'r-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)
plt.axis('equal')
plt.show()
fig.savefig('Jordan101Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.8.13의 결과와 같다. ■

예제 7.8.15 다음 초기값문제를 살펴보자.

$$\dot{x} = -x\,, \quad \dot{y} = -y\,, \quad x(0) = 4\,, \quad y(0) = 1 \tag{1}$$

초기값문제 (1)을 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 4 \\ 1 \end{bmatrix} \tag{2}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x \\ y \end{bmatrix}\,, \quad A = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} -1-\lambda & 0 \\ 0 & -1-\lambda \end{vmatrix} = \lambda^2 + 2\lambda + 1 = 0 \tag{4}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = -1\,, \quad \lambda_2 = -1 \tag{5}$$

특성값 $\lambda_1 = -1$의 대수중복도는 $k_1 = 2$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{11},\, v_{21}\,]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} -1+1 & 0 \\ 0 & -1+1 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{6}$$

즉, 다음 식이 성립한다.

$$0 \cdot v_{11} + 0 \cdot v_{21} = 0 \tag{7}$$

따라서 어떤 벡터라도 특성값 $\lambda_1 = -1$에 해당하는 특성벡터가 될 수 있다. 즉, 다음과 같은 벡터들을 특성벡터로 선택할 수 있다.

$$\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{v}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{8}$$

특성값 $\lambda_1 = -1$의 기하중복도는 다음과 같다.

$$s_1 = n - \operatorname{rank}(A - \lambda_1 I) = 2 - 0 = 2 \tag{9}$$

벡터 $\boldsymbol{v}_1$ 그리고 벡터 $\boldsymbol{v}_2$는 서로 독립이다. 따라서, 미분방정식 (1)의 일반해는 다음과 같다.

$$\boldsymbol{u}(t) = c_1 e^{-t}\boldsymbol{v}_1 + c_2 e^{-t}\boldsymbol{v}_2 = c_1 e^{-t} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + c_2 e^{-t} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{10}$$

행렬 A를 다음과 같이 분해할 수 있다.

$$A = HJH^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{11}$$

따라서 다음 식이 성립한다.

$$\begin{aligned} \boldsymbol{u}(t) = He^{Jt}H^{-1} &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \sum_{k=0}^{\infty} \frac{1}{k!}[-t]^k & 0 \\ 0 & \sum_{k=0}^{\infty} \frac{1}{k!}[-t]^k \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \boldsymbol{u}_0 \\ &= \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} e^{-t} & 0 \\ 0 & e^{-t} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \boldsymbol{u}_0 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} e^{-t} & 0 \\ 0 & e^{-t} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \end{aligned} \tag{12}$$

즉, 식 (10)이 성립함을 확인할 수 있다.

초기조건 (1)를 일반해 (10)에 대입하면, 다음 식들이 성립한다.

$$c_1 = 4, \quad c_2 = 1 \tag{13}$$

따라서 이 초기값문제의 해는 다음과 같다.

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} 4e^{-t} \\ e^{-t} \end{bmatrix} \tag{14}$$

식 (1)의 첫 번째 미분방정식 $\dot{x} = -x$과 초기조건 $x(0) = 4$에서 알 수 있듯이, 다음 식들이 성립한다.

$$x(t) = e^{-t}x(0) = 4e^{-t} \tag{15}$$

식 (1)의 두 번째 미분방정식 $\dot{y} = -y$와 초기조건 $y(0) = 1$에서 알 수 있듯이, 다음 식들이 성립한다.

$$y(t) = e^{-t}y(0) = e^{-t} \tag{16}$$

따라서 식 (14)가 성립함을 확인할 수 있다.

이 초기값문제를 풀기 위해서, 다음 MATLAB프로그램 Jordan102.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename Jordan102.m
```

```
% https://www.math24.net/general-solution-system-differential
%                          -equations-jordan-form/
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary JordanD102.txt
JordanD102g = dsolve('Dx=-x','Dy=-y','t')
JordanD102g.x, JordanD102g.y
JordanD102 = dsolve('Dx=-x','Dy=-y','x(0)=4','y(0)=1','t')
JordanD102.x, JordanD102.y
% Plotting
tt = linspace(-1,1,201);
x1 = double(subs(JordanD102.x,'t',tt));
y1 = double(subs(JordanD102.y,'t',tt));
plot(x1,y1,'r-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12), ylabel('y','fontsize',12)
saveas(gcf,'Jordan102.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 Jordan102.m에서는 심볼릭함수 dsolve.m을 사용해서 이 초기값 문제를 푼다. 이 MATLAB프로그램을 실행한 결과, 식 (14)의 해가 올바름을 확인할 수 있다. 또한, 그림 7.8.2가 그려진다.

다음 MATLAB프로그램 Jordan202.m을 실행하면, MATLAB함수 eig.m을 사용해서 이 초기값문제를 푼다.

```
% ------------------------------------------------------------------
% Filename Jordan202.m
% Solve x' = A*x
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan202.txt
syms t x y
A = [ -1 0 ; 0 -1 ]
u0 = [ 4 1 ]'
% A decomposition solution
[ P D ] = eig(A)
u = P*expm(D*t)*inv(P)*u0
u = vpa(u)
% A direct solution
uu = expm(A*t)*u0
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행한 결과는 MATLAB프로그램 Jordan102.m을 실행한 결과와 같다. ■

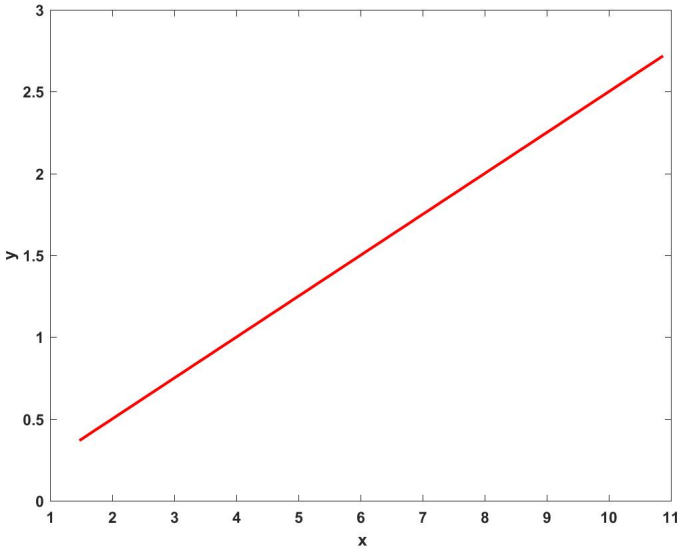


그림 7.8.2. Jordan표준형 미분방정식 II

예제 7.8.16 Python을 사용해서 예제 7.8.15를 다시 다루기 위해서, 다음 Python프로그램 Jordan102.Py를 실행해 보자.

```
"""
% Filename Jordan102.Py
% Jordan form for Ordinary Differential Equation
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t), t), -x(t)-0*y(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), 0*x(t)-y(t))
display(eq2)
sol11 = sy.dsolve((eq1, eq2))
display(sol11)
## Initial condition
C1, C2 = sy.symbols("C1, C2")            # generic constants
constants = sy.solve((sol11[0].subs(t,0).subs(x(0),4), \
                      sol11[1].subs(t,0).subs(y(0),1)),{C1,C2})
display(constants)
## Substitute back into solution
xsol11 = sy.expand(sol11[0].rhs.subs(constants))
ysol11 = sy.simplify(sol11[1].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol11))
display(sy.Eq(y(t),ysol11))
## Double checking
```

```
display(sy.Eq(sy.simplify(sy.diff(xsol11,t))+sy.simplify(ysol11)))
display(sy.Eq(sy.simplify(sy.diff(ysol11,t))-sy.simplify(xsol11)))
## Plotting
fig = plt.figure()
sol11xLam = sy.lambdify(t,xsol11, modules=['numpy'])
sol11yLam = sy.lambdify(t,ysol11, modules=['numpy'])
tt = np.linspace(-1,1,201)
xx = sol11xLam(tt)
yy = sol11yLam(tt)
plt.plot(xx,yy,'r-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)
plt.axis('equal')
plt.show()
fig.savefig('Jordan102Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.8.15의 결과와 같다. ■

예제 7.8.17 다음 초기값문제를 살펴보자.

$$\dot{x} = 2x - y\,, \quad \dot{y} = x + 4y\,, \quad x(0) = 1 \quad y(0) = 2 \tag{1}$$

미분방정식 (1)을 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{2}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x \\ y \end{bmatrix}\,, \quad A = \begin{bmatrix} 2 & -1 \\ 1 & 4 \end{bmatrix} \tag{3}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 2-\lambda & -1 \\ 1 & 4-\lambda \end{vmatrix} = \lambda^2 - 6\lambda + 9 = [\lambda - 3]^2 = 0 \tag{4}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = 3\,, \quad \lambda_2 = 3 \tag{5}$$

특성값 $\lambda_1 = 3$의 대수중복도는 $k_1 = 2$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{11},\, v_{21}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} 2-3 & -1 \\ 1 & 4-3 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{6}$$

즉, 다음 식이 성립한다.

$$v_{11} + v_{21} = 0 \tag{7}$$

즉, 특성값 $\lambda_1 = 3$에 대한 특성벡터를 다음과 같이 선택할 수 있다.

$$\boldsymbol{v}_1 = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \tag{8}$$

특성값 $\lambda_1 = 3$의 기하중복도는 다음과 같다.

$$s_1 = n - \operatorname{rank}(A - \lambda_1 I) = 2 - 1 = 1 \tag{9}$$

다음 식이 성립함을 알 수 있다.

$$A\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1 \tag{10}$$

일반화특성벡터 $\boldsymbol{v}_2 = [v_{12},\, v_{22}]^t$는 다음 식을 만족한다.

$$[A - \lambda_1 I]\, \boldsymbol{v}_2 = \boldsymbol{v}_1 \tag{11}$$

즉, 다음 연립방정식이 성립한다.

$$\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} v_{12} \\ v_{22} \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \tag{12}$$

방정식 (1)의 해는 식 $v_{12} + v_{22} = 1$을 만족한다. 예를 들어, 이 해는 다음과 같다.

$$\boldsymbol{v}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{13}$$

벡터 $\boldsymbol{v}_1$ 그리고 벡터 $\boldsymbol{v}_2$는 서로 독립이다.

행렬 A를 다음과 같은 Jordan표준형으로 나타낼 수 있다.

$$A = HJH^{-1} \tag{14}$$

여기서 J, H와 H^{-1}는 다음과 같다.

$$J = \begin{bmatrix} 3 & 1 \\ 0 & 3 \end{bmatrix}, \quad H = \begin{bmatrix} \boldsymbol{v}_1, & \boldsymbol{v}_2 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}, \quad H^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \tag{15}$$

식 (14)를 다음과 같이 증명할 수 있다.

$$AH = \begin{bmatrix} 2 & -1 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} -3 & 2 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 0 & 3 \end{bmatrix} = HJ \tag{16}$$

다음 식들이 성립한다.

$$e^{Jt} = \sum_{k=0}^{\infty} \frac{1}{k!} J^k t^k = \begin{bmatrix} \sum\limits_{k=0}^{\infty} \frac{1}{k!} 3^k t^k & \sum\limits_{k=0}^{\infty} \frac{1}{k!} k 3^{k-1} t^k \\ 0 & \sum\limits_{k=0}^{\infty} \frac{1}{k!} 3^k t^k \end{bmatrix} = \begin{bmatrix} e^{3t} & te^{3t} \\ 0 & e^{3t} \end{bmatrix} \tag{17}$$

따라서 연립미분방정식 (1)의 일반해는 다음과 같다.

$$\boldsymbol{u}(t) = He^{Jt}H^{-1}\boldsymbol{u}_0 = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} e^{3t} & te^{3t} \\ 0 & e^{3t} \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} u_{01} \\ u_{02} \end{bmatrix} \tag{18}$$

식 (18)을 정리하면 다음과 같다.

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} e^{3t}[-t+1] & -e^{3t}t \\ e^{3t}t & e^{3t}[t+1] \end{bmatrix} \boldsymbol{u}_0 \tag{19}$$

식 (19)를 다음과 같이 쓸 수 있다.

$$\boldsymbol{u}(t) = C_1 e^{3t}\boldsymbol{v}_1 + C_2 e^{3t}\left[t\,\boldsymbol{v}_1 + \boldsymbol{v}_2\right] \tag{20}$$

여기서 $C_1 = u_{02}$이고 $C_2 = u_{01} + u_{02}$이다.

식 (1)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\ddot{x} = 2\dot{x} - \dot{y} = 2\dot{x} - [x + 4y] = 2\dot{x} - x - 4[2x - \dot{x}] \tag{21}$$

즉, 다음 식이 성립한다.

$$\ddot{x} - 6\dot{x} + 9x = 0 \tag{22}$$

미분방정식 (22)의 해가 식 (19)의 벡터임을 쉽게 알 수 있다.

식 (1)의 초기조건을 일반해 (20)에 대입하면, 이 초기값문제의 해가 다음과 같음을 알 수 있다.

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} e^{3t}[1 - 3t] \\ e^{3t}[2 + 3t] \end{bmatrix} \tag{23}$$

이 초기값문제를 풀기 위해서, 다음 MATLAB프로그램 Jordan103.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename Jordan103.m
% https://www.math24.net/general-solution-system-differential
%                          -equations-jordan-form/
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary JordanD103.txt
JordanD103g = dsolve('Dx=2*x-y','Dy=x+4*y','t')
JordanD103g.x, JordanD103g.y
JordanD103 = dsolve('Dx=2*x-y','Dy=x+4*y','x(0)=1','y(0)=2','t')
JordanD103.x, JordanD103.y
% Plotting
tt = linspace(-3,1,201);
x1 = double(subs(JordanD103.x,'t',tt));
y1 = double(subs(JordanD103.y,'t',tt));
plot(x1,y1,'r-','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12), ylabel('y','fontsize',12)
saveas(gcf,'Jordan103.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 Jordan103.m에서는 심볼릭함수 dsolve.m을 사용해서 이 초기값문제를 푼다. 이 MATLAB프로그램을 실행한 결과, 식 (23)의 해가 올바름을 확인할 수 있다. 또한, 그림 7.8.3이 그려진다.

다음 MATLAB프로그램 Jordan203.m을 실행하면, MATLAB함수 eig.m을 사용해서 이 초기값문제를 푼다.

```
% ------------------------------------------------------------------
% Filename Jordan203.m
% Solve x' = A*x
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan203.txt
syms t x y
A = [ 2 -1; 1 4 ]
u0 = [ 1 2 ]'
% A decomposition solution
[ H J ] = jordan(A)
u = H*expm(J*t)*inv(H)*u0
u = vpa(u)
% A direct solution
uu = expm(A*t)*u0
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행한 결과는 MATLAB프로그램 Jordan103.m을 실행한 결과와 같다. ■

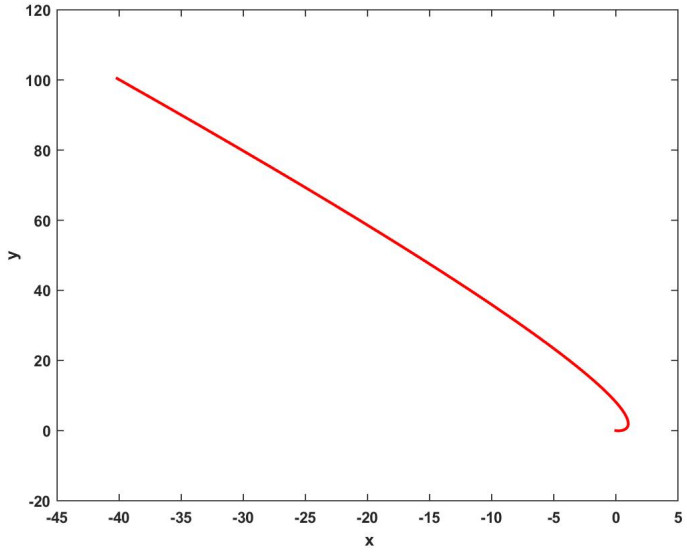


그림 7.8.3. Jordan표준형 미분방정식 III

예제 7.8.18 Python을 사용해서 예제 7.8.17을 다시 다루기 위해서, 다음 Python프로그램 Jordan103.Py를 실행해 보자.

```
"""
% Filename Jordan103.Py
% Jordan form for Ordinary Differential Equation
```

```
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

x = sy.Function('x')
y = sy.Function('y')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t), t), 2*x(t)-y(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), 1*x(t)+4*y(t))
display(eq2)
sol11 = sy.dsolve((eq1, eq2))
display(sol11)
## Initial condition
C1, C2 = sy.symbols("C1, C2")            # generic constants
constants = sy.solve((sol11[0].subs(t,0).subs(x(0),1), \
                      sol11[1].subs(t,0).subs(y(0),2)),{C1,C2})
display(constants)
## Substitute back into solution
xsol11 = sy.expand(sol11[0].rhs.subs(constants))
ysol11 = sy.simplify(sol11[1].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol11))
display(sy.Eq(y(t),ysol11))
## Double checking
display(sy.Eq(sy.simplify(sy.diff(xsol11,t))+sy.simplify(ysol11)))
display(sy.Eq(sy.simplify(sy.diff(ysol11,t))-sy.simplify(xsol11)))
## Plotting
fig = plt.figure()
sol11xLam = sy.lambdify(t,xsol11, modules=['numpy'])
sol11yLam = sy.lambdify(t,ysol11, modules=['numpy'])
tt = np.linspace(-3,1,501)
xx = sol11xLam(tt)
yy = sol11yLam(tt)
plt.plot(xx,yy,'r-',lw=2.0)
plt.xlabel('x',fontsize=12), plt.ylabel('y',fontsize=12)
plt.axis('equal')
plt.show()
fig.savefig('Jordan103Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.8.17의 결과와 같다. ■

예제 7.8.19 다음 초기값문제를 살펴보자.

$$\dot{x} = -4x - 6y - 6z\,, \quad \dot{y} = x + 3y + z\,, \quad \dot{z} = 2x + 4z \tag{1}$$

$$x(0) = -1\,, \quad y(0) = 0\,, \quad z(0) = 0 \tag{2}$$

미분방정식 (1)과 초기조건을 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 각각 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}\,, \quad A = \begin{bmatrix} -4 & -6 & -6 \\ 1 & 3 & 1 \\ 2 & 0 & 4 \end{bmatrix} \tag{4}$$

행렬 A의 특성방정식은 다음과 같다.

$$\begin{aligned} f(\lambda) = \det(A - \lambda I) &= \begin{vmatrix} -4-\lambda & -6 & -6 \\ 1 & 3-\lambda & 1 \\ 2 & 0 & 4-\lambda \end{vmatrix} \\ &= \lambda^3 - 3\lambda^2 + 2\lambda = \lambda\,[\lambda - 1]\,[\lambda - 2] = 0 \end{aligned} \tag{5}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = 0\,, \quad \lambda_2 = 1\,, \quad \lambda_3 = 2 \tag{6}$$

특성값 $\lambda_1 = 0$의 대수중복도는 $k_1 = 1$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{12},\, v_{22}\,, v_{32}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} -4 & -6 & -6 \\ 1 & 3 & 1 \\ 2 & 0 & 4 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \\ v_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

즉, 다음 연립방정식이 성립한다.

$$v_{11} + 3v_{21} + v_{31} = 0, \quad 0v_{11} + 3v_{21} - v_{31} = 0 \tag{8}$$

즉, 특성값 $\lambda_1 = 0$에 대한 특성벡터는 다음과 같다.

$$\boldsymbol{v}_1 = s\ [-6,\ 1,\ 3\,]^t \tag{9}$$

여기서 s는 자유모수(free parameter)이다. 특성값 $\lambda_1 = 0$의 기하중복도는 다음과 같다.

$$s_1 = n - \operatorname{rank}(A - \lambda_1 I) = 3 - 2 = 1 \tag{10}$$

다음 식을 만족함을 확인할 수 있다.

$$A\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1 \tag{11}$$

특성값 $\lambda_2 = 1$의 대수중복도는 $k_2 = 1$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_2 = [v_{12},\, v_{22}\,,\ v_{32}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} -4-1 & -6 & -6 \\ 1 & 3-1 & 1 \\ 2 & 0 & 4-1 \end{bmatrix} \begin{bmatrix} v_{12} \\ v_{22} \\ v_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{12}$$

즉, 다음 연립방정식이 성립한다.

$$v_{12} + 2v_{22} + v_{32} = 0, \quad 0v_{12} + 4v_{22} - v_{32} = 0 \tag{13}$$

즉, 특성값 $\lambda_2 = 1$에 대한 특성벡터는 다음과 같다.

$$\boldsymbol{v}_2 = s\ [-6,\ 1,\ 4\,]^t \tag{14}$$

여기서 s는 자유모수이다. 특성값 $\lambda_2 = 1$의 기하중복도는 다음과 같다.

$$s_2 = n - \operatorname{rank}(A - \lambda_2 I) = 3 - 2 = 1 \tag{15}$$

다음 식을 만족함을 확인할 수 있다.

$$A\boldsymbol{v}_2 = \lambda_2 \boldsymbol{v}_2 \tag{16}$$

특성값 $\lambda_3 = 2$의 대수중복도는 $k_3 = 1$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_3 = [v_{13},\, v_{23}\,, v_{33}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} -4-2 & -6 & -6 \\ 1 & 3-2 & 1 \\ 2 & 0 & 4-2 \end{bmatrix} \begin{bmatrix} v_{13} \\ v_{23} \\ v_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{17}$$

즉, 다음 연립방정식이 성립한다.

$$v_{13} + v_{23} + v_{33} = 0, \quad v_{13} + 0v_{23} + v_{33} = 0 \tag{18}$$

즉, 특성값 $\lambda_3 = 0$에 대한 특성벡터는 다음과 같다.

$$\boldsymbol{v}_3 = s\ [-1,\, 0,\, 1\,]^t \tag{19}$$

여기서 s는 자유모수이다. 특성값 $\lambda_3 = 2$의 기하중복도는 다음과 같다.

$$s_3 = n - \mathrm{rank}(A - \lambda_3 I) = 3 - 2 = 1 \tag{20}$$

다음 식을 만족함을 확인할 수 있다.

$$A\boldsymbol{v}_3 = \lambda_3 \boldsymbol{v}_3 \tag{21}$$

벡터 $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ 그리고 벡터 $\boldsymbol{v}_3$는 서로 독립이다. 행렬 A를 다음과 같이 분해할 수 있다.

$$A = HJH^{-1} \tag{22}$$

여기서 J, H와 H^{-1}는 다음과 같다.

$$J = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \quad H = \begin{bmatrix} -6 & -6 & -1 \\ 1 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix}, \quad H^{-1} = \begin{bmatrix} -1 & -2 & -1 \\ 1 & 3 & 1 \\ -1 & -6 & 0 \end{bmatrix} \tag{23}$$

식 (23)을 다음과 같이 증명할 수 있다.

$$AH = \begin{bmatrix} 0 & -6 & -2 \\ 0 & 1 & 0 \\ 0 & 4 & 2 \end{bmatrix} = HJ \tag{24}$$

다음 식들이 성립한다.

$$e^{Jt} = \sum_{k=0}^{\infty} \frac{1}{k!} J^k t^k = \begin{bmatrix} 0 & 0 & 0 \\ 0 & \sum\limits_{k=0}^{\infty} \frac{1}{k!} t^k & 0 \\ 0 & 0 & \sum\limits_{k=0}^{\infty} \frac{1}{k!} 2^k t^k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & e^t & 0 \\ 0 & 0 & e^{2t} \end{bmatrix} \tag{25}$$

따라서 연립미분방정식 (1)의 일반해는 다음과 같다.

$$\boldsymbol{u}(t) = He^{Jt}H^{-1}\boldsymbol{u}_0 = \begin{bmatrix} -6 & -6 & -1 \\ 1 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & e^t & 0 \\ 0 & 0 & e^{2t} \end{bmatrix} \begin{bmatrix} -1 & -2 & -1 \\ 1 & 3 & 1 \\ -1 & -6 & 0 \end{bmatrix} \begin{bmatrix} u_{01} \\ u_{02} \\ u_{03} \end{bmatrix} \tag{26}$$

식 (26)을 정리하면 다음과 같다.

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} -6 & -6e^t & -e^{2t} \\ 1 & 1e^t & 0 \\ 3 & 4e^t & e^{2t} \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \tag{27}$$

여기서 C_1, C_2, C_3는 상수들이다.

식 (27)을 초기조건 (2)에 대입하면, 다음 식들이 성립한다.

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} -6 & -6 & -1 \\ 1 & 1 & 0 \\ 3 & 4 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 & -2 & -1 \\ 1 & 3 & 1 \\ -1 & -6 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \tag{28}$$

즉, 이 초기값문제의 해가 다음과 같음을 알 수 있다.

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} -6 + 6e^t - e^{2t} \\ 1 - e^t \\ 3 - 4e^t + e^{2t} \end{bmatrix} \tag{29}$$

이 초기값문제를 풀기 위해서, 다음 MATLAB프로그램 Jordan104.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename Jordan104.m
% https://www.math24.net/general-solution-system-differential
%                          -equations-jordan-form/
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan104.txt
JordanD104g = dsolve('Dx=-4*x-6*y-6*z','Dy=x+3*y+z','Dz=2*x+4*z','t')
JordanD104g.x, JordanD104g.y,JordanD104g.z
JordanD104 = dsolve('Dx=-4*x-6*y-6*z','Dy=x+3*y+z','Dz=2*x+4*z', ...
                    'x(0)=-1','y(0)=0','z(0)=0','t')
JordanD104.x, JordanD104.y,JordanD104.z
% Plotting
tt = linspace(-10,1,201);
x1 = double(subs(JordanD104.x,'t',tt));
y1 = double(subs(JordanD104.y,'t',tt));
z1 = double(subs(JordanD104.z,'t',tt));
plot3(x1,y1,z1,'r-','linewidth',2)
set(gca,'fontsize',11,'Fontweigh','bold')
xlabel('x'), ylabel('y'),zlabel('z')
saveas(gcf,'Jordan104.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 Jordan104.m에서는 심볼릭함수 dsolve.m을 사용해서 이 초기값문제를 푼다. 이 MATLAB프로그램을 실행한 결과, 식 (29)의 해가 올바름을 확인할 수 있다. 또한, 그림 7.8.4가 그려진다.

다음 MATLAB프로그램 Jordan204.m을 실행하면, MATLAB함수 eig.m을 사용해서 이 초기값문제를 푼다.

```
% ------------------------------------------------------------------
% Filename Jordan204.m
% Solve x' = A*x
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan204.txt
syms t x y
A = [ -4 6 -6; 1 3 1; 2 0 4 ]
u0 = [ -1 0 0 ]'
```

```
% A decomposition solution
[ H J ] = eig(A)
u = H*expm(J*t)*inv(H)*u0
u = vpa(u)
% A direct solution
uu = expm(A*t)*u0
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행한 결과는 MATLAB프로그램 Jordan104.m을 실행한 결과와 같다. ■

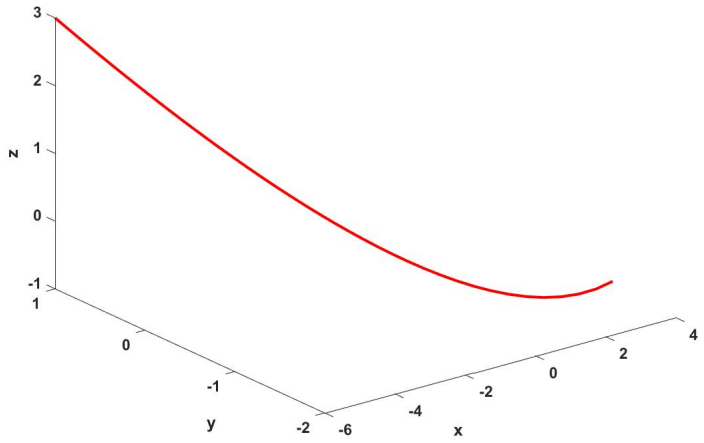


그림 7.8.4. Jordan표준형 미분방정식 IV

예제 7.8.20 Python을 사용해서 예제 7.8.19를 다시 다루기 위해서, 다음 Python프로그램 Jordan104.Py를 실행해 보자.

```
"""
% Filename Jordan104.Py
% Jordan form for Ordinary Differential Equation
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

x = sy.Function('x')
y = sy.Function('y')
```

```
z = sy.Function('z')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t),t), -4*x(t)-6*y(t)-6*z(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), x(t)+3*y(t)+z(t))
display(eq2)
eq3 = sy.Eq(sy.diff(z(t),t), 2*x(t)+0*y(t)+4*z(t))
display(eq3)
sol13 = sy.dsolve((eq1, eq2, eq3))
display(sol13)
## Initial condition
C1, C2, C3 = sy.symbols("C1, C2, C3")            # generic constants
cond0 = sy.Eq(sol13[0].rhs.subs(t,0),-1)
display(cond0)
cond1 = sy.Eq(sol13[1].rhs.subs(t,0),0)
display(cond1)
cond2 = sy.Eq(sol13[2].rhs.subs(t,0),0)
display(cond2)
constants = sy.solve((cond0,cond1,cond2),{C1,C2,C3})
display(constants)
## Substitute back into solution
xsol13 = sy.expand(sol13[0].rhs.subs(constants))
ysol13 = sy.expand(sol13[1].rhs.subs(constants))
zsol13 = sy.expand(sol13[2].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol13))
display(sy.Eq(y(t),ysol13))
display(sy.Eq(z(t),zsol13))
## Double checking
dumx = sy.diff(xsol13,t)+4*xsol13+6*ysol13+6*zsol13;
print(dumx)
dumy = sy.diff(ysol13,t)-xsol13-3*ysol13-zsol13;
print(dumy)
dumz = sy.diff(zsol13,t)-2*xsol13-0*ysol13-4*zsol13;
print(dumz)
## Plotting
sol13xLam = sy.lambdify(t,xsol13, modules=['numpy'])
sol13yLam = sy.lambdify(t,ysol13, modules=['numpy'])
sol13zLam = sy.lambdify(t,zsol13, modules=['numpy'])
tt = np.linspace(-10,1,501)
xx = sol13xLam(tt)
yy = sol13yLam(tt)
zz = sol13zLam(tt)
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(111)
ax = Axes3D(fig)
ax.scatter(xx,yy,zz,color='r',marker='*')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
fig.savefig('Jordan104Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.8.19의 결과와 같다. ■

예제 7.8.21 다음 초기값문제를 살펴보자.

$$\dot{x} = x - y - z\,, \quad \dot{y} = -x + y - z\,, \quad \dot{z} = -x - y + z \tag{1}$$

$$x(0) = -4\,, \quad y(0) = 4\,, \quad z(0) = 3 \tag{2}$$

미분방정식 (1)과 초기조건 (2)를 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} -4 \\ 4 \\ 3 \end{bmatrix} \tag{3}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}\,, \quad A = \begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} \tag{4}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} 1-\lambda & -1 & -1 \\ -1 & 1-\lambda & -1 \\ -1 & -1 & 1-\lambda \end{vmatrix}$$

$$= \lambda^3 - 3\lambda^2 + 4 = [\lambda + 1]\,[\lambda - 2]^2 = 0 \tag{5}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = -1\,, \quad \lambda_2 = 2\,, \quad \lambda_3 = 2 \tag{6}$$

특성값 $\lambda_1 = -1$의 대수중복도는 $k_1 = 1$이다. 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{11},\, v_{21}\,, v_{31}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} 2 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \\ v_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

즉, 다음 연립방정식이 성립한다.

$$2v_{11} - v_{21} - v_{31} = 0, \quad 0v_{11} + v_{21} - v_{31} = 0 \tag{8}$$

즉, 특성값 $\lambda_1 = -1$에 대한 특성벡터는 다음과 같다.

$$\boldsymbol{v}_1 = s\ [1,\ 1,\ 1\,]^t \tag{9}$$

여기서 s는 자유모수(free parameter)이다. 특성값 $\lambda_1 = -1$의 기하중복도는 다음과 같다.

$$s_1 = n - \mathrm{rank}(A - \lambda_1 I) = 3 - 2 = 1 \tag{10}$$

다음 식을 만족함을 확인할 수 있다.

$$A\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1 \tag{11}$$

특성값 $\lambda_2 = 2$의 대수중복도는 $k_2 = 2$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_2 = [v_{12},\, v_{22}\,,\ v_{32}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} 1-2 & -1 & -1 \\ -1 & 1-2 & -1 \\ -1 & -1 & 1-2 \end{bmatrix} \begin{bmatrix} v_{12} \\ v_{22} \\ v_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{12}$$

즉, 다음 방정식이 성립한다.

$$v_{12} + v_{22} + v_{32} = 0 \tag{13}$$

즉, 특성값 $\lambda_2 = 1$에 대한 특성벡터들은 다음과 같다.

$$\boldsymbol{v}_2 = s\ [-1,\ 0,\ 1\,]^t, \quad \boldsymbol{v}_3 = u\ [-1,\ 1,\ 0\,]^t \tag{14}$$

여기서 s와 u는 자유모수들이다. 특성값 $\lambda_2 = 1$의 기하중복도는 다음과 같다.

$$s_2 = n - \mathrm{rank}(A - \lambda_2 I) = 3 - 1 = 2 \tag{15}$$

다음 식들을 만족함을 확인할 수 있다.

$$A\boldsymbol{v}_2 = \lambda_2 \boldsymbol{v}_2, \quad A\boldsymbol{v}_3 = \lambda_2 \boldsymbol{v}_3 \tag{16}$$

행렬 J와 H를 다음과 같이 정의하자.

$$J = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \quad H = [\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3] = \begin{bmatrix} 1 & -1 & -1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \tag{17}$$

식 $\det(H) = 3$이 성립하므로, 벡터 $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ 그리고 벡터 $\boldsymbol{v}_3$가 서로 독립임을 알 수 있다. 따라서, 행렬 A를 다음과 같이 분해할 수 있다.

$$A = HJH^{-1} \tag{18}$$

여기서 H^{-1}는 다음과 같다.

$$H^{-1} = \frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & 2 \\ -1 & 2 & -1 \end{bmatrix} \tag{19}$$

식 (17)을 다음과 같이 예증할 수 있다.

$$AH = \begin{bmatrix} -1 & -2 & -2 \\ -1 & 0 & -2 \\ -1 & 2 & 0 \end{bmatrix} = HJ \tag{20}$$

다음 식들이 성립한다.

$$e^{Jt} = \sum_{k=0}^{\infty} \frac{1}{k!} J^k t^k = \begin{bmatrix} \sum_{k=0}^{\infty} \frac{1}{k!}[-1]^k t^k & 0 & 0 \\ 0 & \sum_{k=0}^{\infty} \frac{1}{k!} 2^k t^k & 0 \\ 0 & 0 & \sum_{k=0}^{\infty} \frac{1}{k!} 2^k t^k \end{bmatrix} = \begin{bmatrix} e^{-t} & 0 & 0 \\ 0 & e^{2t} & 0 \\ 0 & 0 & e^{2t} \end{bmatrix} \tag{21}$$

따라서 연립미분방정식 (1)의 일반해는 다음과 같다.

$$\boldsymbol{u}(t) = He^{Jt}H^{-1}\boldsymbol{u}_0 = \frac{1}{3}\begin{bmatrix}1 & -1 & -1\\ 1 & 0 & 1\\ 1 & 1 & 0\end{bmatrix}\begin{bmatrix}e^{-t} & 0 & 0\\ 0 & e^{2t} & 0\\ 0 & 0 & e^{2t}\end{bmatrix}\begin{bmatrix}1 & 1 & 1\\ -1 & -1 & 2\\ -1 & 2 & -1\end{bmatrix}\begin{bmatrix}u_{01}\\ u_{02}\\ u_{03}\end{bmatrix} \tag{22}$$

식 (22)를 정리하면 다음과 같다.

$$\begin{bmatrix}x(t)\\ y(t)\\ z(t)\end{bmatrix} = \begin{bmatrix}e^{-t} & -e^{2t} & -e^{2t}\\ e^{-t} & 0 & e^{2t}\\ e^{-t} & e^{2t} & 0\end{bmatrix}\begin{bmatrix}C_1\\ C_2\\ C_3\end{bmatrix} \tag{23}$$

여기서 C_1, C_2, C_3는 상수들이다.

초기조건 (2)를 식 (23)에 대입하면, 다음 식들이 성립함을 알 수 있다.

$$\begin{bmatrix}C_1\\ C_2\\ C_3\end{bmatrix} = \begin{bmatrix}1 & -1 & -1\\ 1 & 0 & 1\\ 1 & 1 & 0\end{bmatrix}^{-1}\begin{bmatrix}-4\\ 4\\ 3\end{bmatrix} = \frac{1}{3}\begin{bmatrix}1 & 1 & 1\\ -1 & -1 & 2\\ -1 & 2 & -1\end{bmatrix}\begin{bmatrix}-4\\ 4\\ 3\end{bmatrix} = \begin{bmatrix}1\\ 2\\ 3\end{bmatrix} \tag{24}$$

즉, 이 초기값문제의 해가 다음과 같음을 알 수 있다.

$$\begin{bmatrix}x(t)\\ y(t)\\ z(t)\end{bmatrix} = \begin{bmatrix}e^{-t} - 5e^{2t}\\ e^{-t} + 3e^{2t}\\ e^{-t} + 2e^{2t}\end{bmatrix} \tag{25}$$

이 초기값문제를 풀기 위해서, 다음 MATLAB프로그램 Jordan105.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename Jordan105.m
% https://www.math24.net/general-solution-system-differential
%                          -equations-jordan-form/
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan105.txt
JordanD105g = dsolve('Dx=x-y-z','Dy=-x+y-z','Dz=-x-y+z','t')
JordanD105g.x, JordanD105g.y,JordanD105g.z
JordanD105 = dsolve('Dx=x-y-z','Dy=-x+y-z','Dz=-x-y+z', ...
                    'x(0)=-4','y(0)=4','z(0)=3','t')
JordanD105.x, JordanD105.y,JordanD105.z
% Plotting
```

```
tt = linspace(-1.5,-0.1,201);
x1 = double(subs(JordanD105.x,'t',tt));
y1 = double(subs(JordanD105.y,'t',tt));
z1 = double(subs(JordanD105.z,'t',tt));
plot3(x1,y1,z1,'r-','linewidth',2)
set(gca,'fontsize',11,'Fontweigh','bold')
xlabel('x'), ylabel('y'),zlabel('z')
saveas(gcf,'Jordan105.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 Jordan105.m에서는 심볼릭함수 dsolve.m을 사용해서 이 초기값 문제를 푼다. 이 MATLAB프로그램을 실행한 결과, 식 (25)의 해가 올바름을 확인할 수 있다. 또한, 그림 7.8.5이 그려진다.

다음 MATLAB프로그램 Jordan205.m을 실행하면, MATLAB함수 eig.m을 사용해서 이 초기값문제를 푼다.

```
% ------------------------------------------------------------------
% Filename Jordan205.m
% Solve x' = A*x
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan205.txt
syms t x y
A = [ 1 -1 -1 ; -1 1 -1 ; -1 -1 1 ]
u0 = [ -4 4 3 ]'
% A decomposition solution
[ H J ] = jordan(A)
u = H*expm(J*t)*inv(H)*u0
u = vpa(u)
% A direct solution
uu = expm(A*t)*u0
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행한 결과는 MATLAB프로그램 Jordan105.m을 실행한 결과와 같다. ■

예제 7.8.22 Python을 사용해서 예제 7.8.21을 다시 다루기 위해서, 다음 Python프로그램 Jordan105.Py를 실행해 보자.

```
"""
% Filename Jordan105.Py
% Jordan form for Ordinary Differential Equation
% Programmed by CBS
```

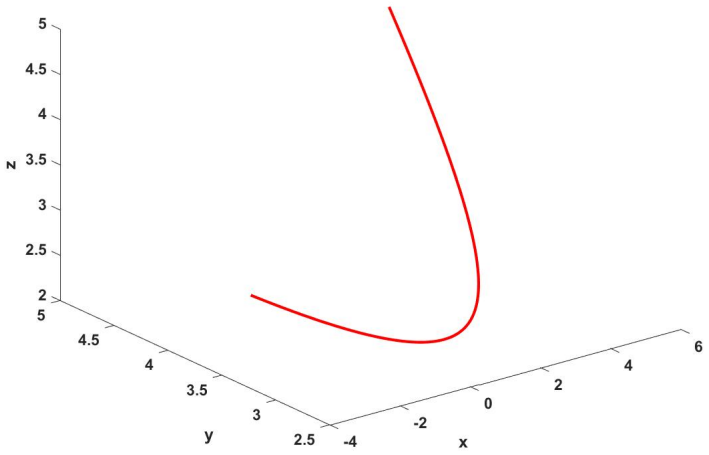


그림 7.8.5. Jordan표준형 미분방정식 V

```
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

x = sy.Function('x')
y = sy.Function('y')
z = sy.Function('z')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t),t), x(t)-y(t)-z(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), -x(t)+y(t)-z(t))
display(eq2)
eq3 = sy.Eq(sy.diff(z(t),t), -x(t)-y(t)+z(t))
display(eq3)
sol13 = sy.dsolve((eq1, eq2, eq3))
display(sol13)
## Initial condition
C1, C2, C3 = sy.symbols("C1, C2, C3")             # generic constants
cond0 = sy.Eq(sol13[0].rhs.subs(t,0),-4)
display(cond0)
cond1 = sy.Eq(sol13[1].rhs.subs(t,0),4)
display(cond1)
cond2 = sy.Eq(sol13[2].rhs.subs(t,0),3)
display(cond2)
constants = sy.solve((cond0,cond1,cond2),{C1,C2,C3})
display(constants)
## Substitute back into solution
xsol13 = sy.expand(sol13[0].rhs.subs(constants))
ysol13 = sy.expand(sol13[1].rhs.subs(constants))
zsol13 = sy.expand(sol13[2].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol13))
display(sy.Eq(y(t),ysol13))
```

```
display(sy.Eq(z(t),zsol13))
## Double checking
from sympy import Matrix
A = np.array([[1,-1,-1], [-1,1,1], [-1,-1,1]])
mA = Matrix(A)
print(mA)
P, J = mA.jordan_form()
print(J)
dumx = sy.diff(xsol13,t)-xsol13+ysol13+zsol13;
print(dumx)
dumy = sy.diff(ysol13,t)+xsol13-ysol13+zsol13;
print(dumy)
dumz = sy.diff(zsol13,t)+xsol13+ysol13-zsol13;
print(dumz)
## Plotting
sol13xLam = sy.lambdify(t,xsol13, modules=['numpy'])
sol13yLam = sy.lambdify(t,ysol13, modules=['numpy'])
sol13zLam = sy.lambdify(t,zsol13, modules=['numpy'])
tt = np.linspace(-10,1,501)
xx = sol13xLam(tt)
yy = sol13yLam(tt)
zz = sol13zLam(tt)
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(111)
ax = Axes3D(fig)
ax.scatter(xx,yy,zz,color='r',marker='*')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
fig.savefig('Jordan105Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.8.21의 결과와 다르다. 이 Python패키지 sympy패키지의 Python함수 dsolve를 사용해서 구한 미분방정식의 해를 다시 미분방정식에 대입하면, 등호가 성립하지 않는다. 즉, 이 해는 올바른 것이 아니다. 이 해는 Jordan블록을 사용한 것과 같은 형태이다. 그러나 이 연립미분방정식의 계수행렬을 대각화할 수 있다. 즉, Jordan행렬을 사용하지 않고 대각행렬을 사용해서 올바른 해를 구할 수 있다. ■

예제 7.8.23 다음 초기값문제를 살펴보자.

$$\dot{x} = -3x - 6y + 6z\,, \quad \dot{y} = x + 6z\,, \quad \dot{z} = -y + 4z \tag{1}$$

$$x(0) = -5\,, \quad y(0) = 6\,, \quad z(0) = 2 \tag{2}$$

미분방정식 (1)과 초기조건 (2)를 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} -5 \\ 6 \\ 2 \end{bmatrix} \tag{3}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}\,, \quad A = \begin{bmatrix} -3 & -6 & 6 \\ 1 & 0 & 6 \\ 0 & -1 & 4 \end{bmatrix} \tag{4}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{vmatrix} -3-\lambda & -6 & 6 \\ 1 & 0-\lambda & 6 \\ 0 & -1 & 4-\lambda \end{vmatrix}$$

$$= \lambda^3 - \lambda^2 = [\lambda - 1]\lambda^2 = 0 \tag{5}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = 1\,, \quad \lambda_2 = 0\,, \quad \lambda_3 = 0 \tag{6}$$

특성값 $\lambda_1 = 1$의 대수중복도는 $k_1 = 1$이다. 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{11},\, v_{21}\,, v_{31}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} -4 & -6 & 6 \\ 1 & -1 & 6 \\ 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{21} \\ v_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

식 (7)의 두 번째 식과 세 번째 식에서 알 수 있듯이, 다음 연립방정식이 성립한다.

$$v_{11} - v_{21} + 6v_{31} = 0, \quad 0v_{11} - v_{21} + 3v_{31} = 0 \tag{8}$$

즉, 특성값 $\lambda_1 = 1$에 대한 특성벡터는 다음과 같다.

$$\boldsymbol{v}_1 = s\ [3,\ -3,\ -1]^t \tag{9}$$

여기서 s는 자유모수이다. 특성값 $\lambda_1 = -1$의 기하중복도는 다음과 같다.

$$s_1 = n - \mathrm{rank}(A - \lambda_1 I) = 3 - 2 = 1 \tag{10}$$

다음 식이 성립함을 확인할 수 있다.

$$A\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1 \tag{11}$$

특성값 $\lambda_2 = 0$의 대수중복도는 $k_2 = 2$이고, 이에 해당하는 특성벡터 $\boldsymbol{v}_2 = [v_{12},\ v_{22}\ ,\ v_{32}]^t$는 다음 식을 만족한다.

$$\begin{bmatrix} -3 & -6 & 6 \\ 1 & 0 & 6 \\ 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} v_{12} \\ v_{22} \\ v_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{12}$$

즉, 다음 방정식이 성립한다.

$$v_{12} + 0v_{22} + 6v_{32} = 0, \quad 0v_{12} - v_{22} + 4v_{32} = 0 \tag{13}$$

즉, 특성값 $\lambda_2 = 0$에 대한 특성벡터는 다음과 같다.

$$\boldsymbol{v}_2 = s\ [-6,\ 4,\ 1\ ]^t \tag{14}$$

여기서 s는 자유모수이다. 특성값 $\lambda_2 = 1$의 기하중복도는 다음과 같다.

$$s_2 = n - \mathrm{rank}(A - \lambda_2 I) = 3 - 2 = 1 \tag{15}$$

다음 식이 성립함을 확인할 수 있다.

$$A\boldsymbol{v}_2 = \lambda_2 \boldsymbol{v}_2 \tag{16}$$

일반화특성벡터 $\boldsymbol{v}_3 = [v_{13},\, v_{23}\,, v_{33}]^t$ 는 다음 식을 만족한다.

$$[A - \lambda_2 I]\boldsymbol{v}_3 = \boldsymbol{v}_2 \tag{17}$$

즉, 다음 식이 성립한다.

$$\begin{bmatrix} -3 & -6 & 6 \\ 1 & 0 & 6 \\ 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} v_{13} \\ v_{23} \\ v_{33} \end{bmatrix} = \begin{bmatrix} -6 \\ 4 \\ 1 \end{bmatrix} \tag{18}$$

즉, 다음 연립방정식이 성립한다.

$$v_{13} + 0v_{23} + 6v_{33} = 4, \quad 0v_{13} - v_{23} + 4v_{33} = 1 \tag{19}$$

식 (19) 에는 자유변수가 하나 있다. 따라서, $v_{33} = 1$ 이라 놓자. 이 값을 식 (19) 에 대입하면, 다음 식이 성립한다.

$$\boldsymbol{v}_3 = [-2,\, 3,\, 1\,]^t \tag{20}$$

벡터 $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ 그리고 벡터 $\boldsymbol{v}_3$ 는 서로 독립이다. 행렬 A 를 다음과 같이 분해할 수 있다.

$$A = HJH^{-1} \tag{21}$$

여기서 J, H 와 H^{-1} 는 각각 다음과 같다.

$$J = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad H = \begin{bmatrix} 3 & -6 & -2 \\ -3 & 4 & 3 \\ -1 & 1 & 1 \end{bmatrix}, \quad H^{-1} = \begin{bmatrix} 1 & 4 & -10 \\ 0 & 1 & -3 \\ 1 & 3 & 6 \end{bmatrix} \tag{22}$$

식 (17) 을 다음과 같이 예증할 수 있다.

$$AH = \begin{bmatrix} 3 & 0 & -6 \\ -3 & 0 & 4 \\ -1 & 0 & 1 \end{bmatrix} = HJ \tag{23}$$

다음 식들이 성립한다.

$$e^{Jt} = \sum_{k=0}^{\infty} \frac{1}{k!} J^k t^k = \begin{bmatrix} \sum_{k=0}^{\infty} \frac{1}{k!} t^k & 0 & 0 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} e^t & 0 & 0 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} \tag{24}$$

따라서 연립미분방정식 (1)의 일반해는 다음과 같다.

$$\boldsymbol{u}(t) = He^{Jt}H^{-1}\boldsymbol{u}_0 = \begin{bmatrix} 3 & -6 & -2 \\ -3 & 4 & 3 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} e^t & 0 & 0 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & -10 \\ 0 & 1 & -3 \\ 1 & 3 & -6 \end{bmatrix} \begin{bmatrix} u_{01} \\ u_{02} \\ u_{03} \end{bmatrix} \tag{25}$$

식 (25)를 정리하면 다음과 같다.

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} 3e^t & -6 & -6t-2 \\ -3e^t & 4 & 4t+3 \\ -e^t & 1 & t+1 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix}, \quad \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 1 & 4 & -10 \\ 0 & 1 & -3 \\ 1 & 3 & -6 \end{bmatrix} \begin{bmatrix} u_{01} \\ u_{02} \\ u_{03} \end{bmatrix} \tag{26}$$

여기서 C_1, C_2, C_3는 상수들이다.

식 (26)을 초기조건 (2)에 대입하면, 다음 식들이 성립한다.

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 3 & -6 & -2 \\ -3 & 4 & 3 \\ -1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} -5 \\ 6 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 & 4 & -10 \\ 0 & 1 & -3 \\ 1 & 3 & -6 \end{bmatrix} \begin{bmatrix} -5 \\ 6 \\ 2 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{27}$$

즉, 이 초기값문제의 해가 다음과 같음을 알 수 있다.

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} -3e^t - 6t - 2 \\ 3e^t + 4t + 3 \\ e^t + t + 1 \end{bmatrix} \tag{28}$$

이 초기값문제를 풀기 위해서, 다음 MATLAB프로그램 Jordan106.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename Jordan106.m
% https://www.math24.net/general-solution-system-differential
```

```
%                          -equations-jordan-form/
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan106.txt
JordanD106g = dsolve('Dx=-3*x-6*y+6*z','Dy=x+6*z','Dz=-y+4*z','t')
JordanD106g.x, JordanD106g.y,JordanD106g.z
JordanD106 = dsolve('Dx=-3*x-6*y+6*z','Dy=x+6*z','Dz=-y+4*z', ...
                    'x(0)=-5','y(0)=6','z(0)=2','t')
JordanD106.x, JordanD106.y,JordanD106.z
% Plotting
tt = linspace(-7,3,201);
x1 = double(subs(JordanD106.x,'t',tt));
y1 = double(subs(JordanD106.y,'t',tt));
z1 = double(subs(JordanD106.z,'t',tt));
plot3(x1,y1,z1,'r-','linewidth',2)
set(gca,'fontsize',11,'Fontweigh','bold')
xlabel('x'), ylabel('y'),zlabel('z')
saveas(gcf,'Jordan106.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 Jordan106.m에서는 심볼릭함수 dsolve.m을 사용해서 이 초기값 문제를 푼다. 이 MATLAB프로그램을 실행한 결과, 식 (28)의 해가 올바름을 확인할 수 있다. 또한, 그림 7.8.6이 그려진다.

다음 MATLAB프로그램 Jordan206.m을 실행하면, MATLAB함수 eig.m을 사용해서 이 초기값문제를 푼다.

```
% ------------------------------------------------------------------
% Filename Jordan206.m
% Solve x' = A*x
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan206.txt
syms t x y
A = [ -3 -6 6; 1 0 6; 0 -1 4 ]
u0 = [ -5 6 2 ]'
% A decomposition solution
[ H J ] = jordan(A)
u = H*expm(J*t)*inv(H)*u0
u = vpa(u)
% A direct solution
uu = expm(A*t)*u0
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행한 결과는 MATLAB프로그램 Jordan106.m을 실행한 결과와 같다. ■

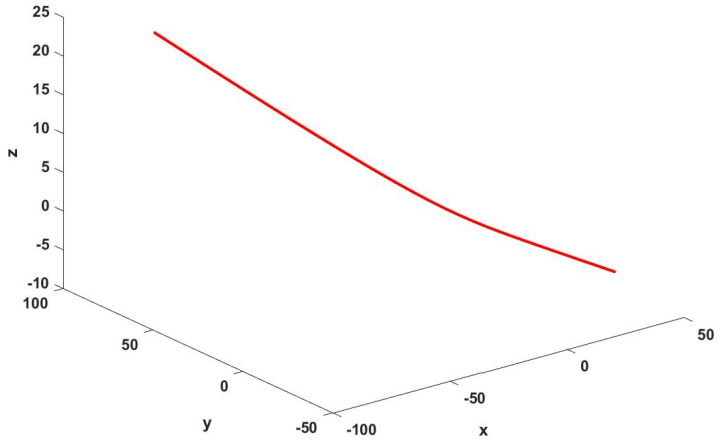


그림 7.8.6. Jordan표준형 미분방정식 VI

예제 7.8.24 Python을 사용해서 예제 7.8.23을 다시 다루기 위해서, 다음 Python프로그램 Jordan106.Py를 실행해 보자.

```
"""
% Filename Jordan106.Py
% Jordan form for Ordinary Differential Equation
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

x = sy.Function('x')
y = sy.Function('y')
z = sy.Function('z')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t),t), -3*x(t)-6*y(t)+6*z(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), x(t)+0*y(t)+6*z(t))
display(eq2)
eq3 = sy.Eq(sy.diff(z(t),t), 0*x(t)-y(t)+4*z(t))
display(eq3)
sol13 = sy.dsolve((eq1, eq2, eq3))
display(sol13)
## Initial condition
C1, C2, C3 = sy.symbols("C1, C2, C3")            # generic constants
cond0 = sy.Eq(sol13[0].rhs.subs(t,0),-5)
display(cond0)
cond1 = sy.Eq(sol13[1].rhs.subs(t,0),6)
display(cond1)
cond2 = sy.Eq(sol13[2].rhs.subs(t,0),2)
display(cond2)
constants = sy.solve((cond0,cond1,cond2),{C1,C2,C3})
```

```
display(constants)
## Substitute back into solution
xsol13 = sy.expand(sol13[0].rhs.subs(constants))
ysol13 = sy.expand(sol13[1].rhs.subs(constants))
zsol13 = sy.expand(sol13[2].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol13))
display(sy.Eq(y(t),ysol13))
display(sy.Eq(z(t),zsol13))
## Double checking
from sympy import Matrix
A = np.array([[1,-1,-1], [-1,1,1], [-1,-1,1]])
mA = Matrix(A)
print(mA)
P, J = mA.jordan_form()
print(J)
dumx = sy.diff(xsol13,t)+3*xsol13+6*ysol13-6*zsol13;
print(dumx)
dumy = sy.diff(ysol13,t)-xsol13-0*ysol13-6*zsol13;
print(dumy)
dumz = sy.diff(zsol13,t)+0*xsol13+ysol13-4*zsol13;
print(dumz)
## Plotting
sol13xLam = sy.lambdify(t,xsol13, modules=['numpy'])
sol13yLam = sy.lambdify(t,ysol13, modules=['numpy'])
sol13zLam = sy.lambdify(t,zsol13, modules=['numpy'])
tt = np.linspace(-7,3,501)
xx = sol13xLam(tt)
yy = sol13yLam(tt)
zz = sol13zLam(tt)
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(111)
ax = Axes3D(fig)
ax.scatter(xx,yy,zz,color='r',marker='*')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
fig.savefig('Jordan106Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.8.23의 결과와 같다. ■

예제 7.8.25 다음 초기값문제를 살펴보자.

$$\dot{x} = 4x + 6y - 15z\,, \quad \dot{y} = x + 3y - 5z\,, \quad \dot{z} = x + 2y - 4z \tag{1}$$

$$x(0) = 8\,, \quad y(0) = -1\,, \quad z(0) = 1 \tag{2}$$

미분방정식 (1)과 초기조건 (2)를 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 8 \\ -1 \\ 1 \end{bmatrix} \tag{3}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}\,, \quad A = \begin{bmatrix} 4 & 6 & -15 \\ 1 & 3 & -5 \\ 1 & 2 & -4 \end{bmatrix} \tag{4}$$

행렬 A의 특성방정식은 다음과 같다.

$$\begin{aligned} f(\lambda) = \det(A - \lambda I) &= \begin{vmatrix} 4-\lambda & 6 & -15 \\ 1 & 3-\lambda & -5 \\ 1 & 2 & -4-\lambda \end{vmatrix} \\ &= \lambda^3 - 3\lambda^2 + 3\lambda - 1 = [\lambda - 1]^3 = 0 \end{aligned} \tag{5}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = 1\,, \quad \lambda_2 = 1\,, \quad \lambda_3 = 1 \tag{6}$$

특성값 $\lambda_1 = 1$의 대수중복도는 $k_1 = 3$이다. 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{11},\, v_{21}\,, v_{31}]^t$는 다음 식을 만족한다.

$$\begin{vmatrix} 4-1 & 6 & -15 \\ 1 & 3-1 & -5 \\ 1 & 2 & -4-1 \end{vmatrix} \begin{bmatrix} v_{11} \\ v_{21} \\ v_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

즉, 다음 방정식이 성립한다.

$$v_{11} + 2v_{21} - 5v_{31} = 0 \tag{8}$$

즉, 특성값 $\lambda_1 = 1$에 대한 특성벡터들 다음과 같다.

$$\boldsymbol{v}_1 = s\,[-2,\,1,\,0]^t\,, \quad \boldsymbol{v}_2 = u\,[5,\,0,\,1]^t \tag{9}$$

여기서 s와 u는 자유모수들이다. 특성값 $\lambda_1 = 1$의 기하중복도는 다음과 같다.

$$s_1 = n - \mathrm{rank}(A - \lambda_1 I) = 3 - 1 = 2 \tag{10}$$

다음 식을 만족함을 확인할 수 있다.

$$A\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1, \quad A\boldsymbol{v}_2 = \lambda_1 \boldsymbol{v}_2 \tag{11}$$

여기서 유의할 점은 $\boldsymbol{v}_1$이나 $\boldsymbol{v}_2$가 반드시 Jordan기저에 속한다고 말할 수는 없다.

지금부터는 2×2 Jordan블록을 생각해보자. 이러한 Jordan블록에 대응하는 3차원 부분공간은 특성벡터 하나(즉, $\boldsymbol{v}_1$ 또는 $\boldsymbol{v}_2$)와 이에 대응하는 일반화특성벡터로 구성되어 있다. 이 벡터들을 각각 $\boldsymbol{u}_1(\in \mathbb{R}^3)$와 $\boldsymbol{u}_2(\in \mathbb{R}^3)$로 표기하면, 다음 식들이 성립해야 한다.

$$[A - \lambda_1 I]\,\boldsymbol{u}_1 = \boldsymbol{0} \tag{12}$$

$$[A - \lambda_1 I]\,\boldsymbol{u}_2 = \boldsymbol{u}_1 \tag{13}$$

식 (12)와 식 (13)으로부터 다음 식이 성립함을 알 수 있다.

$$[A - \lambda_1 I]^2\,\boldsymbol{u}_2 = \boldsymbol{0} \tag{14}$$

또한 다음 식들이 성립함을 쉽게 알 수 있다.

$$[A - \lambda_1 I]^2 = \begin{bmatrix} 3 & 6 & -15 \\ 1 & 2 & -5 \\ 1 & 2 & -5 \end{bmatrix}^2 = O \tag{15}$$

따라서 임의의 벡터 $\boldsymbol{x} \neq \boldsymbol{0}$는 연산자 $[A - \lambda_1 I]^2$의 커널에 속한다. 행렬 $A - \lambda_1 I$의 첫 번째 열이 영벡터가 아니므로, 단위벡터 $\boldsymbol{u}_2 = [1,\,0,\,0]^t$를 특성값 $\lambda_1 = 1$에 해당하는 일반화특성벡터로

선택할 수 있다. 우선 식 (13)을 적용해서, 다음과 같은 벡터 $\boldsymbol{u}_1$을 구할 수 있다.

$$\boldsymbol{u}_1 = [A - \lambda_1 I]\,\boldsymbol{u}_2 = \begin{bmatrix} 3 & 6 & -15 \\ 1 & 2 & -5 \\ 1 & 2 & -5 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 1 \end{bmatrix}. \tag{16}$$

식 (8)에서 알 수 있듯이, 벡터 $\boldsymbol{u}_1$은 행렬 A의 특성벡터이다. 따라서 특성벡터 $\boldsymbol{u}_1$과 일반화 특성벡터 $\boldsymbol{u}_2$는 특성값 1에 해당하는 2×2 Jordan블록과 연관되어 있다. 나머지 특성벡터는 $\boldsymbol{u}_1 = [3,\, 1,\, 1]^t$과 독립인 임의의 벡터이다. 예를 들어, 식 (9)를 바탕으로 $\boldsymbol{v}_2 = [5,\, 0,\, 1]^t$를 나머지 특성벡터로 선택하고, 다음과 같은 행렬을 정의하자.

$$H = \begin{bmatrix} 3 & 1 & 5 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \quad J = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{17}$$

다음 식들이 성립한다.

$$AH = \begin{bmatrix} 4 & 6 & -15 \\ 1 & 3 & -5 \\ 1 & 2 & -4 \end{bmatrix} \begin{bmatrix} 3 & 1 & 5 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 4 & 5 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 5 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = HJ \tag{18}$$

따라서 행렬 A를 다음과 같이 분해할 수 있다.

$$A = HJH^{-1} \tag{19}$$

다음 식들이 성립한다.

$$e^{Jt} = \sum_{k=0}^{\infty} \frac{1}{k!} J^k t^k = \begin{bmatrix} \sum\limits_{k=0}^{\infty} \frac{1}{k!} t^k & \sum\limits_{k=0}^{\infty} \frac{1}{k!} k t^k & 0 \\ 0 & \sum\limits_{k=0}^{\infty} \frac{1}{k!} t^k & 0 \\ 0 & 0 & \sum\limits_{k=0}^{\infty} \frac{1}{k!} t^k \end{bmatrix} = \begin{bmatrix} e^t & te^t & 0 \\ 0 & e^t & 0 \\ 0 & 0 & e^t \end{bmatrix} \tag{20}$$

따라서 연립미분방정식 (1)의 일반해는 다음과 같다.

$$\boldsymbol{u}(t) = He^{Jt}H^{-1}\boldsymbol{u}_0 = \begin{bmatrix} 3 & 1 & 5 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} e^t & te^t & 0 \\ 0 & e^t & 0 \\ 0 & 0 & e^t \end{bmatrix} H^{-1}\boldsymbol{u}_0 \tag{21}$$

여기서 $C = [C_1,\, C_2,\, C_3\,]^t \doteq H^{-1}\boldsymbol{u}_0$ 이다. 식 (21)을 정리하면 다음과 같다.

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} 3e^t & 3te^t + e^t & 5e^t \\ e^t & te^t & 0 \\ e^t & te^t & e^t \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \tag{22}$$

식 (22)를 초기조건 (2)에 대입하면, 다음 식들이 성립한다.

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 3 & 1 & 5 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 8 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & -5 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} 8 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 2 \end{bmatrix} \tag{23}$$

즉, 이 초기값문제의 해가 다음과 같음을 알 수 있다.

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} 8e^t + 3te^t \\ -e^t + te^t \\ e^t + te^t \end{bmatrix} \tag{24}$$

이 초기값문제를 풀기 위해서, 다음 MATLAB프로그램 Jordan107.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename Jordan107.m
% https://www.math24.net/general-solution-system-differential
%                          -equations-jordan-form/
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan107.txt
JordanD107g = dsolve('Dx=4*x+6*y-15*z','Dy=x+3*y-5*z', ...
                     'Dz=x+2*y-4*z','t')
JordanD107g.x, JordanD107g.y,JordanD107g.z
JordanD107 = dsolve('Dx=4*x+6*y-15*z','Dy=x+3*y-5*z', ...
               'Dz=x+2*y-4*z','x(0)=8','y(0)=-1','z(0)=1','t')
JordanD107.x, JordanD107.y,JordanD107.z
% Plotting
tt = linspace(-5,1,201);
```

```
x1 = double(subs(JordanD107.x,'t',tt));
y1 = double(subs(JordanD107.y,'t',tt));
z1 = double(subs(JordanD107.z,'t',tt));
plot3(x1,y1,z1,'r-','linewidth',2)
set(gca,'fontsize',11,'Fontweigh','bold')
xlabel('x'), ylabel('y'),zlabel('z')
saveas(gcf,'Jordan107.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 Jordan107.m에서는 심볼릭함수 dsolve.m을 사용해서 이 초기값 문제를 푼다. 이 MATLAB프로그램을 실행한 결과, 식 (24)의 해가 올바름을 확인할 수 있다. 또한, 그림 7.8.7이 그려진다.

다음 MATLAB프로그램 Jordan207.m을 실행하면, MATLAB함수 eig.m을 사용해서 이 초기값문제를 푼다.

```
% ------------------------------------------------------------------
% Filename Jordan207.m
% Solve x' = A*x
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan207.txt
syms t x y
A = [ 4 6 -15; 1 3 -5; 1 2 -4 ]
u0 = [ 8 -1 1 ]'
% A decomposition solution
[ H J ] = jordan(A)
u = H*expm(J*t)*inv(H)*u0
u = vpa(u)
% A direct solution
uu = expm(A*t)*u0
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행한 결과는 MATLAB프로그램 Jordan107.m을 실행한 결과와 같다. ■

예제 7.8.26 Python을 사용해서 예제 7.8.25를 다시 다루기 위해서, 다음 Python프로그램 Jordan107.Py를 실행해 보자.

```
"""
% Filename Jordan107.Py
% Jordan form for Ordinary Differential Equation
% Programmed by CBS
"""
```

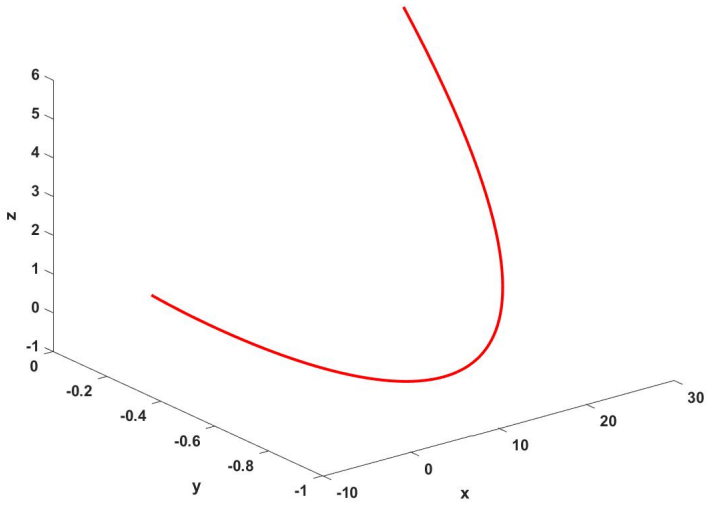


그림 7.8.7. Jordan표준형 미분방정식 VII

```
6
7  from IPython.display import display
8  import numpy as np
9  import sympy as sy
10 import matplotlib.pyplot as plt
11 sy.init_printing()  # LaTeX-like pretty printing for IPython
12
13 x = sy.Function('x')
14 y = sy.Function('y')
15 z = sy.Function('z')
16 t = sy.symbols('t')
17 eq1 = sy.Eq(sy.diff(x(t),t), 4*x(t)+6*y(t)-15*z(t))
18 display(eq1)
19 eq2 = sy.Eq(sy.diff(y(t),t), x(t)+3*y(t)-5*z(t))
20 display(eq2)
21 eq3 = sy.Eq(sy.diff(z(t),t), x(t)+2*y(t)-4*z(t))
22 display(eq3)
23 # Double Check
24 from sympy import Matrix
25 A = np.array([[4,6,-15], [1,3,-5], [1,2,-4]])
26 mA = Matrix(A)
27 print(mA)
28 P, J = mA.jordan_form()
29 print(J)
30 # Solving Simultaneous ODE
31 # sol13 = sy.dsolve((eq1, eq2, eq3))
32 # display(sol13)
33 C1, C2, C3 = sy.symbols("C1, C2, C3")
34 solX = C1*(3*sy.exp(t))+C2*(3*t*sy.exp(t)+sy.exp(t))+C3*(5*sy.exp(t))
35 solY = C1*(sy.exp(t))+C2*(t*sy.exp(t))+C3*(0*sy.exp(t))
36 solZ = C1*(sy.exp(t))+C2*(t*sy.exp(t))+C3*(1*sy.exp(t))
37 print(solX,solY,solZ)
38 ## Initial condition
39 C1, C2, C3 = sy.symbols("C1, C2, C3")            # generic constants
40 cond0 = sy.Eq(solX.subs(t,0),8)
41 display(cond0)
42 cond1 = sy.Eq(solY.subs(t,0),-1)
```

```
display(cond1)
cond2 = sy.Eq(solZ.subs(t,0),1)
display(cond2)
constants = sy.solve((cond0,cond1,cond2),{C1,C2,C3})
display(constants)
## Substitute back into solution
xsol13 = sy.expand(solX.subs(constants))
ysol13 = sy.expand(solY.subs(constants))
zsol13 = sy.expand(solZ.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol13))
display(sy.Eq(y(t),ysol13))
display(sy.Eq(z(t),zsol13))
## Double checking
dumx = sy.diff(xsol13,t)-4*xsol13-6*ysol13+15*zsol13;
print(dumx)
dumy = sy.diff(ysol13,t)-xsol13-3*ysol13+5*zsol13;
print(dumy)
dumz = sy.diff(zsol13,t)-xsol13-2*ysol13+4*zsol13;
print(dumz)
## Plotting
sol13xLam = sy.lambdify(t,xsol13, modules=['numpy'])
sol13yLam = sy.lambdify(t,ysol13, modules=['numpy'])
sol13zLam = sy.lambdify(t,zsol13, modules=['numpy'])
tt = np.linspace(-5,1,501)
xx = sol13xLam(tt)
yy = sol13yLam(tt)
zz = sol13zLam(tt)
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(111)
ax = Axes3D(fig)
ax.scatter(xx,yy,zz,color='r',marker='*')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
fig.savefig('Jordan107Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.8.25의 결과와 다르다. 이 Python패키지 sympy의 Python함수 dsolve를 사용해서 이 미분방정식의 해를 구할 수가 없었다. 이 Python 프로그램 Jordan107.Py에서는 식 (22)의 일반해를 사용해서 특수해를 구했다. ■

예제 7.8.27 다음 초기값문제를 살펴보자.

$$\dot{x} = -7x - 5y - 3z\,, \quad \dot{y} = 2x - 2y - 3z\,, \quad \dot{z} = y \tag{1}$$

$$x(0) = 4\,, \quad y(0) = -4\,, \quad z(0) = 2 \tag{2}$$

미분방정식 (1)과 초기조건 (2)를 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{u}} = A\boldsymbol{u}\,, \quad \boldsymbol{u}_0 = \begin{bmatrix} 4 \\ -4 \\ 2 \end{bmatrix} \tag{3}$$

여기서 벡터 $\boldsymbol{u}$와 행렬 A는 다음과 같다.

$$\boldsymbol{u} = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}\,, \quad A = \begin{bmatrix} -7 & -5 & -3 \\ 2 & -2 & -3 \\ 0 & 1 & 0 \end{bmatrix} \tag{4}$$

행렬 A의 특성방정식은 다음과 같다.

$$f(\lambda) = \det(A - \lambda I) = \begin{bmatrix} -7-\lambda & -5 & -3 \\ 2 & -2-\lambda & -3 \\ 0 & 1 & 0-\lambda \end{bmatrix}$$

$$= -\left[\lambda^3 + 9\lambda^2 + 27\lambda + 27\right] = -[\lambda+3]^3 = 0 \tag{5}$$

따라서 행렬 A의 특성값들은 다음과 같다.

$$\lambda_1 = -3\,, \quad \lambda_2 = -3\,, \quad \lambda_3 = -3 \tag{6}$$

특성값 $\lambda_1 = -3$의 대수중복도는 $k_1 = 3$이다. 이에 해당하는 특성벡터 $\boldsymbol{v}_1 = [v_{11},\, v_{21}\,, v_{31}]^t$는 다음 식을 만족한다.

$$\begin{vmatrix} -7+3 & -5 & -3 \\ 2 & -2+3 & -3 \\ 0 & 1 & 0+3 \end{vmatrix} \begin{bmatrix} v_{11} \\ v_{21} \\ v_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

즉, 다음 연립방정식이 성립한다.

$$2v_{11} + v_{21} - 3v_{31} = 0, \quad 0v_{11} + v_{21} + 3v_{31} = 0 \tag{8}$$

특성값 $\lambda_1 = -3$의 기하중복도는 다음과 같다.

$$s_1 = n - \mathrm{rank}(A - \lambda_1 I) = 3 - 2 = 1 \tag{9}$$

따라서, 특성값 $\lambda_1 = -3$에 해당하는 부분공간은 특성벡터 $\boldsymbol{v}_1$와 일반화특성벡터들 $\boldsymbol{v}_2$와 $\boldsymbol{v}_3$에 의해서 생성된다. 따라서 이 벡터들은 서로 독립이며, 다음 식들을 만족한다.

$$[A - \lambda_1 I]\,\boldsymbol{v}_1 = \boldsymbol{0} \tag{10}$$

$$[A - \lambda_1 I]\,\boldsymbol{v}_2 = \boldsymbol{v}_1 \tag{11}$$

$$[A - \lambda_1 I]\,\boldsymbol{v}_3 = \boldsymbol{v}_2 \tag{12}$$

식 (10)∼식 (12)로부터 다음 식이 성립함을 알 수 있다.

$$[A - \lambda_1 I]^3\,\boldsymbol{v}_3 = \boldsymbol{0} \tag{13}$$

다음 식들이 성립한다.

$$[A - \lambda_1 I]^3 = [A - \lambda_1 I]^2\,[A - \lambda_1 I] = \begin{bmatrix} 6 & 12 & 18 \\ -6 & -12 & -18 \\ 2 & 4 & 6 \end{bmatrix} \begin{bmatrix} -4 & -5 & -3 \\ 2 & 1 & -3 \\ 0 & 1 & 3 \end{bmatrix} = O \tag{14}$$

지금부터 특성벡터 $\boldsymbol{v}_1$을 구하기로 하자. 식 (13)에서 알 수 있듯이, $\boldsymbol{v}_3$로 임의의 벡터를 사용할 수 있다. 따라서, 다음 벡터를 선택하기로 하자.

$$\boldsymbol{v}_3 = [1\,,0\,,0\,]^t \tag{15}$$

식 (12)에서 알 수 있듯이, 다음 식이 성립한다.

$$\boldsymbol{v}_2 = [A - \lambda_1 I]\,\boldsymbol{v}_3 = [-4\,,2\,,0\,]^t \tag{16}$$

식 (11)에서 알 수 있듯이, 다음 식이 성립한다.

$$\boldsymbol{v}_1 = [A - \lambda_1 I]\,\boldsymbol{v}_2 = [6\,,-6\,,2\,]^t \tag{17}$$

행렬 H를 다음과 같이 정의하자.

$$H \doteq [\boldsymbol{v}_1\,, \boldsymbol{v}_2\,, \boldsymbol{v}_3\,] = \begin{bmatrix} 6 & -4 & 1 \\ -6 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix} \tag{18}$$

다음 식들이 성립한다.

$$H^{-1}AH = \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{3}{2} \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} -7 & -5 & -3 \\ 2 & -2 & -3 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 6 & -4 & 1 \\ -6 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix} = \begin{bmatrix} -3 & 1 & 0 \\ 0 & -3 & 1 \\ 0 & 0 & -3 \end{bmatrix} \tag{19}$$

식 (19)의 우변의 행렬을 J로 표기하면, 행렬 A를 다음과 같이 분해할 수 있다.

$$A = HJH^{-1} \tag{20}$$

다음 식이 성립한다.

$$e^{Jt} = e^{-3t} \begin{bmatrix} 1 & t & \frac{1}{2}t^2 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} \tag{21}$$

따라서 연립미분방정식 (1)의 일반해는 다음과 같다.

$$\boldsymbol{v}(t) = He^{Jt}H^{-1}\boldsymbol{v}_0 = \begin{bmatrix} 6 & -4 & 1 \\ -6 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix} e^{-3t} \begin{bmatrix} 1 & t & \frac{1}{2}t^2 \\ 0 & 1 & t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{3}{2} \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} u_{01} \\ u_{02} \\ u_{03} \end{bmatrix} \tag{22}$$

식 (22)를 정리하면 다음과 같다.

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = e^{-3t} \begin{bmatrix} 6 & 6t-4 & 3t^2-4t+1 \\ -6 & -6t+2 & -3t^2+2t \\ 2 & 2t & t^2 \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \tag{23}$$

여기서 C_1, C_2, C_3는 상수들이다.

식 (23)을 초기조건 (2)에 대입하면, 다음 식들이 성립한다.

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} 6 & -4 & 1 \\ -6 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 4 \\ -4 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{3}{2} \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 4 \\ -4 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \tag{24}$$

즉, 이 초기값문제의 해가 다음과 같음을 알 수 있다.

$$\begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = e^{-3t} \begin{bmatrix} 6t^2 - 2t + 4 \\ -6t^2 - 2t - 4 \\ 2t^2 + 2t + 2 \end{bmatrix} \tag{25}$$

이 초기값문제를 풀기 위해서, 다음 MATLAB프로그램 Jordan108.m을 실행하라.

```
% ------------------------------------------------------------------
% Filename Jordan108.m
% https://www.math24.net/general-solution-system-differential
%                          -equations-jordan-form/
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan108.txt
JordanD108g = dsolve('Dx=-7*x-5*y-3*z','Dy=2*x-2*y-3*z','Dz=y','t')
JordanD108g.x, JordanD108g.y,JordanD108g.z
JordanD108 = dsolve('Dx=-7*x-5*y-3*z','Dy=2*x-2*y-3*z','Dz=y', ...
                    'x(0)=4','y(0)=-4','z(0)=2','t')
JordanD108.x, JordanD108.y,JordanD108.z
% Plotting
tt = linspace(-0.3,3,201);
x1 = double(subs(JordanD108.x,'t',tt));
y1 = double(subs(JordanD108.y,'t',tt));
z1 = double(subs(JordanD108.z,'t',tt));
plot3(x1,y1,z1,'r-','linewidth',2)
set(gca,'fontsize',11,'Fontweigh','bold')
xlabel('x'), ylabel('y'),zlabel('z')
saveas(gcf,'Jordan108.jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 Jordan108.m에서는 심볼릭함수 dsolve.m을 사용해서 이 초기값문제를 푼다. 이 MATLAB프로그램을 실행한 결과, 식 (25)의 해가 올바름을 확인할 수 있다. 또한, 그림 7.8.8이 그려진다.

다음 MATLAB프로그램 Jordan208.m을 실행하면, MATLAB함수 eig.m을 사용해서 이 초기값문제를 푼다.

```
% ------------------------------------------------------------------
% Filename Jordan208.m
% Solve x' = A*x
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary Jordan208.txt
syms t x y
A = [ -7 -5 -3; 2 -2 -3; 0 1 0 ]
u0 = [ 4 -4 2 ]'
% A decomposition solution
[ H J ] = jordan(A)
u = H*expm(J*t)*inv(H)*u0
u = vpa(u)
% A direct solution
uu = expm(A*t)*u0
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행한 결과는 MATLAB프로그램 Jordan108.m을 실행한 결과와 같다. ■

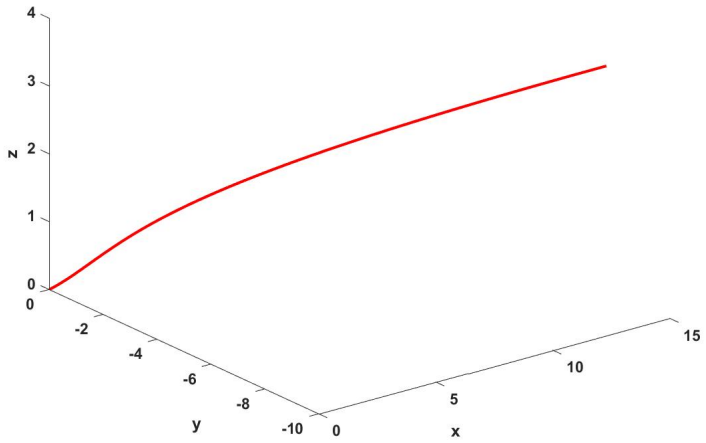


그림 7.8.8. Jordan표준형 미분방정식 VIII

예제 7.8.28 Python을 사용해서 예제 7.8.27을 다시 다루기 위해서, 다음 Python프로그램 Jordan108.Py를 실행해 보자.

```
"""
% Filename Jordan108.Py
% Jordan form for Ordinary Differential Equation
```

```
% Programmed by CBS
"""

from IPython.display import display
import numpy as np
import sympy as sy
import matplotlib.pyplot as plt
sy.init_printing()  # LaTeX-like pretty printing for IPython

x = sy.Function('x')
y = sy.Function('y')
z = sy.Function('z')
t = sy.symbols('t')
eq1 = sy.Eq(sy.diff(x(t),t), -7*x(t)-5*y(t)-3*z(t))
display(eq1)
eq2 = sy.Eq(sy.diff(y(t),t), 2*x(t)-2*y(t)-3*z(t))
display(eq2)
eq3 = sy.Eq(sy.diff(z(t),t), 0*x(t)+1*y(t)+0*z(t))
display(eq3)
sol13 = sy.dsolve((eq1, eq2, eq3))
display(sol13)
## Initial condition
C1, C2, C3 = sy.symbols("C1, C2, C3")            # generic constants
cond0 = sy.Eq(sol13[0].rhs.subs(t,0),4)
display(cond0)
cond1 = sy.Eq(sol13[1].rhs.subs(t,0),-4)
display(cond1)
cond2 = sy.Eq(sol13[2].rhs.subs(t,0),2)
display(cond2)
constants = sy.solve((cond0,cond1,cond2),{C1,C2,C3})
display(constants)
## Substitute back into solution
xsol13 = sy.expand(sol13[0].rhs.subs(constants))
ysol13 = sy.expand(sol13[1].rhs.subs(constants))
zsol13 = sy.expand(sol13[2].rhs.subs(constants))
print("Solution with initial conditions:")
display(sy.Eq(x(t),xsol13))
display(sy.Eq(y(t),ysol13))
display(sy.Eq(z(t),zsol13))
## Double checking
from sympy import Matrix
A = np.array([[-7,-5,-3], [2,-2,-3], [0,1,0]])
mA = Matrix(A)
print(mA)
P, J = mA.jordan_form()
print(J)
dumx = sy.diff(xsol13,t)+7*xsol13+5*ysol13+3*zsol13;
print(dumx)
dumy = sy.diff(ysol13,t)-2*xsol13+2*ysol13+3*zsol13;
print(dumy)
dumz = sy.diff(zsol13,t)+0*xsol13-ysol13-0*zsol13;
print(dumz)
## Plotting
sol13xLam = sy.lambdify(t,xsol13, modules=['numpy'])
sol13yLam = sy.lambdify(t,ysol13, modules=['numpy'])
sol13zLam = sy.lambdify(t,zsol13, modules=['numpy'])
tt = np.linspace(-0.3,3,501)
xx = sol13xLam(tt)
yy = sol13yLam(tt)
zz = sol13zLam(tt)
from mpl_toolkits.mplot3d import Axes3D
```

```
fig = plt.figure(111)
ax = Axes3D(fig)
ax.scatter(xx,yy,zz,color='r',marker='*')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
fig.savefig('Jordan108Py.png')

# End of Program
```

이 Python프로그램을 수행한 결과는 예제 7.8.27의 결과와 같다. ■

제7.9절 예제들

7.9.1 비선형미분방정식과 Newton-Raphson법

다음 경계값문제를 살펴보자.

$$\frac{d^2u}{dx^2} + \lambda e^u = 0, \ (0 < x < 1), \quad u(0) = 0, \ \ u(1) = 0 \tag{7.9.1}$$

여기서 λ는 상수이다.

이 경계값문제를 풀기 위해서 다음 차분방정식을 이용하기로 하자.

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \lambda \exp(u_i) = 0, \ \ (i = 1, 2, \cdots, n) \tag{7.9.2}$$

여기서 $u_i = u(x_i)$이고 x_i는 다음과 같다.

$$x_i \doteq ih, \ (i = 1, 2, \cdots, n), \quad h \doteq \frac{1}{n+1} \tag{7.9.3}$$

식 (7.9.2)의 경계조건을 다음과 같이 쓰는 것은 타당하다.

$$u_0 = 0, \quad u_{n+1} = 0 \tag{7.9.4}$$

편의상 $\boldsymbol{u}$와 $f(\boldsymbol{u})$를 다음과 같이 정의하자.

$$\boldsymbol{u} \doteq \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix}, \quad \boldsymbol{f}(\boldsymbol{u}) \doteq \begin{bmatrix} f_1(\boldsymbol{u}) \\ f_2(\boldsymbol{u}) \\ \vdots \\ f_n(\boldsymbol{u}) \end{bmatrix} = \begin{bmatrix} \frac{u_2-2u_1+u_0}{h^2} + \lambda \exp(u_1) \\ \frac{u_3-2u_2+u_1}{h^2} + \lambda \exp(u_2) \\ \vdots \\ \frac{u_{n+1}-2u_n+u_{n-1}}{h^2} + \lambda \exp(u_n) \end{bmatrix} \tag{7.9.5}$$

Jacobian행렬은 다음과 같다.

$$\begin{aligned} J_f(\boldsymbol{u}) &= \left[\frac{\partial f_i(\boldsymbol{u})}{\partial u_j} \right] \\ &= \begin{bmatrix} \lambda e^{u_1} - \frac{2}{h^2} & \frac{1}{h^2} & 0 & \cdots & 0 & 0 \\ 0 & \lambda e^{u_2} - \frac{2}{h^2} & \frac{1}{h^2} & \cdots & 0 & 0 \\ 0 & 0 & \lambda e^{u_3} - \frac{2}{h^2} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda e^{u_{n-1}} - \frac{2}{h^2} & \frac{2}{h^2} \\ 0 & 0 & 0 & \cdots & \frac{2}{h^2} & \lambda e^{u_n} - \frac{2}{h^2} \end{bmatrix} \end{aligned} \tag{7.9.6}$$

Newton-Raphson알고리즘은 다음과 같다.

$$\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} - \left[J_f\left(\boldsymbol{u}^{(k)}\right) \right]^{-1} f\left(\boldsymbol{u}^{(k)}\right) \tag{7.9.7}$$

예제 7.9.1 경계값문제 (7.9.1)을 풀기 위해서 Newton-Raphson알고리즘 (7.9.7)을 적용하기로 하자. 이 Newton-Raphson알고리즘의 초기함수로 다음 식을 사용하기로 하자.

$$v_\alpha(x) = \alpha x[1-x], \quad \alpha = 20 \tag{1}$$

경계값문제 (7.9.1)를 풀기 위해서, 다음 MATLAB프로그램 NonlinearODE101DY.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: NonlinearODE101DY.m
%  Boundary Value Problem for Bratu's equation Using Newton's mehtod
%  Programmed by CDY
%------------------------------------------------------------------------
clear all, close all
lambda = 1;
```

```
n = 24;
h = 1/(n+1);
x = (h*(1:n))';
f = zeros(n,1);
MaxIter = 10000;
alpha = 20
v = alpha.*x.*(1-x);
uold = v;
stockkk = zeros(10,1);
Jacob = zeros(n,n);
f = zeros(n,1);
for kk = 1:MaxIter
       % Initial vector
       uoldp1 = [ uold(2:n); 0];
       uoldm1 = [ 0; uold(1:n-1) ];
       f = (uoldp1-2*uold+uoldm1)/h^2 + lambda*exp(uold);
       for jj1 = 1:n
            Jacob(jj1,jj1) = lambda*exp(uold(jj1))-2/h^2;
       end % end-for-jj1
       for jj2 = 1:n-1
        Jacob(jj2,jj2+1) = 1/h^2;
        Jacob(jj2+1,jj2) = 1/h^2;
       end % end-for-jj2
       if abs(det(Jacob)) > 10^100  % for Part 2
          disp('kk'), disp(kk)
          detJacob = det(Jacob);
          disp('|Jacob|'), disp(detJacob)
          disp('1/|Jacob|'), disp(1/detJacob')
           break
        end  % end-if
        unew = uold - Jacob\f;
        RelErr = norm(unew-uold)/norm(uold);
        if RelErr < 10^(-7)
            break
        end % end-if
        uold = unew;
end  % end-for-kk
plot([0 ; x; 1], [0; v; 0 ],'r--','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold','ylim',[0 6])
hold on
plot([0 ; x; 1], [0; unew; 0 ],'k-','LineWidth',2);
legend('Initial function','Newton Solution','location','NE')
hold off
xlabel('\bf x'); ylabel('\bf u','rotation',0)
saveas(gcf,'NonlinearODE101DY.jpg')
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 NonlinearODE101DY.m을 실행하면, 그림 7.9.1이 그려진다. 다양한 초기값 α에 해당하는 초기함수 $v_\alpha(x)$를 사용해서 Newton-Raphson알고리즘을 적용하면, 해가 많이 달라짐을 알 수 있다. ■

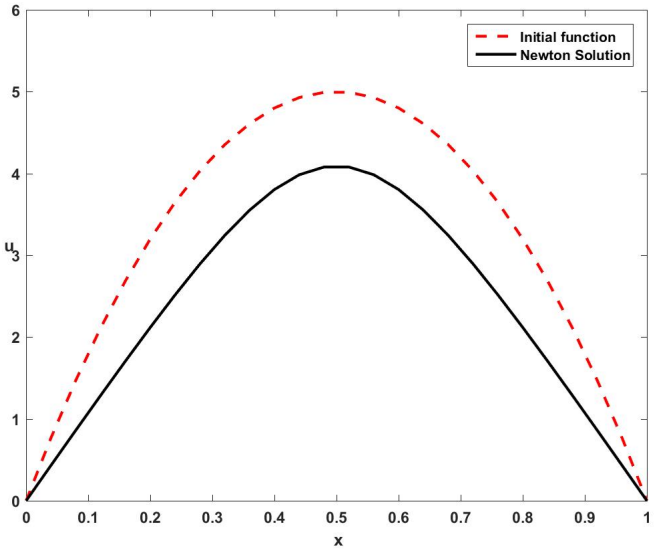


그림 7.9.1. 경계값문제와 Newton-Raphson알고리즘

7.9.2 비선형미분방정식과 Gauss소거법

다음 경계값문제를 살펴보자.

$$\frac{d^2u}{dx^2} + e^x = 0, \quad u(x) > 0, \quad (0 < x < 1), \quad u(0) = 0, \quad u(1) = 0 \tag{7.9.8}$$

이 미분방정식의 해가 $u(x) = a + bx + c\exp(x)$ 임을 추측할 수가 있다. 이 함수 $u(x)$ 를 주어진 경계조건들에 대입하면, 진짜해 $u^*(x)$ 가 다음과 같음을 알 수 있다.

$$u^*(x) = 1 - e^x + [e-1]x, \quad (0 < x < 1) \tag{7.9.9}$$

이 경계값문제를 풀기 위해서 다음 차분방정식을 이용하기로 하자.

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + \exp(x_i) = 0, \quad (i = 1, 2, \cdots, n) \tag{7.9.10}$$

여기서 $u_i = u(x_i)$ 이고, x_i 는 다음과 같다.

$$x_i \doteq ih, \ (i = 1, 2, \cdots, n), \qquad h \doteq \frac{1}{n+1} \tag{7.9.11}$$

또한, 식 $u_0 = 0$ 와 식 $u_{n+1} = 0$ 가 성립한다. 식 (7.9.10) 을 다음과 같이 쓸 수 있다.

$$A\boldsymbol{u} = h^2\boldsymbol{f}(\boldsymbol{x}) \tag{7.9.12}$$

여기서 벡터들 $\boldsymbol{u}$와 $\boldsymbol{f}(\boldsymbol{x})$와 행렬 A는 각각 다음과 같다.

$$\boldsymbol{u} \doteq [u_1, u_2, \cdots, u_n]^t \qquad \boldsymbol{f}(\boldsymbol{x}) \doteq [\exp(x_1), \exp(x_2), \cdots, \exp(x_n)]^t \tag{7.9.13}$$

$$A \doteq \begin{bmatrix} 2 & -1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & -1 & 2 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \tag{7.9.14}$$

예제 7.9.2 경계값문제 (7.9.8)을 풀기 위해서 식 (7.9.12)에 Gauss소거법을 적용하기로 하자. 다음 MATLAB프로그램 NonlinearOdeGauss101DY.m을 실행하라.

```
% ------------------------------------------------------------------------
%  Filename: NonlinearOdeGauss101DY.m
%  Boundary Value Problem for Bratu's equation
%  Error Analysis using Gauss Elimination
%  Programmed by CDY
%------------------------------------------------------------------------
clear all, close all
lambda = 1;
for ii=1:3
    n = 10^(ii+1)
    h(ii) = 1/(n+1)
    x = (1:n)'*h(ii);
    eVec = ones(n,1);
    A = spdiags([ -eVec 2*eVec -eVec ],[-1 0 1],n,n);
    b = h(ii)^2*exp(x);
    u = A\b;                                % Matlab solution
    ustar = (1-exp(x)) + (exp(1)-1)*x;      % True value
    RelErr(ii) = norm(u-ustar)/norm(u);     %  Relative Error
end
RelErr
error = ustar - u;
% Plotting
subplot(2,2,1)
plot(x,ustar,'k:','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf Exact Solution',2)
subplot(2,2,2)
plot(x,u,'r-','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf Gauss Elimination',2)
subplot(2,2,3)
plot(x,error,'b-.','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf Error',2,'location','SE')
```

```
subplot(2,2,4)
loglog(h,RelErr,'LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold','xDir','reverse')
saveas(gcf,'NonlinearOdeGauss101DY.jpg')
% End of Program
% --------------------------------------------------------------------------
```

이 MATLAB프로그램 NonlinearOdeGauss101DY.m을 실행하면, 그림 7.9.2가 그려진다. 그림 7.9.2의 좌측상단 그래프는 진짜해를 그린 것이고, 우측상단 그래프는 Gauss소거법에 의한 수치해를 그린 것이고, 좌측하단 그래프는 진짜해에서 이 수치해를 뺀 오차를 그린 것이다. 이 그래프들에서 알 수 있듯이, 상대오차가 $1.8 \cdot 10^{-10}$ 보다 작다.

다음 식들이 성립한다.

$$\frac{u(x+h)-2u(x)+u(x-h)}{h^2} - u''(x) = \frac{1}{12}u^{(4)}(x)h^2 = O\left(h^2\right) \tag{1}$$

따라서 다음 식이 성립한다.

$$\max\left|\frac{u(x+h)-2u(x)+u(x-h)}{h^2} - u''(x)\right| \approx c_1 h^2 \tag{2}$$

여기서 c_1 은 상수이다. 즉, 다음 식이 성립한다.

$$\max\left|\frac{d^2u}{dx^2} - \frac{\delta^2 u}{\delta x^2}\right| \approx c_1 h^2 \tag{3}$$

함수 $u^*(x)$가 점 $x=a$에서 최대값을 갖는다고 하자. 만약 $|h|$가 충분히 작으면, 식 $\tilde{u}(a) = u^*(a+\alpha h)$를 만족하는 α가 존재한다. 다음 식들이 성립한다.

$$\begin{aligned}\tilde{u}(a) - u^*(a) &= u^*(a+\alpha h) - u^*(a) \\ &= \alpha[u^*(a)]'h + \frac{1}{2!}\alpha^2[u^*(a)]'' + O\left(h^3\right) \approx \alpha\frac{1}{2!}\alpha^2\nabla u(a)\end{aligned} \tag{4}$$

식 (4)에서 알 수 있듯이, 다음 식을 만족하는 상수들 $c_1, c_2, \cdots, c_n$이 존재한다.

$$\tilde{\boldsymbol{u}}(a) - \boldsymbol{u}^*(a) = \left[c_1h^2, c_2h^2, \cdots, c_nh^2\right]^t \tag{5}$$

식 (5)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\|\tilde{\boldsymbol{u}}(a) - \boldsymbol{u}^*(a)\|_2 = \sqrt{\sum_{i=1}^{n} c_i^2 h^4} = \sqrt{\bar{c^2}\, nh^4} = O\left(h^{1.5}\right) \tag{6}$$

여기서 두 번째 등호는 식 $\bar{c^2} \doteq \sum_{i=1}^{n} c_i^2$에 의해서, 세 번째 등호는 식 $h[n+1] = 1$에 의해서 성립한다. 식 (6)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{\|\tilde{\boldsymbol{u}}(a) - \boldsymbol{u}^*(a)\|_2}{\|\boldsymbol{u}^*(a)\|_2} = O\left(h^{1.5}\right) \tag{7}$$

식 (5)에서 알 수 있듯이, 다음 식이 성립한다.

$$\|\tilde{\boldsymbol{u}}(a) - \boldsymbol{u}^*(a)\|_\infty = \max_{1 \le i \le n} c_i\, h^2 \tag{8}$$

식 (8)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{\|\tilde{\boldsymbol{u}}(a) - \boldsymbol{u}^*(a)\|_\infty}{\|\boldsymbol{u}^*(a)\|_\infty} = O\left(h^2\right) \tag{9}$$

식 (7)에서 알 수 있듯이 l^2-노름에 의한 상대오차는 $O\left(h^{1.5}\right)$이고, 식 (9)에서 알 수 있듯이 $l^\infty-$노름에 의한 상대오차는 $O\left(h^2\right)$이다. 그러나, l^2-노름에 의한 상대오차와 $l^\infty-$노름에 의한 상대오차가 동치성(equivalence of normal property)을 갖는다. 따라서 다음 식이 성립한다고 할 수 있다.

$$\ln \frac{\|\tilde{\boldsymbol{u}}(a) - \boldsymbol{u}^*(a)\|_2}{\|\boldsymbol{u}^*(a)\|_2} = 2 \ln h + c_0 \tag{10}$$

여기서 c_0는 상수이다. 그림 7.9.2의 우측하단의 X축에는 이웃하는 점들 사이의 거리 h를, 그리고 Y축에는 $l^\infty-$노름에 의한 상대오차를 로그로그그래프에 그린 것이다. 이 그래프에서 기울기는 2이다. 이는 식 (10)이 성립함을 의미한다. ■

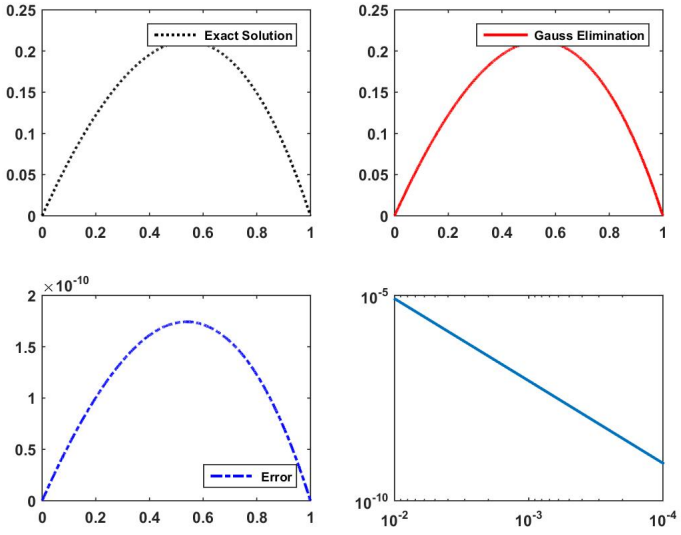


그림 7.9.2. 경계값문제와 Newton-Raphson알고리즘

7.9.3 Gauss소거법과 Hessenberg행렬

다음 정방행렬 $A_{N\times N}$을 살펴보자.

$$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1,n-1} & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2,n-1} & a_{2n} \\ 0 & a_{32} & a_{33} & a_{34} & \cdots & a_{3,n-1} & a_{3n} \\ 0 & 0 & a_{43} & a_{44} & \cdots & a_{4,n-1} & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & 0 & \cdots & a_{n,n-1} & a_{n,n} \end{bmatrix} \tag{7.9.15}$$

행렬 A는 좌측하단의 $[N-1]\times[N-1]$부행렬이 상삼각행렬이다. 이러한 행렬 A를 Hessenberg 행렬이라 부른다.

정방행렬 $A_{N\times N}$에 대해서 Q^tAQ가 Hessenberg행렬이 되는 직교행렬 Q가 존재함이 알려져 있다. 정방행렬 $A_{N\times N}$을 Hessenberg행렬로 만드는데 필요한 계산량은 $10N^3/3$ flops 이다. 정방행렬 $A_{N\times N}$가 대칭이면, 해당 Hessenberg행렬은 삼대각행렬(tridiagonal matrix) 이고 Hessenberg행렬로 만드는데 필요한 계산량은 $4N^3/3$ flops이다.

예제 7.9.3 식 (7.9.14)의 A는 삼대각행렬이다. 따라서, A는 Hessenberg행렬이다. Gauss 소거법을 사용해서 방정식 (7.9.12)를 풀기위해서, 다음 MATLAB프로그램 GaussHessenberg101DY.m을 실행하라.

```
% ------------------------------------------------------------------
%  Filename: GaussHessenberg101DY.m
%  Boundary Value Problem for Bratu's equation
%  Using Gauss Elimination w/ Hessenberg Matrix
% (cf)  Charles Van Loan, 2014 @Cornell
% Numerical Analysis and Differential Equations
% http://www.cs.cornell.edu/courses/cs4210/2014fa/CVLBook/chap6.pdf
%-------------------------------------------------------------------
clear all, close all
% Making the Matrix-form of the equation
    lambda = 1;
    n = 10^3
    h = 1/(n+1)
    xx = (1:n)'*h;
    eVec = ones(n,1);
    A = spdiags([ -eVec 2*eVec -eVec ],[-1 0 1],n,n);
    b =  h^2*exp(xx);
% Convert  Hessenberg to Upper Triangular
Ab = [ A b ];
[n,n] = size(A);
v = zeros(n,1);
for kk=1:n-1
    v(kk+1) = Ab(kk+1,kk)/Ab(kk,kk);
    Ab(kk+1,kk:n+1) = Ab(kk+1,kk:n+1) - v(kk+1)*Ab(kk,kk:n+1);
end % end-for-kk
Autri = Ab(:,1:n);
butri = Ab(:,n+1);
% Double Check
xold = A\b;  dumold = norm(A*xold -b)
xutri = Autri\butri; dumutri = norm(Autri*xutri-butri)
xdif1 = max(abs(xold - xutri))
% Solve an Upper triangular by Backward substitution
for jj = n:-1:2
    xbs(jj,1) = butri(jj)/Autri(jj,jj);
    butri(1:jj-1) = butri(1:jj-1) - xbs(jj,1)*Autri(1:jj-1,jj);
end % end-for-jj
xbs(1,1) = butri(1)/Autri(1,1);
dumbs = norm(A*xbs - b)
xdif2 = max(abs(xold - xbs))
utrue = (1-exp(xx))+xx*(exp(1)-1);
% Plotting
subplot(2,1,1)
plot([0 ; xx; 1], [0; utrue; 0 ],'g-',[0 ; xx; 1], [0; xbs; 0 ],'k:','
    LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
% title('\bf Gauss Elimination w/ Hessenberg')
legend('\bf Exact Solution','\bf Hessenberg','location','NW')
subplot(2,1,2)
error = utrue -xbs;
plot([0 ; xx; 1], [0; error; 0 ],'r-','LineWidth',2);
set(gca,'fontsize',11,'fontweigh','bold')
legend('\bf Error','location','NW')
saveas(gcf,'GaussHessenberg101DY.jpg')
% End of Program
```

이 MATLAB프로그램을 실행하면, 그림 7.9.3가 그려진다. 그림 7.9.3의 상단 그래프는 진짜해와 Hessenberg행렬을 이용한 Gauss소거법으로 구한 해를 그린 것이다. 이 그래프에서

알 수 있듯이, 이 수치해는 진짜해와 아주 비슷하다. 이들의 차이를 그린 것이 그림 7.9.3의 하단 그래프이다. 이 그래프에서 알 수 있듯이, 각 점에서 오차는 1.8×10^{-8} 이하이다. ■

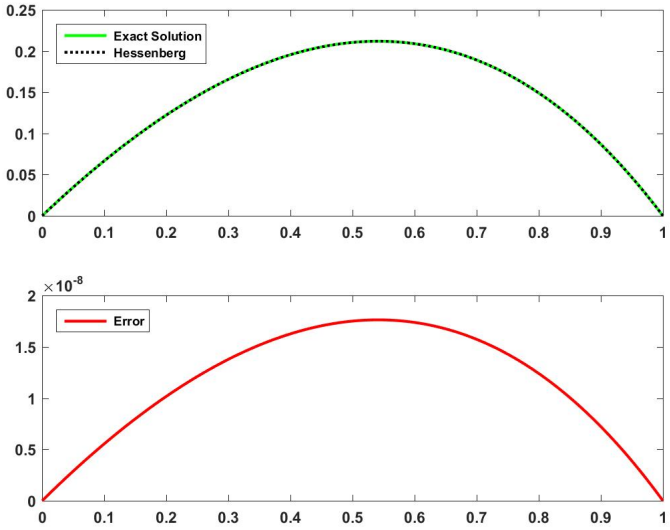


그림 7.9.3. Hessenberg행렬과 Gauss소거법

제 8 장

적분과 미분방정식

제8.1절 구적법

연속형 확률변수 x의 확률밀도함수를 $f(x)$라고 하면, 확률변수 x가 구간 $[a,b]$에 들어갈 확률과 함수 $g(x)$의 기대값은 각각 다음과 같다.

$$\Pr(a \leq x \leq b) = \int_a^b f(x)dx \tag{8.1.1}$$

$$E(g(x)) = \int_{-\infty}^{\infty} g(x)f(x)dx \tag{8.1.2}$$

즉, 확률이나 기대값을 계산하는 것은 정적분을 구하는 문제로 귀착된다. 물론, 몬테카를로법을 사용해서 기대값을 구할 수도 있으나, 기대값을 계산하는 기본적인 방법은 정적분을 구하는 것이다. 피적분함수의 부정적분이 초등함수로서 구해지는 경우가 아니라면, 수치적으로 이 정적분을 계산하지 않을 수 없다. 수치적분을 구적법(求積法, quadrature) 또는 구분구적법(區分求積法, quadrature by parts)이라 부른다. 이 장에서는 주로 독립변수가 1개인 함수의 적분을 다루기로 하자. 물론, 일부 수치적분법들은 독립변수들이 2개 이상인 함수에도 적용할 수 있다.

제8.2절 Newton-Cotes형 수치적분

다음 정적분을 수치적으로 계산하기로 하자.

$$I = \int_a^b f(x)dx \tag{8.2.1}$$

적분구간 $[a,b]$의 분할 $\Pi = \{a = x_0 < x_1 < \cdots < x_n = b\}$를 생각해보자. 우선, 등간격인 분할을 생각해보자. 즉, 다음 식이 성립한다고 가정하자.

$$h \doteq \frac{b-a}{n}, \quad x_k \doteq a + kh, \quad (k = 0, 1, \cdots, n) \tag{8.2.2}$$

또한, 다음 값들을 정의하자.

$$f_k \doteq f(x_k) = f(a + kh), \quad (k = 0, 1, \cdots, n) \tag{8.2.3}$$

정적분 I를 수치적으로 계산하기 위해서는, 먼저 각 소구간 $[x_k, x_{k+1}]$에서 정적분의 근사값을 구한 다음, 이들을 더한다. 또한, 함수 $f(x)$의 적분구간에서 몇 점들을 선택한 다음, 그 점들을 지나는 멱함수를 구하고, 그 멱함수를 함수 $f(x)$의 근사식으로 간주해서 수치적분을 하는 방법이 많이 사용된다. 그러나, 실제 이러한 방법을 적용할 때는 멱함수로 근사시키는 과정을 거치지 않고, 그러한 원리에 입각해서 유도된 공식을 이용한다. 이러한 공식들을 총칭해서 Newton-Cotes식이라 부른다. Newton-Cotes식에 대한 자세한 내용은 Stoer & Bulirsch [56, 제3장]을 참조하라.

소구간 $[x_{k-1}, x_{k+1}]$에서 함수 $f(x)$를 0차 멱함수 또는 1차 멱함수로 근사시키는 Newton-Cotes식들은 다음과 같다. 첫째, 소구간 $[x_k, x_{k+1}]$에서 함수 $f(x)$를 상수함수, 즉 0차 멱함수로 근사시키는 적분공식들은 다음과 같다.

$$\text{하한공식 (lower-point rule):} \quad L_n \doteq h \sum_{k=1}^{n} f(x_{k-1}) \tag{8.2.4}$$

$$\text{상한공식 (upper-point rule):} \quad U_n \doteq h \sum_{k=1}^{n} f(x_k) \tag{8.2.5}$$

$$\text{중점공식 (midpoint rule):} \quad M_n \doteq h \sum_{k=1}^{n} f\left(\frac{x_{k-1} + x_k}{2}\right) \tag{8.2.6}$$

식 (8.2.4)의 L_n을 좌Riemann합, 그리고 식 (8.2.5)의 U_n을 우Riemann합이라 부른다. 둘째, 소구간 $[x_k, x_{k+1}]$에서 함수 $f(x)$를 1차 멱함수로 근사시키는 사다리꼴공식 (traperzoidal rule)은 다음과 같다.

$$T_n \doteq h \sum_{k=1}^{n} \frac{f(x_{k-1}) + f(x_k)}{2} \tag{8.2.7}$$

다음 식이 성립함은 자명하다.

$$T_n = \frac{L_n + U_n}{2} \tag{8.2.8}$$

예제 8.2.1 제0차와 제1차 Newton-Cotes식들을 사용해서 함수 $f(x) = 3x^2$ 의 정적분을 계산하기 위해서, 다음 MATLAB프로그램 NewtonCotes101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: NewtonCotes101.m
%  Lower-, Upper- and Mid-point Rules
%  Traperzoidal Rule
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, close all
sqr = @(x) 3*x.^2
a = 0, b = 1, n = 5
h = (b-a)/n;
xk = linspace(a,b,n+1)
fk = sqr(xk)
% Newton-Cotes formulae
Ln = h*sum(fk(1:n))                       % Lower-point rule
Un = h*sum(fk(2:n+1))                     % Upper-point rule
Mn = h*sum(sqr((xk(1:n)+xk(2:n+1))/2))    % Mid-point rule
Tn = (Un + Ln)/2                          % Traperzoidal rule
Tn2 = trapz(xk,fk)                        % MATLAB function
save('NewtonCotes101.txt','Ln','Un','Mn','Tn','Tn2','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 NewtonCotes101.m을 실행하면, 구간 $[0,1]$의 분할 $\{0 < 0.2 < 0.4 < 0.6 < 0.8 < 1\}$에서 다음 정적분을 수치적으로 계산한다.

$$I = \int_0^1 3x^2 dx \tag{1}$$

하한공식, 상한공식 그리고 중점공식에 의한 수치적분값들은 각각 다음과 같다.

$$L_5 = 0.72, \quad U_5 = 1.32, \quad M_5 = 0.99 \tag{2}$$

사다리꼴공식에 의한 수치적분값은 다음과 같다.

$$T_5 = 1.02 \tag{3}$$

MATLAB의 내장함수 trapz.m을 사용해도 사다리꼴공식에 의한 수치적분을 할 수 있다. 이 내장함수를 사용해도 같은 값이 얻어짐을 확인할 수 있다. ■

예제 8.2.2 Python을 사용해서 예제 8.2.1를 다시 다루기 위해서, 다음 Python프로그램 NewtonCotes101.Py를 실행해 보자.

```
"""
Filename: NewtonCotes101.Py
Lower-, Upper-, Mid-point Rules, Traperzoidal Rule
Programmed by CBS
"""

import numpy as np

def sqr(x):
    return 3*x**2

a = 0; b = 1; n = 5
h = (b-a)/n
xk = np.linspace(a,b,n+1)
fk = sqr(xk)

# Newton-Cotes formulae
Ln = h*np.sum(fk[0:n]); Ln                          # Lower-point rule
Un = h*np.sum(fk[1:(n+1)]); Un                      # Upper-point rule
Mn = h*np.sum(sqr((xk[0:n]+xk[1:n+1])/2)); Mn       # Mid-point rule
Tn = (Un + Ln)/2; Tn                                # Traperzoidal rule
Tn2 = np.trapz(fk,xk); Tn2                          # Python function

# Save Dataset
OutX = [ Ln, Un, Mn, Tn, Tn2 ]; OutX
np.savetxt(r'NewtonCotes101Py.txt', OutX, fmt='%d')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 8.2.1의 결과와 같다. ■

소구간 $[x_{k-1}, x_{k+1}]$에서 함수 $f(x)$를 2차 멱함수로 근사시키는 적분공식을 유도하기로 하자. 세 점들 x_{k-1}, x_k, x_{k+1}을 지나는 Lagrange보간식은 다음과 같다,

$$\begin{aligned} p_{k+1}(x) =& f(x_{k-1})\frac{[x-x_k][x-x_{k+1}]}{[x_{k-1}-x_k][x_{k-1}-x_{k+1}]} + f(x_k)\frac{[x-x_{k-1}][x-x_{k+1}]}{[x_k-x_{k-1}][x_k-x_{k+1}]} \\ &+ f(x_{k+1})\frac{[x-x_{k-1}][x-x_k]}{[x_{k+1}-x_{k-1}][x_{k+1}-x_k]} \end{aligned} \tag{8.2.9}$$

식 (8.2.9)를 바탕으로 다음 식들을 유도할 수 있다.

$$\int_{x_{k-1}}^{x_{k+1}} f(x)dx \approx \int_{x_{k-1}}^{x_{k+1}} p_2(x)dx = \frac{h}{3}[f_{k-1} + 4f_k + f_{k+1}] \tag{8.2.10}$$

식 (8.2.10)을 바탕으로 정적분의 근사값을 구하는 식을 Simpson공식이라 부른다. 정적분 I를 구하는 Simpson공식은 다음과 같다.

$$S_n = \frac{h}{6}\sum_{k=1}^{n}\Big[f(a+[k-1]h) + 4f(a+[k-0.5]h) + f(a+kh)\Big] \tag{8.2.11}$$

Simpson공식은 다음 식들을 만족한다.

$$S_n = \frac{L_n + 4M_n + U_n}{6} = \frac{T_n + 2M_n}{3} \tag{8.2.12}$$

적분구간 $[a,b]$를 길이가 같은 $2n$개 소구간들로 나누고 각 소구간의 길이를 $\eta = [b-a]/[2n]$라고 하면, Simpson공식은 다음과 같다.

$$S_n = \frac{\eta}{3}\left[f(a) + 4\sum_{k=1}^{n} f(a+[2k-1]\eta) + 2\sum_{k=1}^{n-1} f(a+2k\eta) + f(b)\right] \tag{8.2.13}$$

예제 8.2.3 함수 $f(x) = 3x^2$의 정적분을 0차와 1차 Newton-Cotes식들을 사용해서 계산하기 위해서, 다음 MATLAB프로그램 SimpsonRule101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: SimpsonRule101.m
%  Simpson Rule = Newton-Cotes w/ order 1
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, clear all, format rat
sqr = @(x) 3*x.^2
a = 0, b = 1, n = 5
n2 = 2*n;
h = (b-a)/n2
xk = linspace(a,b,n2+1)
fk = sqr(xk)
% Simpson Rule
LL = sum(fk(1:2:n2-1))
RR = sum(fk(3:2:n2+1))
CC = sum(fk(2:2:n2))
Sim = h/3*(LL + 4*CC + RR)
Tn2 = quad(sqr,0,1)                    % MATLAB function
save('SimpsonRule101.txt','Sim','Tn2','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 SimpsonRule101.m을 실행하면, Simpson공식을 사용해서 다음 정적분을 수치적으로 계산한다.

$$I = \int_0^1 3x^2 dx \tag{1}$$

이 MATLAB프로그램 Simpson101.m을 실행한 결과는 다음과 같다.

$$I_5 = \frac{0.1}{3}\left[\frac{18}{5} + 4\cdot\frac{99}{20} + \frac{33}{5}\right] = 1 \tag{2}$$

MATLAB의 내장함수 quad.m을 사용해도 Simpson공식에 의한 수치적분을 할 수 있다. 내장함수 quad.m을 사용해도 같은 값이 얻어짐을 확인할 수 있다. ■

예제 8.2.4 Python을 사용해서 예제 8.2.3을 다시 다루기 위해서, 다음 Python프로그램 SimpsonRule101.Py를 실행해 보자.

```
"""
% Filename: SimpsonRule101.Py
% Simpson Rule = Newton-Cotes w/ order 1
% Programmed by CBS
"""

import numpy as np
from scipy.integrate import simps

def sqr(x):
    return 3*x**2

a = 0; b = 1; n = 5
n2 = 2*n
h = (b-a)/n2
xk = np.linspace(a,b,n2+1)
fk = sqr(xk)

# Simpson Rule
LL = sum(fk[0:n2:2])
RR = sum(fk[2:n2+2:2])
CC = sum(fk[1:n2+1:2])
Sim = h/3*(LL + 4*CC + RR)
Sim2 = simps(fk,xk)

# Save Dataset
OutX = [ LL, RR, CC, Sim, Sim2 ]; OutX
np.savetxt(r'SimpsonRule101Py.txt', OutX, fmt='%d')

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 8.2.3의 결과와 같다. ■

소구간 $[x_k, x_{k+1}]$에서 함수 $f(x)$를 m차 멱함수 $p_m^{(k)}(x)$로 근사시키는 적분공식을 살펴보자. 독자들에게 먼저 말해두고 싶은 것은 보편적인 경우에는 사다리꼴공식이나 Simpson 공식으로 충분하다는 점이다. 그러나, 특성함수를 계산하는 등 특수한 문제에 부딪히면, 좀 더 높은 차수의 Newton-Cotes식이 필요할 수도 있다. 다음 상수들을 정의하자.

$$H \doteq \frac{h}{m} = \frac{x_{k+1} - x_k}{m} \tag{8.2.14}$$

$$f_j^{(k)} \doteq f(x_k + jH) = f\left(x_k + j\frac{h}{m}\right) = f\left(a + \left[k + \frac{j}{m}\right]h\right), \quad (j = 0, 1, \cdots, m) \tag{8.2.15}$$

소구간 $[x_k, x_{k+1}]$에서 함수 $f(x)$의 정적분의 근사값은 다음과 같다.

$$\int_{x_k}^{x_{k+1}} f(x)dx \approx \int_{x_k}^{x_{k+1}} p_m^{(k)}(x)dx = \frac{h}{m} K_m \sum_{j=0}^{m} B_{m,j} f_j^{(k)} \tag{8.2.16}$$

식 (8.2.16)의 K_m과 계수 $B_{m,j}$가 표 8.2.1에 수록되어 있다. 식 (8.2.16)에서 알 수 있듯이, 함수 $f(x)$를 m차 멱함수로 근사시키는 Newton-Cotes식은 다음과 같다.

$$\int_a^b f(x)dx \approx \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} p_m^{(k)}(x)dx = \frac{h}{m} K_m \sum_{k=0}^{n-1} \sum_{j=0}^{m} B_{m,j} f_j^{(k)} \tag{8.2.17}$$

m	이름	K_m	$B_{m,0}$	$B_{m,1}$	$B_{m,2}$	$B_{m,3}$	$B_{m,4}$	$B_{m,5}$	$B_{m,6}$
1	사다리꼴	1/2	1	1					
2	Simpson	1/3	1	4	1				
3	Simpson-$\frac{3}{8}$	3/8	1	3	3	1			
4	Miline/Boole	2/45	7	32	12	32	7		
5		5/288	19	75	50	50	75	19	
6	Weddle	1/140	41	216	272	27	27	216	41

표 8.2.1. Newton-Cotes계수

표 8.2.1에 기술한 Simpson의 3/8공식이란 소구간 $[x_k, x_{k+1}]$에서 함수 $f(x)$를 3차 Lagrange보간함수 $p_3^{(k)}(x)$에 적합시켜 수치적분을 하는 것이다. 소구간 $[x_k, x_{k+1}]$에서 Simpson의 3/8공식은 다음과 같다.

$$\begin{aligned}&\int_{x_k}^{x_{k+1}} f(x)dx \approx \int_{x_k}^{x_{k+1}} p_3^{(k)}(x)dx \\ &= \frac{h}{3} \cdot \frac{3}{8} \left[f(a+kh) + 3f\left((a+kh+\frac{h}{3}\right) + 3f\left(a+kh+\frac{2h}{3}\right) + f(a+[k+1]h) \right]\end{aligned} \tag{8.2.18}$$

제8.3절 Newton-Cotes식의 정밀도

함수 $f(x)$가 적분구간 $[a, b]$ 내에서 적당한 횟수만큼 미분가능하다고 가정하고, 소구간 $[x_k, x_{k+1}]$에서 오차를 평가하자. 함수 $f(x)$를 점 $x = x_k$ 주변에서 Taylor전개하면, 다음과 같다.

$$f(x) = f(x_k) + f'(x_k)[x-x_k] + \frac{1}{2!} f''(x_k)[x-x_k]^2 + \frac{1}{3!} f^{(3)}(x_k)[x-x_k]^3 + \cdots \tag{8.3.1}$$

따라서, 다음 식들이 성립한다.

$$\begin{aligned} I_k &= \int_{x_k}^{x_{k+1}} f(x)dx \\ &= f(x_k)h + \frac{1}{2!}f'(x_k)h^2 + \frac{1}{3!}f''(x_k)h^3 + \frac{1}{4!}f^{(3)}(x_k)h^4 + \frac{1}{5!}f^{(4)}(x_k)h^5 + \cdots \end{aligned} \tag{8.3.2}$$

각 k에 대해서 식 $h = x_{k+1} - x_k$가 성립함을 상기하라.

첫째, 소구간 $[x_k, x_{k+1}]$에서 함수 $f(x)$를 상수함수, 즉 0차 멱함수로 근사시키는 경우를 살펴보자. 하한공식에 의한 적분값은 $L_k \doteq f(x_k)h$이므로, 식 (8.3.2)에서 알 수 있듯이 오차는 다음과 같다.

$$e_k^L \doteq L_k - I_k = -\frac{1}{2}f'(\xi_L)h^2 + O(h^3) \tag{8.3.3}$$

여기서 ξ_L은 소구간 (x_k, x_{k+1})에 속하는 상수이다. 상한공식에 의한 적분값은 $U_k \doteq f(x_{k+1})h$이므로, 다음 식들이 성립한다.

$$U_k = f(x_k + h)h = h\left[f(x_k) + f'(x_k)h + \frac{1}{2!}f''(x_k)h^2 + \cdots\right] \tag{8.3.4}$$

식 (8.3.2)와 식 (8.3.4)에서 알 수 있듯이, 오차는 다음과 같다.

$$e_k^U \doteq U_k - I_k = \frac{1}{2}f'(\xi_U)h^2 + O(h^3) \tag{8.3.5}$$

여기서 ξ_U는 소구간 (x_k, x_{k+1})에 속하는 상수이다. 따라서, 하한공식과 상한공식이 같은 정도의 정밀도를 갖는다는 것을 알 수 있다. 중점공식에 의한 적분값은 $M_k \doteq f([x_{k+1} + x_k]/2)h$이므로, 다음 식들이 성립한다.

$$M_k = f\left(x_k + \frac{h}{2}\right)h = h\left\{f(x_k) + f'(x_k)\frac{h}{2} + \frac{1}{2!}f''(x_k)\left[\frac{h}{2}\right]^2 + \cdots\right\} \tag{8.3.6}$$

식 (8.3.2)와 식 (8.3.6)에서 알 수 있듯이, 오차는 다음과 같다.

$$e_k^M \doteq M_k - I_k = -\frac{1}{24}f''(\xi_M)h^3 + O(h^4) \tag{8.3.7}$$

여기서 ξ_M은 소구간 (x_k, x_{k+1})에 속하는 상수이다. 즉, 중심공식은 하한공식과 상한공식보다 높은 정밀도를 갖는다. 식 (8.3.7)에서 알 수 있듯이, 중점공식은 함수 $f(x)$를 상수함수로 근사시킴에도 불구하고, $f(x)$가 1차함수인 경우에도 오차는 없다.

둘째, 소구간 $(x_k, x_{k=1})$에서 함수 $f(x)$를 2차 멱함수로 근사시키는 사다리꼴공식의 오차를 살펴보자. 식 (8.3.1)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{1}{2}[f(x_k)+f(x_{k+1})]h = f(x_k)h + \frac{1}{2}f'(x_k)h^2 + \frac{1}{4}f''(x_k)h^2 + \cdots \tag{8.3.8}$$

따라서, 사다리꼴공식 $T_k \doteq h[f(x_k)+f(x_{k+1})]/2$의 오차는 다음과 같다.

$$e_k^T \doteq T_k - I_k = \frac{1}{12}f''(\xi_T)h^3 + O(h^4) \tag{8.3.9}$$

여기서 ξ_T는 소구간 (x_k, x_{k+1})에 속하는 상수이다. 식 (8.3.7)과 식 (8.3.9)에서 알 수 있듯이, 사다리꼴공식의 오차는 중점공식 오차의 약 2배이다.

셋째, 소구간 $[x_k, x_{k+1}]$에서 함수 $f(x)$를 2차 멱함수로 근사시키는 Simpson공식의 오차를 살펴보자. 식 (8.3.1)과 식 (8.3.6)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\begin{aligned} &f(x_{k+1})h = f(x_k+h)h \\ &= h\left[f(x_k) + f'(x_k)h + \frac{1}{2!}f''(x_k)h^2 + \frac{1}{3!}f^{(3)}(x_k)h^3 + \frac{1}{4!}f^{(4)}h^4 + \cdots\right] \end{aligned} \tag{8.3.10}$$

$$\begin{aligned} &f\left(\frac{x_k+x_{k+1}}{2}\right)h = f\left(x_k+\frac{h}{2}\right)h \\ &= h\left[f(x_k) + f'(x_k)\frac{h}{2} + \frac{1}{2!}f''(x_k)\frac{h^2}{4} + \frac{1}{3!}f^{(3)}(x_k)\frac{h^3}{8} + \frac{1}{4!}f^{(4)}(x_k)\frac{h^4}{16} + \cdots\right] \end{aligned} \tag{8.3.11}$$

식 (8.3.1), 식 (8.3.10) 그리고 식 (8.3.11)에서 알 수 있듯이, 소구간 $[x_k, x_{k+1}]$에서 Simpson 공식 S_k는 다음 식을 만족한다.

$$\begin{aligned} S_k &\doteq \frac{1}{6}\left[f(x_k) + 4f\left(\frac{x_k+x_{k+1}}{2}\right) + f(x_{k+1})\right]h \\ &= f(x_k)h + \frac{1}{2}f'(x_k)h^2 + \frac{1}{6}f''(x_k)h^3 + \frac{1}{24}f^{(3)}(x_k)h^4 + \frac{5}{576}f^{(4)}(x_k)h^5 + \cdots \end{aligned} \tag{8.3.12}$$

식 (8.3.2)와 식 (8.3.12)에서 알 수 있듯이, Simpson공식 S_k의 오차는 다음과 같다.

$$e_k^S \doteq S_k - I_k = \frac{1}{2880}f^{(4)}(\xi_S)h^5 + O(h^6) \tag{8.3.13}$$

여기서 ξ_S는 소구간 (x_k, x_{k+1})에 속하는 상수이다. Simpson공식에서는 함수값을 평가하는 간격이 h가 아닌 $H \doteq h/2$이므로, 식 (8.3.13)을 다음과 같이 쓸 수 있다.

$$e_k^S \doteq S_k - I_k = \frac{1}{90}f^{(4)}(\xi_S)H^5 + O(H^6) \tag{8.3.14}$$

Simpson공식에서는 함수 $f(x)$의 근사에 2차 멱함수를 이용하고 있으므로, 함수 $f(x)$가 2차 멱함수인 경우에 오차가 없는 것은 당연하지만, 오차 e_k^S가 h의 5제곱이라는 것은 $f(x)$가 3차 멱함수인 경우에도 오차가 없다는 것을 의미한다. 실제, 이 성질을 증명하는 것은 어렵지 않다. 따라서, Simpson공식은 실제로 아주 유용하다고 할 수 있다. 일반적으로, m이 짝수인 Newton-Cotes식에서는 $m+1$차 멱함수에 대해서 오차를 발생시키지 않는다는 것을 증명할 수 있다. 따라서, m이 홀수인 경우에는 $[m-1]$에 대한 Newton-Cotes식을 사용하는 것이 좋다.

지금까지 내용을 바탕으로, 함수 $f(x)$가 적분구간 $[a,b]$에서 Newton-Cotes식의 오차를 살펴보자. 앞에서와 마찬가지로, 적분구간 $[a,b]$를 n등분한 소구간 길이를 $h=[b-a]/n$이라고 하면, 적분구간 $[a,b]$에서 각 Newton-Cotes식의 오차는 다음과 같다.

$$e_L \doteq |L_n - I| = ne_k^L \le n\left|\frac{1}{2}f'(\xi)h^2\right| = \frac{[b-a]^2}{2n}|f'(\xi)| \tag{8.3.15}$$

$$e_U \doteq |U_n - I| = ne_k^U \le n\left|\frac{1}{2}f'(\xi)h^2\right| = \frac{[b-a]^2}{2n}|f'(\xi)| \tag{8.3.16}$$

$$e_M \doteq |M_n - I| = ne_k^M \le n\left|\frac{1}{24}f''(\xi)h^3\right| = \frac{[b-a]^3}{24n^2}|f''(\xi)| \tag{8.3.17}$$

$$e_T \doteq |T_n - I| = ne_k^T \le n\left|\frac{1}{12}f''(\xi)h^3\right| = \frac{[b-a]^3}{12n^2}|f''(\xi)| \tag{8.3.18}$$

$$e_S \doteq |S_n - I| = ne_k^S \le n\left|\frac{1}{2880}f^{(4)}(\xi)h^5\right| = \frac{[b-a]^5}{2880n^4}|f^{(4)}(\xi)| \tag{8.3.19}$$

그러나, Simpson공식의 경우에는 n개로 분할한 소구간을 둘로 나누므로, 총분할 수는 $N=2n$이다. 따라서, 식 (8.3.19)를 다음과 같이 쓸 수 있다.

$$e_S \doteq |S_n - I| = \frac{N}{2}e_k^S \le \frac{N}{2}\left|\frac{1}{2880}f^{(4)}(\xi)h^5\right| = \frac{[b-a]^5}{180N^4}|f^{(4)}(\xi)| \tag{8.3.20}$$

식 (8.3.15)$\sim$식 (8.3.20)에서 알 수 있듯이, n을 2배해서 소구간 길이 h를 절반 $h/2$으로 만들면, 상한공식과 하한공식에서는 오차가 1/2로, 중점공식과 사다리꼴공식에서는 오차가 1/4로, 그리고 Simpson공식에서는 오차가 1/16이 된다.

제8.4절 Romberg적분법

적분구간 $[a,b]$를 n개 소구간들로 나누어, 정적분 I의 근사값 $I(a,b,h)$를 구했다고 하자. 오차를 감소시키기 위해서는 소구간 길이 $h=[b-a]/n$을 반으로 줄여 좀 더 정밀도가 높은

근사값 $I(a,b,h/2)$를 구하고자 할 때, 처음부터 다시 계산하지 않고 이미 구한 근사값 $I(a,b,h)$에 추가된 격자점(grid point)들에 대한 값들만을 추가해서, 근사값 $I(a,b,h/2)$를 재귀적으로(recursively) 구하는 방법을 살펴보자.

이 재귀적 방법을 설명하기 위해서는 Bernoulli멱함수와 Bernoulli수를 필요로 한다. 다음 식을 살펴보자.

$$\frac{te^{xt}}{e^t-1}=\sum_{n=0}^{\infty}B_n(x)\frac{t^n}{n!},\quad (|t|<2\pi) \tag{8.4.1}$$

식 (8.4.1)의 $B_n(x)$를 n차 Bernoulli멱함수라 부르고, $B_n \doteq B_n(0)$를 Bernoulli수라 부른다. 첫 Bernouille수들은 다음과 같다.

$$B_0=1,\ B_1=-\frac{1}{2},\ B_2=\frac{1}{6},\ B_4=-\frac{1}{30},\ B_6=\frac{1}{42},\ B_8=-\frac{1}{30} \tag{8.4.2}$$

다음 명제는 Abramowitz & Stegun [7, p. 886]에서 인용한 것이다.

명제 8.4.1: Euler-Maclaurin식

구간 $[a,b]$에서 함수 $f(x)$가 C^{2m}급이라고 하자. 구간 $[a,b]$를 등간격 $h=[b-a]/n$으로 나눈 분할 $\Pi=\{a=x_0<x_1<\cdots<x_n=b\}$에 대해서 다음 식이 성립한다.

$$\begin{aligned}\int_a^b f(x)dx =&\frac{h}{2}\left[f(a)+2\sum_{k=1}^{n-1}f(x_k)+f(b)\right]\\ &-\sum_{j=1}^{n}\frac{1}{(2j)!}B_{2j}h^{2j}[f^{(2j-1)}(b)-f^{(2j-1)}(a)]+R_{2m}\end{aligned}$$

여기서 R_{2m}은 잔차항이다.

식 (8.3.9)에서 알 수 있듯이, 사다리꼴공식의 오차는 다음과 같다.

$$\frac{h^3}{12}\sum_{j=1}^{n}f''(\xi_j)\approx\frac{h^2}{12}\int_a^b f''(\xi)d\xi=\frac{h^2}{12}[f'(b)-f'(a)] \tag{8.4.3}$$

식 (8.4.3)에서 알 수 있듯이, 사다리꼴공식에서 소구간들의 개수 n을 2배하여 소구간 길이 h를 절반으로 하면 오차는 1/4이 된다. 또한, 식 $f'(a)=f'(b)$이 성립할 때 오차가 작아진다. 명제 8.4.1에서 알 수 있듯이, Euler-Maclaurin식의 우변의 첫 번째 항이 사다리꼴공식에 의한

적분근사값 T_n 이다. 이 적분근사값을 소구간 길이 h 의 함수 $T(h)$ 로 표기하면, 다음 식이 성립한다.

$$T(h) = \int_a^b f(x)dx + c_1 h^2 + c_2 h^4 + c_3 h^6 + \cdots \tag{8.4.4}$$

여기서 $c_1, c_2, c_3, \cdots$ 는 $f(x)$, a, 그리고 b에 의존하지만 h에는 의존하지 않는 상수들이다. 또한, Euler-Maclaurin식에서 알 수 있듯이, 다음 식이 성립한다.

$$c_1 = \frac{f'(b) - f'(a)}{12} \tag{8.4.5}$$

식 (8.4.4)에서 알 수 있듯이, 사다리꼴공식에서 소구간 길이를 $h/2$로 하는 적분근사값 $T(h/2)$는 다음 식을 만족한다.

$$T\left(\frac{h}{2}\right) = \int_a^b f(x)dx + c_1 \frac{h^2}{4} + c_2 \frac{h^4}{16} + c_3 \frac{h^6}{64} + \cdots \tag{8.4.6}$$

식 (8.4.4)와 식 (8.4.6)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{1}{3}\left[4T\left(\frac{h}{2}\right) - T(h)\right] = \int_a^b f(x)dx + d_2 h^4 + d_3 h^6 + \cdots \tag{8.4.7}$$

식 (8.4.4)와는 달리 식 (8.4.7)에서는 h^2 항이 소거되었다. 따라서, 식 (8.4.4)보다 식 (8.4.7)이 정밀도가 높은 근사식이다. 식 (8.4.7)의 좌변이 Simpson공식이라는 것을 쉽게 증명할 수 있다. 같은 방법을 반복적으로 적용해서, 소구간 길이를 절반으로 함으로써 정밀도가 높은 수치적분값을 구할 수 있다. 이러한 수치적분법을 Romberg적분법이라 한다.

알고리즘 8.4.1: Romberg알고리즘

함수 $f(x)$를 구간 $[a, b]$에서 C^{2m} 급이라 하고, 사다리꼴공식의 소구간 길이를 $h_k = [b-a]/2^k$ 라고 하자. 사다리꼴공식에 의한 초기 적분근사값을 $T_0(h_0)$로 하면, 다음 점화식을 적용해서, 좀 더 정밀한 적분근사값들 $\{T_l(h_k)\}$을 구할 수 있다.

$$T_l(h_k) = \frac{1}{4^l - 1}[4^l T_{l-1}(h_{k+1}) - T_{l-1}(h_k)], \quad (l = 1, 2, \cdots, \min\{k, m-1\})$$

Romberg알고리즘에서 $k = l = 1$인 경우가 식 (8.4.7)이다. Romberg알고리즘이 Richardson알고리즘의 특수한 경우임은 명백하다. 따라서, 이 알고리즘을 Romberg-Richardson알고

리즘이라 부르기도 한다. 다음 식들이 성립한다.

$$I = T_n(h) + O(h^{2n}), \quad (n = 1, 2, \cdots) \tag{8.4.8}$$

정적분과 피적분함수의 미분값을 알지 못해도 정적분과 수치적분값의 차이를 짐작할 수가 있다. 소구간 $[x_{k-1}, x_{k+1}]$에서 Simpson공식에 의한 수치적분을 다음과 같이 표기하자.

$$S(x_{k-1}, x_{k+1}, h) \doteq \frac{h}{3}[f(x_{k-1}) + 4f(x_k) + f(x_{k+1})] \tag{8.4.9}$$

식 (8.3.14)에서 알 수 있듯이, 정적분 I_k에서 Simpson공식에 의한 수치적분 $S(x_{k-1}, x_{k+1}, h)$를 뺀 오차는 다음과 같다.

$$e_k^S(h) \doteq \int_{x_{k-1}}^{x_{k+1}} f(x)dx - S(x_{k-1}, x_{k+1}, h) \approx -\frac{1}{90}h^5 f(\xi) \tag{8.4.10}$$

여기서 ξ는 적분구간 내의 점이다. 소구간의 폭을 반으로 줄이면, 격자점들이 3개에서 5개로 증가한다. 이 경우에 오차는 다음과 같다.

$$\begin{aligned} e_k^S\left(\frac{h}{2}\right) &\doteq \int_{x_{k-1}}^{x_{k+1}} f(x)dx - S\left(x_{k-1}, x_{k+1}, \frac{h}{2}\right) \\ &= \int_{x_{k-1}}^{x_k} f(x)dx - S\left(x_{k-1}, x_k, \frac{h}{2}\right) + \int_{x_k}^{x_{k+1}} f(x)dx - S\left(x_k, x_{k+1}, \frac{h}{2}\right) \\ &\approx -2\frac{1}{90}\left[\frac{h}{2}\right]^5 f(\xi) = \frac{1}{16}e_k^S(h) \end{aligned} \tag{8.4.11}$$

여기서 세 번째 등식은 식 (8.4.10)에 의해서 성립한다. 식 (8.4.11)에서 알 수 있듯이, 다음 식들이 성립한다.

$$e_k^S(h) - e_k^S\left(\frac{h}{2}\right) = S(x_{k-1}, x_{k+1}, h) - S\left(x_{k-1}, x_{k+1}, \frac{h}{2}\right) \approx \frac{15}{16}e_k^S(h) \approx 15e_k^S\left(\frac{h}{2}\right) \tag{8.4.12}$$

식 (8.4.12)에서 정적분과 Simpson공식에 의한 수치적분값의 차이가 다음 근사식을 만족함을 짐작할 수 있다.

$$e_k^S\left(\frac{h}{2}\right) \approx \frac{1}{4^2 - 1}\left[S(x_{k-1}, x_{k+1}, h) - S\left(x_{k-1}, x_{k+1}, \frac{h}{2}\right)\right] \tag{8.4.13}$$

같은 방법을 적용해서, 소구간 $[x_{k-1}, x_{k+1}]$에서 정적분과 사다리꼴공식에 의한 수치적분값 $T(x_{k-1}, x_{k+1}, h)$의 차이가 다음 근사식을 만족함을 짐작할 수 있다.

$$e_k^S\left(\frac{h}{2}\right) \approx \frac{1}{2^2-1}\left[T(x_{k-1}, x_{k+1}, h) - T\left(x_{k-1}, x_{k+1}, \frac{h}{2}\right)\right] \tag{8.4.14}$$

예제 8.4.1 함수 $f(x) = x[1-x]e^{-x}$의 정적분을 Romberg알고리즘을 사용해서 계산하기 위해서, 다음 MATLAB프로그램 RombergTable101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: RombergTable101.m
%  Romberg-Richardson Table to calculate ff over [a,b]
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, format short
ff = @(x) x.*(1-x).*exp(-x);
a = 0, b = 4, nn = 1
h = (b-a)/nn;
tol = 1.0e-08;
xk = linspace(a,b,nn+1);
fk = ff(xk);
IntTrap = h*sum((fk(1:nn) + fk(2:nn+1))/2)     % Traperzoidal rule
RR(1,1) = IntTrap;
for kk = 2:1:10
    h = h/2; nn = nn*2;
    sumodd = 0;
    for jj = 1:2:nn-1
        sumodd = sumodd + ff(a+jj*h);
    end
    RR(kk,1) = RR(kk-1,1)/2 + h*sumodd;
    dum1 = 1;
    for ii = 2:1:kk
        dum1 = dum1*4;
        RR(kk,ii) = (dum1*RR(kk,ii-1)-RR(kk-1,ii-1))/(dum1-1);
    end
    err = abs(RR(kk,kk-1)-RR(kk-1,kk-1))/(dum1-1); % (cf) Equation (6.9.54)
    if err < tol,
        break
    end
end
RR
format long
IntRomberg = RR(kk,kk)
% Calculating true integral
syms xx
fx = xx*(1-xx)*exp(-xx);
IntTrue = int(fx,0,4)
IntTrue = double(IntTrue)
% Traperzoidal Rule and Simpson Rule
h = (b-a)/32;
xk = linspace(a,b,32+1); fk = ff(xk);
Traper = trapz(xk,fk)
Simpson = h/3*(sum(fk(1:2:32-1)) + 4*sum(fk(2:2:32)) ...
    + sum(fk(3:2:32+1)))
save('RombergTable101.txt','Traper','Simpson','-ascii')
% End of program
```

```
48 % ------------------------------------------------------------
```

이 MATLAB프로그램 RombergTable.m은 Romberg알고리즘을 사용해서 다음 정적분을 재귀적으로 계산하는 것이다.

$$I = \int_0^4 x[1-x]e^{-x}dx \tag{1}$$

이 정적분의 진짜값은 다음과 같다.

$$I = 21e^{-4} - 1 = -0.615371583336582 \tag{2}$$

이 MATLAB프로그램 RombergTable.m을 실행하면, 아래와 같은 행렬 RR을 출력한다.

```
RR =


   -0.4396         0         0         0         0         0
   -0.7611   -0.8683         0         0         0         0
   -0.6793   -0.6520   -0.6376         0         0         0
   -0.6335   -0.6183   -0.6160   -0.6157         0         0
   -0.6201   -0.6156   -0.6154   -0.6154   -0.6154         0
   -0.6166   -0.6154   -0.6154   -0.6154   -0.6154   -0.6154
```

이 행렬 RR을 Romberg열 (Romberg table 또는 Romberg array) 이라 부른다. 행렬 RR의 (k,l) 원소는 $T_{l-1}(h_{k-l})$ 이다. 특히, $(1,1)$ 원소 $T_0(h_0) = -0.4396$ 은 사다리꼴공식에 의해서 구한 것이다. 알고리즘 8.4.1을 이용해서, $(k,l-1)$ 원소와 $(k+1,l-1)$ 원소로부터 $(k+1,l)$ 원소를 계산한다. 즉, 각 원소는 그 좌측 원소와 그 위 원소로부터 구할 수 있다. 이 행렬에서 알 수 있듯이, $(5,5)$ 원소 $T_4(h_4) = -0.6154$는 유효숫자 범위 내에서 식 (2) 에서 구한 I 값과 같다. 이 경우, 적분구간을 $2^4 = 16$ 으로 나눈 것이다. 또한, 오차허용값 (tolerance) 을 만족하는 $(6,6)$ 원소는 $T_5(h_5) = -0.6153716$ 이다. 이 경우, 소구간들의 개수는 $2^5 = 32$ 이다.

반면에, 소구간들의 개수를 32 로 한 사다리꼴공식에 의한 수치적분값은 -0.6165514 이고, 소구간들의 개수를 32 로 한 Simpson공식에 의한 수치적분값은 -0.6153838 이다. 이 예제에서 볼 수 있듯이, 사다리꼴공식이나 Simpson공식보다 Romberg알고리즘이 더 좋은 결과를 보여준다. 즉, Romberg알고리즘을 사용하면, 소구간 길이가 그렇게 작지 않아도 좋은 정밀도를 갖는 수치적분을 얻을 수 있다. ■

예제 8.4.2 Python을 사용해서 예제 8.4.1을 다시 다루기 위해서, 다음 Python프로그램 RombergTable101.Py를 실행해 보자.

```
"""
% ------------------------------------------------------------------
%  Filename: RombergTable101.Py
%  Romberg-Richardson Table to calculate ff over [a,b]
%  Programmed by CBS
"""

import numpy as np
from sympy import Symbol, exp, integrate
from scipy.integrate import simps

def ff(x):
    return x*(1-x)*np.exp(-x)

a = 0; b = 4; nn = 1
h = (b-a)/nn;
tol = 1.0e-08
xk = np.linspace(a,b,nn+1)
fk = ff(xk)
IntTrap = h*sum((fk[0:nn] + fk[1:nn+1])/2)     # Traperzoidal rule

# Make Romberg-Richardson Table
RR = np.zeros((10,10))
RR[0,0] = IntTrap

for kk in range(1,10):
    h = h/2; nn = nn*2;
    sumodd = 0;
    for jj in range(0,nn-1,2):
        sumodd = sumodd + ff(a+(jj+1)*h);
    RR[kk,0] = RR[kk-1,0]/2 + h*sumodd;
    dum1 = 1;
    for ii in range(1,kk+1):
        dum1 = dum1*4;
        RR[kk,ii] = (dum1*RR[kk,ii-1]-RR[kk-1,ii-1])/(dum1-1);
    err = np.abs(RR[kk,kk-1]-RR[kk-1,kk-1])/(dum1-1);
    if err < tol:
        kTerminal = kk
        break
RR
kTerminal
RRterminal = RR[0:(kTerminal+1),0:(kTerminal+1)]
IntRomberg = RRterminal[-1,-1]

# Calculating true integral
xx = Symbol('x')
fx = xx*(1-xx)*exp(-xx);
IntTrue = integrate(fx,(xx,0,4)); IntTrue
IntTrue.evalf()   # IntTrue = np.float64(IntTrue)
# Traperzoidal Rule and Simpson Rule
h = (b-a)/32;
xk = np.linspace(a,b,32+1); fk = ff(xk);
Traper = np.trapz(fk,xk); Traper
n2 = 32
LL = sum(fk[0:n2:2])
RR = sum(fk[2:n2+2:2])
CC = sum(fk[1:n2+1:2])
```

```
Sim = h/3*(LL + 4*CC + RR)
Sim2 = simps(fk,xk)

OutX = [ Traper, Sim, Sim2 ]; OutX
np.savetxt(r'RombergTable101Py.txt', OutX, fmt='%d')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.4.1의 결과와 같다. ■

제8.5절 적응적 수치적분

적응적 수치적분(adaptive quadrature)은 적분구간을 등간격으로 나누는 것이 아니라 피적분함수에서 변동이 심한 구간은 더 잘게 나누고 변동이 별로 없는 구간은 성기게 나눈다. 이렇게 함으로써, 계산부담을 작게 하면서 오차를 줄일 수 있다. 즉, 어떤 소구간에서 계산한 수치적분값 I_0와 이 소구간을 두 조각구간들로 나누어 계산한 수치적분값들 I_{11}과 I_{12}를 더한 값 I_1을 식 (8.4.13)이나 식 (8.4.14)에 대입해서, 그 소구간에서 진짜 정적분값과 I_1 사이의 오차를 추정한다. 만약 이 추정오차가 오차허용값 tol보다 작으면, I_1을 그 소구간에서 수치적분값으로 한다. 그렇지 않으면, 소구간을 둘로 나누는 동시에 오차허용값을 tol/2로 감소시킨 다음, 각 조각구간 별로 계산한 추정오차가 새로운 오차허용값 tol/2보다 작아질 때까지 동일한 과정을 반복수행한다.

예제 8.5.1 MATLAB함수 quad.m을 사용해서 적응Simpson알고리즘(adaptive Simpson algorithm)을 적용할 수 있다. 이 MATLAB함수의 사용법을 살펴보기 위해서, 다음 MATLAB 프로그램 USEquad101.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: USEquad101.m
%  Adaptive Simpson Rule
%  Programmed by CBS
% ------------------------------------------------------------------------
function USEquad101
clear all, close all, format long
% (Example 1)
f1 = @(x) x.*(1-x).*exp(-x);
a = 0, b = 4, tol = 1.0e-08, trace = 1
Int1 = quad(f1,a,b,tol,trace)
% Plotting
xx = 0:0.001:4;
yy = f1(xx);
x0 = [ 0.0000000000    0.0678950000    0.1357900000  ...
       0.2036850000    0.2715800000    0.4073700000  ...
```

```
      0.5431600000    0.6789500000    0.8147400000  ...
      0.9505300000    1.0863200000    1.2005300000  ...
      1.3147400000    1.4289500000    1.5431600000  ...
      1.7715800000    2.0000000000    2.2284200000  ...
      2.4568400000    2.6852600000    2.9136800000  ...
      3.1852600000    3.4568400000    3.7284200000  4.0 ]
plot(xx,yy,'k',x0,0,'ro','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold')
legend('function','calculating points',1)
hold on
plot([0 4],[0,0],'k-','LineWidth',1.5)
hold off
% (Example 2)
[Int2,NoFunEva2] = quad(@myAQfun1,0,4,1.0e-08)
saveas(gcf,'USEquad101','jpg')
save('USEquad101.txt','xx','yy','Int1','Int2','-ascii')
end
% End of program
% ------------------------------------------------------------------------
function ff = myAQfun1(x)
ff = x.*(1-x).*exp(-x);
end
%------------------------------------------------------------------------
```

MATLAB함수 quad.m의 표준적 사용법은 다음과 같다.

```
>> q = quad(fun,a,b)
>> q = quad(fun,a,b,tol,trace)
>> [q,fcnt] = quad(fun,a,b,tol,trace)
```

첫 번째 입력변수 fun은 피적분함수이고, 두 번째와 세 번째 입력변수들 a와 b는 각각 적분구간의 하한과 상한을 나타낸다. 옵션인 네 번째 입력변수 tol은 오차허용값이다. 또한, 다섯 번째 입력변수 trace에 0이 아닌 값을 할당하면, 함수값을 계산하는 분할점들에 대한 정보를 출력한다. 첫 번째 출력변수 q는 수치적분값이고, 두 번째 출력변수 fcnt는 함수값을 계산하는 횟수이다.

MATLAB프로그램 USEquad101.m을 실행하면, 적응Simpson알고리즘을 적용해서 다음 정적분을 계산한다.

$$q = \int_0^4 x[1-x]e^{-x}dx \tag{1}$$

첫 번째 quad.m을 실행한 결과, 정적분 q의 수치적분값은 -0.6154이다. 옵션 trace에 1을 지정하였으므로, 함수를 계산하는 횟수 fcnt, 적분하는 소구간의 시작점 a, 소구간의 길이 $[b-a]$ 그리고 수치적분값 q를 출력한다. 이 값들을 바탕으로 그린 그래프가 그림 8.5.1이다. 그림 8.5.1에서 알 수 있듯이, 곡선의 기울기 변화가 심한 부분에서는 분할점들이 촘촘히 분포되고, 그렇지 않으면 성기게 분포되어 있다. 두 번째 quad.m에는 오차허용값으로 10^{-8}을 지정하였다. 이 경우, 두 번째 출력변수 NoFunEval2에는 이 적분에서 함수값을 계산하는 횟수가 저장된다. ■

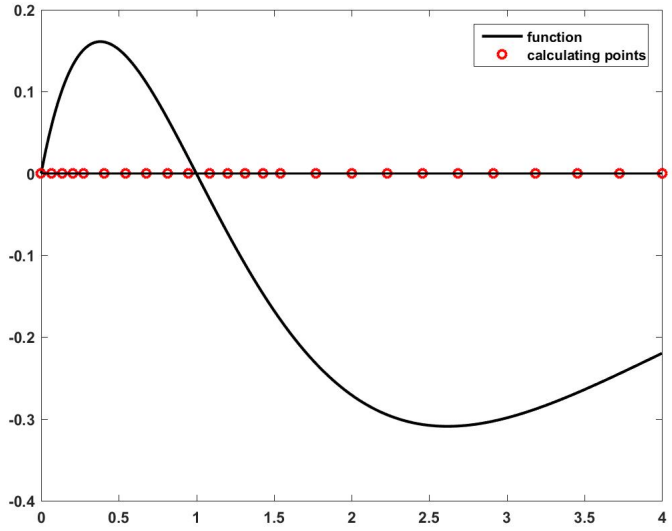


그림 8.5.1. MATLAB함수 quad.m

예제 8.5.2 Python을 사용해서 예제 8.5.1을 다시 다루기 위해서, 다음 Python프로그램 USEquad101.Py를 실행해 보자.

```
"""
%  Filename: USEquad101.Py
% https://en.wikipedia.org/wiki/Adaptive_Simpson%27s_method
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

def simpsons_rule(f,a,b):
    c = (a+b) / 2.000
    h3 = abs(b-a) / 6.0
    return h3*(f(a) + 4.0*f(c) + f(b))

def recursive_asr(f,a,b,eps,whole):
    "Recursive implementation of adaptive Simpson's rule."
    c = (a+b) / 2.0
    left = simpsons_rule(f,a,c)
    right = simpsons_rule(f,c,b)
    if abs(left + right - whole) <= 15*eps:
        return left + right + (left + right - whole)/15.0
    return recursive_asr(f,a,c,eps/2.0,left) \
           + recursive_asr(f,c,b,eps/2.0,right)

def adaptive_simpsons_rule(f,a,b,eps):
    "Calculate integral of f from a to b with max error of eps."
    return recursive_asr(f,a,b,eps,simpsons_rule(f,a,b))

## (Example 1)
f1 = lambda x: x*(1-x)*np.exp(-x)
a = 0; b = 4; tol = 1.0e-08;
Int1 = adaptive_simpsons_rule(f1, a, b, tol)
```

```
print(Int1)

# Plotting
xx =  np.linspace(0,4,4001)
yy = f1(xx);
x0 = [ 0.0000000000,     0.0678950000,     0.1357900000,
       0.2036850000,     0.2715800000,     0.4073700000,
       0.5431600000,     0.6789500000,     0.8147400000,
       0.9505300000,     1.0863200000,     1.2005300000,
       1.3147400000,     1.4289500000,     1.5431600000,
       1.7715800000,     2.0000000000,     2.2284200000,
       2.4568400000,     2.6852600000,     2.9136800000,
       3.1852600000,     3.4568400000,     3.7284200000,   4.0 ]
zero0 = np.zeros((len(x0),1))
fig = plt.figure()
plt.plot(xx,yy,'k', linewidth=2.0, label="y=f(x)")
plt.plot(x0,zero0,"r*", linewidth=2.5, label="Integral Points")
plt.legend(loc='lower left')
plt.plot(x0,zero0,"g-", linewidth=1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
fig.savefig('USEquad101aPy.jpg')

## (Example 2)
def ff(x):
    return x*(1-x)*np.exp(-x)

Int2 = quad(ff, 0, 4)
print(Int2)

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.5.1의 결과와 같다. ■

제8.6절 무한구간에서 적분

적분구간이 $(-\infty, a]$, $[a, \infty)$ 또는 $(-\infty, \infty)$와 같이 무한구간인 경우에는 Riemann합을 구해서 수치적분을 하는 것은 불가능하다. 그러나, 무한대는 수학적인 개념이므로 실제 수치적분을 하기 위해서는 무한대 대신 아주 큰 수를 사용하면 된다. 무한구간에서 정적분이 유한값이 되기 위해서는, 피적분함수 $f(x)$가 극한 $x \to \pm\infty$에 대해서 극한 $f(x) \to 0$를 만족해야 한다.

예제 8.6.1 다음과 같은 표준정규확률밀도함수에 해당하는 수치적분값을 구하자.

$$I = \int_{-\infty}^{\infty} \exp\left(-\frac{x^2}{2}\right) dx \tag{1}$$

이 정적분 I는 $\sqrt{2\pi} \approx 2.506628275$ 이다. 이 정적분을 수치적분하기 위해서, 다음 MATLAB 프로그램 ImproperIntegral101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: ImproperIntegral101.m
%  Improper Integral
%  Programmed by CBS
% ------------------------------------------------------------------
function ImproperIntegral101
clear all, close all, format long
ef = @(x) exp(-x.^2/2);
for kk = 1:1:19
    ulim(kk) = (kk-1)/2
    NN = 10*ulim(kk);
    IntTrap(kk) = 2*SRtraperzoid(ef,0,ulim(kk),NN);
    IntSim(kk) = 2*SRsimpson(ef,0,ulim(kk),NN);
    if abs(IntTrap(kk)-sqrt(2*pi)) < 1.0e-10
       break
    end
end
uplim = ulim(1:kk);
TrueInt = sqrt(2*pi)
% Plotting
plot([uplim(1),uplim(end)],[TrueInt,TrueInt],'r-.','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
hold on
plot(uplim,IntTrap(1:kk),'g-',uplim,IntSim(1:kk),'k--', ...
    'LineWidth',2)
legend('True Integral','Traperzoidal Integral', ...
    'Simpson Integral','location','SE')
hold off
% Error
format short e
errTrap = TrueInt - IntTrap(1:kk);
errSim = TrueInt - IntSim(1:kk);
fprintf('      A          Trap Error  Simpson Error\n')
disp([ uplim' errTrap' errSim' ])
saveas(gcf,'ImproperIntegral101','jpg')
save('ImproperIntegral101.txt','IntTrap','IntSim','-ascii')
end
% End of program
% ------------------------------------------------------------------
%  Filename: SRsimpson.m
%  Subroutine for Simpson Rule
% ------------------------------------------------------------------
function IntS = SRsimpson(f,a,b,N)
if mod(N,2) ~= 0, N = N+1;
end  % Make N an even number
if b == a
    IntS = 0;
else
    h = (b-a)/N;
    xk = linspace(a,b,N+1);
    fk = feval(f,xk);
    IntS = h/3*(sum(fk(1:2:N-1))+4*sum(fk(2:2:N)) ...
           +sum(fk(3:2:N+1)));
end
end
% ------------------------------------------------------------------
%  Filename: SRtraperzoid.m
%  Subroutine for Traperzoidal Rule
% ------------------------------------------------------------------
function IntT = SRtraperzoid(f,a,b,N)
if b == a
```

```
    IntT = 0;
else
    h = (b-a)/N;
    xk = linspace(a,b,N+1);
    fk = feval(f,xk);
    IntT = (h*sum(fk(1:N)) + h*sum(fk(2:N+1)))/2;
end
end
% --------------------------------------------------------------------
```

이 MATLAB프로그램 ImproperIntegral101.m을 실행하기 위해서는 MATLAB프로그램 SRtraperzoid.m과 MATLAB프로그램 SRsimpson.m이 필요하다. 이 프로그램들은 사다리꼴공식과 Simpson공식을 수행하는 서브루틴들로서, 다음 정적분을 수치적분한다.

$$I_A = \int_{-A}^{A} \exp\left(-\frac{x^2}{2}\right) dx \tag{2}$$

이 MATLAB프로그램 ImproperIntegral101.m을 실행한 결과는 다음과 같다.

```
---------------------------------------
     A         Trap Error   Simpson Error
---------------------------------------
             0  2.5066e+000  2.5066e+000
  5.0000e-001  1.5475e+000  1.5468e+000
  1.0000e+000  7.9639e-001  7.9538e-001
  1.5000e+000  3.3573e-001  3.3492e-001
  2.0000e+000  1.1450e-001  1.1405e-001
  2.5000e+000  3.1314e-002  3.1131e-002
  3.0000e+000  6.8229e-003  6.7676e-003
  3.5000e+000  1.1790e-003  1.1663e-003
  4.0000e+000  1.6101e-004  1.5880e-004
  4.5000e+000  1.7333e-005  1.7037e-005
  5.0000e+000  1.4680e-006  1.4375e-006
  5.5000e+000  9.7663e-008  9.5240e-008
  6.0000e+000  5.0975e-009  4.9493e-009
  6.5000e+000  2.0853e-010  2.0150e-010
  7.0000e+000  6.6809e-012  6.4238e-012
  ---------------------------------------
```

이 결과물에서 알 수 있듯이, $A = 7$이면 적분오차는 10^{-10}이하이다. 따라서, 구간 $[-7, 7]$에서 적분하는 것으로 충분하다. 또한, 이 MATLAB프로그램 ImproperIntegral101.m을 실행하면, 그림 8.6.1이 그려진다. 그림 8.6.1에서 알 수 있듯이, A가 커짐에 따라 수치적분은 진짜 적분값에 빠르게 수렴한다. ■

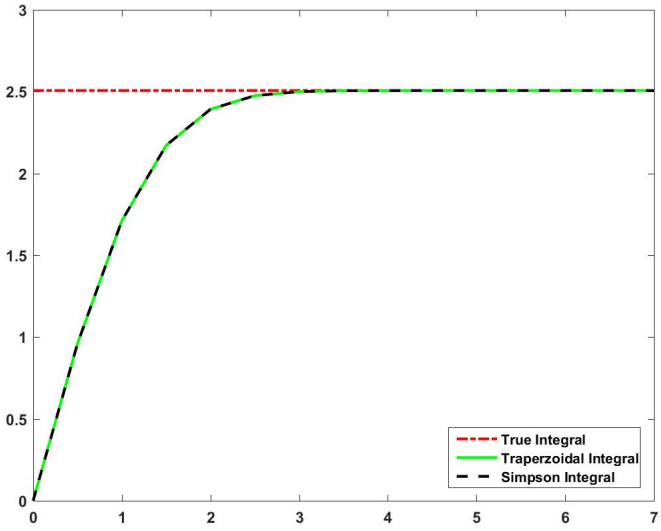


그림 8.6.1. 적분구간길이가 무한대인 수치적분

예제 8.6.2 Python을 사용해서 예제 8.6.1을 다시 다루기 위해서, 다음 Python프로그램 ImproperIntegral101.Py를 실행해 보자.

```
"""
% Filename: ImproperIntegral101.Py
% Improper Integral
% Programmed by CBS
"""

import numpy as np
from scipy.integrate import simps
import matplotlib.pyplot as plt

ef = lambda x: np.exp(-x**2/2)

ulim = np.zeros(19)
IntTrap = np.zeros(19)
IntSim = np.zeros(19)
for kk in range(1,19):
    ulim[kk] = round((kk+1)/2)
    NN = 10*ulim[kk]
    xk = np.linspace(0,ulim[kk],NN)
    fk = ef(xk)
    IntTrap[kk] = 2*np.trapz(fk,xk)
    IntSim[kk] = 2*simps(fk,xk)
    if abs(IntTrap[kk]-np.sqrt(2*np.pi)) < 1.0e-10:
       kTerminal = kk
       break
uplim = ulim[0:kTerminal+1]
TrueInt = np.sqrt(2*np.pi)

# Plotting
fig = plt.figure()
plt.plot([uplim[0],uplim[-1]],[TrueInt,TrueInt],'r-.',
```

```
          label='True Integral',lw=2.0)
plt.plot(uplim,IntTrap[0:kTerminal+1],'g-',label='Traperzoidal Integral',
          lw=2.0)
plt.plot(uplim,IntSim[0:kTerminal+1],'k--',label='Simpson Integral',lw=2.0)
plt.legend(loc='lower right')
plt.xlabel("A")
plt.ylabel("Integral")
plt.show()
fig.savefig('ImproperIntegral101Py.jpg')

# Error
errTrap = TrueInt - IntTrap[0:kTerminal+1]
errSim = TrueInt - IntSim[0:kTerminal+1]
uplimC = np.reshape(uplim,(kTerminal+1,1))
errTrapC = np.reshape(errTrap,(kTerminal+1,1))
errSimC = np.reshape(errSim,(kTerminal+1,1))
Outp = np.concatenate((uplimC,errTrapC,errSimC), axis=1)
print(Outp)

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.6.1의 결과와 같다. ■

제8.7절 Gauss구적법

Newton-Cotes식에서 수치적분에 포함된 분할점들은 적분구간 $[a,b]$를 등간격으로 나눈 분할 $\Pi = \{a = x_0 < x_1 < \cdots < x_n = b\}$의 점들 $\{x_i\}$이다. 그러나, 피적분함수가 식으로 주어져서 적분구간 내 어떤 점에서도 함수값을 구할 수 있다면, 반드시 소구간들이 등간격일 필요가 없다. 즉, 등간격이 아닌 분할점들을 잘 선택함으로써, 오차가 적은 수치적분값을 구할 수 있다. 물론, 피적분함수의 형태와 적분구간에 따라, 분할점의 선택을 달리해야 한다. 이 절에서는 분할점을 잘 선택해서 가능한 한 고차 멱함수까지 정확히 수치적분값을 얻을 수 있는 Gauss구적법(Gauss quadrature)에 대해서 살펴보자.

다음 정적분의 수치적분값을 구하는 문제를 생각해보자.

$$I = \int_a^b w(x) f(x) dx \tag{8.7.1}$$

여기서 $w(x)(>0)$는 가중함수(weight function)이다. 적분구간 $[a,b]$ 내에 n개 분할점들 $x_1 < x_2 < \cdots < x_n$과 적당한 상수들 $A_1, A_2, \cdots, A_n$에 대한 합을 다음과 같이 표기하자.

$$I_n = \sum_{k=1}^{n} A_k f(x_k) \tag{8.7.2}$$

Gauss구적법은 이 가중합 I_n을 정적분 I의 근사값으로 하는 것이다. 분할점들의 개수가 n인 Gauss구적법에서 함수 $f(x)$가 $2[n-1]$차 이하 멱함수이면, 수치적분값이 정적분과 같다. 즉, 오차없이 정적분이 구해진다.

Gauss구적법의 아이디어는 아주 단순한 것이다. 예를 들어, 어떤 함수 $f(x)$를 적분구간 $[-1,1]$에서 수치적으로 적분하기로 하자. 즉, 다음 정적분 I를 수치적분하기로 하자.

$$I = \int_{-1}^{1} f(x)dx \tag{8.7.3}$$

이 정적분 I를 다음과 같이 표기할 수 있다고 가정하자.

$$I[x_1, x_2] \doteq A_1 f(x_1) + A_2 f(x_2) \tag{8.7.4}$$

여기서 미지수들은 x_1, x_2, A_1, A_2이다. 이 미지수들을 구하기 위해서는 방정식들 4개가 필요로 한다. 우선, $f(x)$가 멱함수들 $x^i,\ \ (i=0,1,2,3)$인 경우를 살펴보자. 다음 식들이 성립한다.

$$A_1 x_1^0 + A_2 x_2^0 = \int_{-1}^{1} x^0 dx = 2 \tag{8.7.5}$$

$$A_1 x_1^1 + A_2 x_2^1 = \int_{-1}^{1} x^1 dx = 0 \tag{8.7.6}$$

$$A_1 x_1^2 + A_2 x_2^2 = \int_{-1}^{1} x^2 dx = \frac{2}{3} \tag{8.7.7}$$

$$A_1 x_1^3 + A_2 x_2^3 = \int_{-1}^{1} x^3 dx = 0 \tag{8.7.8}$$

연립방정식 (8.7.5)∼(8.7.8)을 풀면, 그 해는 다음과 같다.

$$A_1 = 1,\ \ A_2 = 1,\ \ x_1 = -\frac{1}{\sqrt{3}},\ \ x_2 = \frac{1}{\sqrt{3}} \tag{8.7.9}$$

식 (8.7.5)∼식 (8.7.9)와 적분연산자의 선형성에서 알 수 있듯이, 임의의 3차 이하 멱함수 $f(x)$에 대해서 다음 식이 성립한다.

$$f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right) = \int_{-1}^{1} f(x)dx \tag{8.7.10}$$

즉, 정적분 I를 두 점에서 함수값들의 가중합으로 나타낼 수 있다. 다음 예제는 식 (8.7.10)을 예증하기 위한 것이다.

예제 8.7.1 식 (8.7.10)을 확인하기 위해서, 다음 MATLAB프로그램 GaussQuadTest101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: GaussQuadTest101.m
%  Gaussian-Legrndre Quadrature of polynomial w/ order 3
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, format long
syms c0 c1 c2 c3 x
f = c0 + c1*x + c2*x^2 + c3*x^3
Intf = int(f,x,-1,1)
f1 = subs(f,x,-1/sqrt(3))
f2 = subs(f,x,1/sqrt(3))
GQf = f1 +f2
Err = Intf-GQf
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 GaussQuadTest101.m은 다음과 같은 3차함수 $f(x)$를 구간 $[-1,1]$에서 적분하는 것이다.

$$f(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 \tag{1}$$

여기서 계수들 c_0, c_1, c_2, c_3는 임의의 실수들이다.

이 MATLAB프로그램을 실행하면, 다음 식들이 성립함을 알 수 있다.

$$\int_{-1}^{1} f(x)dx = 2c_0 + \frac{2}{3}c_2 \tag{2}$$

$$f\left(-\frac{1}{\sqrt{3}}\right) = c_0 - \frac{\sqrt{3}}{3}c_1 + \frac{1}{3}c_2 - \frac{\sqrt{3}}{9}c_3 \tag{3}$$

$$f\left(\frac{1}{\sqrt{3}}\right) = c_0 + \frac{\sqrt{3}}{3}c_1 + \frac{1}{3}c_2 + \frac{\sqrt{3}}{9}c_3 \tag{4}$$

$$f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right) = 2\left[c_0 + \frac{1}{3}c_2\right] \tag{5}$$

식 (2)와 식 (5)에서 알 수 있듯이, 식 (8.7.10)이 성립한다. ■

예제 8.7.2 Python을 사용해서 예제 8.7.1을 다시 다루기 위해서, 다음 Python프로그램 GaussQuadTest101.Py를 실행해 보자.

```
"""
%  Filename: GaussQuadTest101.m
%  Gaussian-Legrndre Quadrature of polynomial w/ order 3
%  Programmed by CBS
"""
```

```
6
7  import numpy as np
8  from sympy import Symbol, integrate
9
10 c0 = Symbol('c0');  c1 = Symbol('c1');  c2 = Symbol('c2')
11 c3 = Symbol('c3');  x = Symbol('x')
12 f = c0 + c1*x + c2*x**2 + c3*x**3
13 Intf = integrate(f,(x,-1,1));  print(Intf)
14 f1 = f.subs(x,-1/np.sqrt(3));  print(f1)
15 f2 = f.subs(x,1/np.sqrt(3));   print(f2)
16 GQf = f1 +f2;                  print(GQf)
17 Err = Intf-GQf;                print(Err)
18
19 #  End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.1의 결과와 같다. ■

우리의 목적은 식 (8.7.1)의 정적분 I를 식 (8.7.2)의 가중합 I_n으로 나타내는 것이다. 이러한 목적을 달성하기 위해서, 우선 다음과 같은 n차 멱함수 $\alpha(x)$를 정의하자.

$$\alpha(x) \doteq \prod_{k=1}^{n} [x - x_k] \tag{8.7.11}$$

여기서 $x_1, x_2, \cdots, x_n$은 서로 다른 실수들이다. 즉, $x_1, x_2, \cdots, x_n$은 n차 방정식 $\alpha(x) = 0$의 서로 다른 근들이다. 피적분함수 $f(x)$에 대해서 다음과 같은 함수 $q(x)$를 정의하자.

$$q(x) \doteq \sum_{k=1}^{n} f(x_k) \frac{\alpha(x)}{[x - x_k]\alpha'(x_k)} \tag{8.7.12}$$

L'Hospital정리에 의해서, 다음 식이 성립함을 알 수 있다.

$$\lim_{x \to x_k} \frac{\alpha(x)}{[x - x_k]\alpha'(x_k)} = 1 \tag{8.7.13}$$

함수 $q(x)$는 $[n-1]$차 멱함수이므로, 연속이다. 따라서, 식 (8.7.12)와 식 (8.7.13)에서 알 수 있듯이, 다음 식들이 성립한다.

$$q(x_k) = f(x_k), \quad (k = 1, 2, \cdots, n) \tag{8.7.14}$$

다음과 같이 함수 $f(x)$를 $[n-1]$차 멱함수 $q(x)$와 나머지 $r(x)$의 합으로 나타내자.

$$f(x) = q(x) + r(x) \tag{8.7.15}$$

만약 나머지 항 $r(x)$ 의 적분값이 작다면, 다음 식들이 성립한다.

$$\int_a^b w(x)f(x)dx \approx \int_a^b w(x)q(x)dx = \sum_{k=1}^{n} f(x_k)\int_a^b w(x)\frac{\alpha(x)}{[x-x_k]\alpha'(x_k)}dx \tag{8.7.16}$$

다음과 같이 A_k 를 정의하자.

$$A_k \doteq \int_a^b w(x)\frac{\alpha(x)}{[x-x_k]\alpha'(x_k)}dx \tag{8.7.17}$$

식 (8.7.16) 에서 알 수 있듯이, 다음 근사식이 성립한다.

$$\int_a^b w(x)f(x)dx \approx \sum_{k=1}^{n} A_k f(x_k) \tag{8.7.18}$$

그러나, 만약 함수 $f(x)$ 가 $[n-1]$ 차 멱함수이면, $f(x) = q(x)$ 이고 나머지는 $r(x) = 0$ 이다. 따라서, 이 경우에는 식 (8.7.18) 이 근사식이 아닌 등식이 된다.

예제 8.7.3 함수 $\alpha(x)$ 가 다음과 같다고 하자.

$$\alpha(x) = [x - x_0 + h][x - x_0][x - x_0 - h] \tag{1}$$

즉, 다음 식들이 성립한다고 하자.

$$x_1 = x_0 - h, \;\; x_2 = x_0, \;\; x_2 = x_0 + h \tag{2}$$

다음 식이 성립함은 자명하다.

$$\alpha'(x) = [x - x_2][x - x_3] + [x - x_1][x - x_3] + [x - x_1][x - x_2] \tag{3}$$

또한, 가중함수는 $w(x) \equiv 1$ 이라고 가정하자. 다음 식들이 성립한다.

$$A_1 = \int_{x_1}^{x_3} \frac{\alpha(x)}{[x-x_1]\alpha'(x_2)}dx = \int_{x_1}^{x_3} \frac{[x-x_2][x-x_3]}{[x_1-x_2][x_1-x_3]}dx = \frac{h}{6} \tag{4}$$

$$A_2 = \int_{x_2}^{x_3} \frac{\alpha(x)}{[x-x_2]\alpha'(x_2)}dx = \int_{x_2}^{x_3} \frac{[x-x_1][x-x_3]}{[x_2-x_1][x_2-x_3]}dx = \frac{4h}{6} \tag{5}$$

$$A_3 = \int_{x_1}^{x_3} \frac{\alpha(x)}{[x-x_3]\alpha'(x_3)}dx = \int_{x_1}^{x_3} \frac{[x-x_1][x-x_2]}{[x_3-x_1][x_3-x_2]}dx = \frac{h}{6} \tag{6}$$

따라서, 만약 함수 $f(x)$가 2차 멱함수이면, 다음 식이 성립한다.

$$\int_{x_0-h}^{x_0+h} f(x)dx = \frac{h}{3}[f(x_0-h)+f(x_0)+f(x_0+h)] \tag{7}$$

식 (7)은 Simpson공식이다. ■

이제 우리의 문제는 어떤 $\alpha(x)$, 즉 $x_1, x_2, \cdots, x_n$을 선택하느냐는 것이다. 가장 쉽고도 널리 사용되는 방법은 주어진 가중함수 $w(x)$에 대해서 직교멱함수를 선택하는 것이다.

명제 8.7.1

서로 다른 $x_1, x_2, \cdots, x_n$에 대해서 다음 함수들을 살펴보자.

$$\alpha(x) = \prod_{k=1}^{n}[x-x_k], \quad A_k = \int_a^b w(x)\frac{\alpha(x)}{[x-x_k]\alpha'(x_k)}dx, \ (k=1,2,\cdots,n)$$

차수가 $[2n-1]$ 이하인 어떤 멱함수 $f(x)$에 대해서도 다음 등식이 성립하기 위한 필요충분조건은 구간 $[a,b]$에서 가중함수 $w(x)$에 대해 멱함수 $\alpha(x)$가 차수가 $[n-1]$ 이하인 어떤 멱함수와도 직교인 것이다.

$$\int_a^b w(x)f(x)dx = \sum_{k=1}^{n} A_k f(x_k)$$

증명. 우선 필요조건을 생각해보자. 차수가 $[2n-1]$ 이하인 어떤 멱함수 $f(x)$에 대해서 다음 등식이 성립한다고 가정하자.

$$\int_a^b w(x)f(x)dx = A_k f(x_k) \tag{1}$$

차수가 $[n-1]$ 이하인 멱함수 $Q_1(x)$에 대해서, 함수 $f(x)=\alpha(x)Q_1(x)$는 차수가 $[2n-1]$ 이하인 멱함수이다. 식 (1)에서 알 수 있듯이, 다음 식들이 성립한다.

$$\int_a^b w(x)[\alpha(x)Q_1(x)]dx = \sum_{k=1}^{n} A_k[\alpha(x_k)Q_1(x_k)] = 0 \tag{2}$$

여기서 두 번째 등호는 식 $\alpha(x_k) = 0$에 의해서 성립한다. 식 (2)가 의미하는 바는 멱함수 $\alpha(x)$가 차수가 $[n-1]$ 이하인 어떤 멱함수와도 구간 $[a,b]$에서 가중함수 $w(x)$에 대해 직교라는 것이다.

충분조건을 생각해보자. 멱함수 $\alpha(x)$가 차수가 $[n-1]$ 이하인 어떤 멱함수와도 구간 $[a,b]$에서 가중함수 $w(x)$에 대해 직교라고 가정하자. 차수가 $[2n-1]$ 이하인 멱함수 $f(x)$를 $\alpha(x)$로 나눈 몫과 나머지를 각각 $Q_2(x)$와 $r(x)$라 하면, 다음 식이 성립한다.

$$f(x) = \alpha(x)Q_2(x) + r(x) \tag{3}$$

여기서 $Q_2(x)$의 차수는 $[n-1]$ 이하이다. 식 (3)에서 알 수 있듯이, 다음 식이 성립한다.

$$\int_a^b w(x)f(x)dx = \int_a^b w(x)\alpha(x)Q_2(x)dx + \int_a^b w(x)r(x)dx \tag{4}$$

식 (4)와 직교성가정에 의해서, 다음 식이 성립한다.

$$\int_a^b w(x)f(x)dx = \int_a^b w(x)r(x)dx \tag{5}$$

멱함수 $r(x)$가 차수가 $[n-1]$ 이하이므로, 다음 식들이 성립한다.

$$\int_a^b w(x)f(x)dx = \int_a^b w(x)r(x)dx = \sum_{k=1}^{n} A_k r(x_k) \tag{6}$$

다음 식들이 성립한다.

$$f(x_k) = \alpha(x_k)Q_2(x_k) + r(x_k) = r(x_k) \tag{7}$$

여기서 첫 번째 등호는 식 (3)에 의해서, 그리고 두 번째 등호는 식 $\alpha(x_k) = 0$에 의해서 성립한다. 식 (5)$\sim$식 (7)에서 알 수 있듯이, 차수가 $[2n-1]$ 이하인 멱함수 $f(x)$에 대해서 다음 식이 성립한다.

$$\int_a^b w(x)f(x)dx = \sum_{k=1}^{n} A_k f(x_k) \tag{8}$$

■

명제 8.7.1에서 알 수 있듯이, n차 멱함수 $\alpha(x)$가 차수가 $[n-1]$ 이하인 어떤 멱함수와도 구간 $[a,b]$에서 가중함수 $w(x)$에 대해 직교이면, 차수가 $[2n-1]$ 이하인 어떤 멱함수 $f(x)$에 대해서도 다음 등식이 성립한다.

$$\int_a^b w(x)f(x)dx = \sum_{k=1}^{n} A_k f(x_k) \tag{8.7.19}$$

8.7.1 Gauss-Legendre구적법

함수 $f(x)$를 적분구간 $[-1,1]$에서 적분하기로 하자. 즉, 다음 정적분 I를 수치적분하기로 하자.

$$I = \int_{-1}^{1} f(x)dx \tag{8.7.20}$$

이 경우에 가중함수는 $w(x) \equiv 1$이다. 제2.1.2소절에서 알 수 있듯이, 구간 $[-1,1]$에서 가중함수 $w(x) \equiv 1$에 대한 직교함수는 Legendre함수이다.

명제 2.1.4에서 알 수 있듯이, Legendre함수 $P_n(x)$는 구간 $(-1,1)$에서 근 n개를 갖는다. 식 (2.1.3)∼식 (2.1.6) 그리고 명제 2.1.3을 사용해서, 각 차수 n에 대한 Legendre함수 $P_n(x)$를 구할 수 있다. 따라서, 방정식 $P_n(x)=0$의 실근들을 구할 수 있다. 예를 들어, 3차 Legendre 함수 $P_3(x)$의 근들은 $-\sqrt{0.6}, 0, \sqrt{0.6}$이다. 방정식 $P_n(x)=0$의 근들을 $x_{n,1}, x_{n,2}, \cdots, x_{n,n}$이라 하고, 이에 해당하는 계수들을 $A_{n,1}, A_{n,2}, \cdots, A_{n,n}$이라 하자. 즉, $x_{n,1}, x_{n,2}, \cdots, x_{n,n}$은 분할점들(abscissas)이다. 식 (8.7.5)∼식 (8.7.8)을 사용해서 미지수들 A_1과 A_2를 구하는 방법을 일반화하면, 다음 식들이 성립함을 알 수 있다.

$$\sum_{k=1}^{n} x_{n,k}^{j-1} A_{n,k} = \frac{1-[-1]^j}{j}, \quad (j=1,2,\cdots,n) \tag{8.7.21}$$

연립방정식 (8.7.21)를 풀면, 계수들 $A_{n,1}, A_{n,2}, \cdots, A_{n,n}$을 구할 수 있다. 이 계수들을 다음과 같이 구할 수도 있다. Weisstein [67, pp. 1731-1732]에서 알 수 있듯이, 다음 식들이 성립한다.

$$A_{n,k} = \frac{2}{nP_{n-1}(x_{n,k})P_n'(x_{n,k})} = \frac{2}{[n+1]P_{n+1}(x_{n,k})P_n'(x_{n,k})} \tag{8.7.22}$$

식 (2.1.1)과 명제 (2.1.3)을 적용해서, 다음 점화식이 성립함을 증명할 수 있다.

$$[1-x^2]P_n'(x) = nxP_n(x) + nP_{n-1}(x) \tag{8.7.23}$$

식 (8.7.23)을 식 (8.7.22)에 적용하면, 다음 식들이 성립함을 알 수 있다.

$$A_{n,k} = \frac{2}{[1-x_{n,k}^2][P_n'(x_{n,k})]^2} = \frac{2[1-x_{n,k}^2]}{[n+1]^2[P_{n+1}(x_{n,k})]^2} \tag{8.7.24}$$

예제 8.7.4 식 (8.7.21)을 사용해서, Gauss-Legendre구적법의 분할점(abscissa)과 계수를 구하기 위해서, 다음 MATLAB프로그램 GaussLegendreQuad101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename GaussLegendreQuad101.m
% Gauss-Legendre Quadrature
% Using Vandermonde matrix
% Programmed by CBS
% ------------------------------------------------------------------
function [ absci coeff ] = GaussLegendreQuad1(n);
% Input n =  order of Legendre function Pn
format long
syms x
mm1= n-1
P0 = 1;
P1 = x;
for ii = 1:1:mm1
    Pn=((2.0*ii+1)*x*P1-ii*P0)/(ii+1.0);
    P0 = P1;
    P1 = Pn;
end
if n == 1
   Pn = P1;
end
Pn = vpa(expand(Pn))
abscissa = sort(solve(vpa(Pn,32)))
% Coeffcient formula is based on Vandermonde matrix
A(1,:) = ones(1,n);
b(1) = 2;
for ii = 2:1:n
    A(ii,:) = A(ii-1,:).*abscissa';
    if mod(ii,2) == 0, b(ii) = 0;
    else b(ii) = 2/ii;
    end
end
A
b
coeff = b/A';
absci = double(abscissa);
coeff = double(coeff');
[ absci coeff ]
save('GaussLegendreQuad101.txt','coeff','absci','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하기 위해서, MATLAB커맨드행에 다음 명령을 내려보자.

```
>> [ Abscissa, Coefficient] = GaussLegendreQuad101(7)
```

이 MATLAB명령문을 실행하면, 차수가 $n = 7$인 Gauss-Legendre구적법의 분할점과 계수를 출력한다. 그 결과는 다음과 같다.

```
-----------------------------------------
      Abscissa           Coefficient
  -0.949107912342758   0.129484966168870
  -0.741531185599394   0.279705391489277
  -0.405845151377397   0.381830050505119
                   0   0.417959183673469
   0.405845151377397   0.381830050505119
   0.741531185599394   0.279705391489277
   0.949107912342758   0.129484966168870
     A         Trap Error   Simpson Error
-----------------------------------------
```

■

예제 8.7.5 Python을 사용해서 예제 8.7.4을 다시 다루기 위해서, 다음 Python프로그램 GaussLegendreQuad101.Py를 실행해 보자.

```
"""
% Filename GaussLegendreQuad101.Py
% Gauss-Legendre Quadrature
% Using Vandermonde matrix
% Programmed by CBS
"""

import numpy as np
# import pandas as pd
from sympy import Symbol, simplify, nroots, expand

n = 7
x = Symbol('x')
mm1= n-1
P0 = 1;
P1 = x;
for ii in range(1,mm1+1):
    Pn = ((2.0*ii+1)*x*P1-ii*P0)/(ii+1.0);
    P0 = P1;
    P1 = Pn;
if n == 1:
   Pn = P1;
Pn = simplify(Pn); print(Pn)
Pn = expand(Pn); print(Pn)
abscissa = nroots(Pn)
abscissa = sorted(abscissa)

# Coeffcient formula is based on Vandermonde matrix
A = np.zeros((n,n))
b = np.zeros((n,1))
```

```
A[0,:] = 1;
b[0] = 2;
for ii in range(1,n):
    A[ii,:] = A[ii-1,:]*abscissa
    if np.mod(ii,2) == 1:
       b[ii] = 0;
    else:
       b[ii] = 2/(ii+1);
coeff = np.linalg.solve(A,b); print(coeff)

# Output
absci = np.reshape(abscissa,(n,1))
Outp = np.concatenate((absci,coeff), axis=1)
print(Outp)
np.savetxt(r'GaussLegendreQuad101Py.txt', Outp, fmt='%d')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.4의 결과와 같다. ■

예제 8.7.6 식 (8.7.24)를 사용해서, Gauss-Legendre구적법의 분할점과 계수를 구하기 위해서, 다음 MATLAB프로그램 GaussLegendreQuad102.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename GaussLegendreQuad102.m
% Gauss-Legendre Quadrature
% Using formula at Weisstein (1998, pp. 1731-1732)
% Programmed by CBS
% ------------------------------------------------------------------
function  [ absci coeff ] = GaussLegendreQuad102(n);
% Input n =  order of Legendre function Pn
format long
syms x
mm1= n-1;
P0 = 1;
P1 = x;
for ii = 1:1:mm1
    Pn=((2.0*ii+1)*x*P1-ii*P0)/(ii+1.0);
    P0 = P1;
    P1 = Pn;
end
if n == 1
   Pn = P1;
end
abscissa = sort(solve(vpa(Pn,32)));
% Coefficient formula is given at Weisstein
for jj = 1:1:n
    P0 = 1;
    P1 = x;
    for kk = 1:1:n
        Pn = ((2.0*kk+1)*x*P1-kk*P0)/(kk+1.0);
        P0 = P1;
        P1 = Pn;
    end
    Pn = P1;
    dum1 = subs(Pn,x,abscissa(jj))^2;
```

```
    coeff(jj) = vpa(2*(1-abscissa(jj)^2)/(n+1)^2/dum1);
end
absci = double(abscissa);
coeff = double(coeff');
[ absci coeff ]
save('GaussLegendreQuad102.txt','coeff','absci','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하기 위해서, MATLAB커맨드행에 다음 명령을 내려보자.

```
>> [ Abscissa, Coefficient] = GaussLegendreQuad102(7)
```

이 MATLAB명령문을 실행하면, 예제 8.7.4와 동일한 결과를 얻는다. ■

예제 8.7.7 Python을 사용해서 예제 8.7.6을 다시 다루기 위해서, 다음 Python프로그램 GaussLegendreQuad102.Py를 실행해 보자.

```
"""
% Filename GaussLegendreQuad102.Py
% Gauss-Legendre Quadrature
% Using formula at Weisstein (1998, pp. 1731-1732)
% Programmed by CBS
"""

import numpy as np
from sympy import Symbol, simplify, nroots

n = 7
x = Symbol('x');
mm1= n-1;
P0 = 1;
P1 = x;
for ii in range(1,mm1+1):
    Pn=((2.0*ii+1)*x*P1-ii*P0)/(ii+1.0);
    P0 = P1;
    P1 = Pn;
if n == 1:
   Pn = P1;
Pn = simplify(Pn); print(Pn)
abscissa = nroots(Pn); print(abscissa)   # faster than solve
abscissa.sort(); print(abscissa)

# Coeffcient formula is given at Weisstein
coeff = np.zeros(n)
for jj in range(1,n+1):
    P0 = 1;
    P1 = x;
    for kk in range(1,n+1):
        Pn = ((2.0*kk+1)*x*P1-kk*P0)/(kk+1.0);
        P0 = P1;
        P1 = Pn;
    Pn = P1;
    dum0= Pn.subs(x,abscissa[jj-1]);
    dum1 = dum0**2
```

```
    coeff[jj-1] = 2*(1-abscissa[jj-1]**2)/(n+1)**2/dum1
print(coeff)

# Output
absci = np.reshape(abscissa,(n,1))
coeffi = np.reshape(coeff,(n,1))
Outp = np.concatenate((absci,coeffi), axis=1)
print(Outp)
np.savetxt(r'GaussLegendreQuad102Py.txt', Outp, fmt='%d')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.6의 결과와 같다. ■

다음 예제는 명제 8.7.1을 예증하기 위한 것이다.

예제 8.7.8 명제 8.7.1에서 알 수 있듯이, $[2n-1]$차 이하 멱함수에 대해 Gauss-Legendre 구적법을 적용해서 수치적분을 하면, 오차없는 적분값이 구해진다. 차수가 $n=3$ 경우에 대해서 이 성질이 성립함을 확인해보자.

차수가 3인 Legendre함수는 다음과 같다.

$$P_3(x) = \frac{1}{2}x[5x^2 - 3] \tag{1}$$

방정식 $P_3(x)$의 해는 다음과 같다.

$$x = -\sqrt{\frac{3}{5}},\ \ 0,\ \ \sqrt{\frac{3}{5}} \tag{2}$$

차수가 $2n-1=5$인 멱함수의 일반형은 다음과 같다.

$$f(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + g \tag{3}$$

구간 $[-1,1]$에서 이 함수의 정적분은 다음과 같다.

$$I = \int_{-1}^{1} f(x)dx = \left[\frac{a}{6}x^6 + \frac{b}{5}x^5 + \frac{c}{4}x^4 + \frac{d}{3}x^3 + \frac{e}{2}x^2 + gx\right]_{-1}^{1} = \frac{2}{5}b + \frac{2}{3}d + 2g \tag{4}$$

Legendre함수의 각 근에서 함수값은 다음과 같다.

$$f\left(-\sqrt{\frac{3}{5}}\right) = -\left[\frac{3}{5}\right]^2\sqrt{\frac{3}{5}}a + \left[\frac{3}{5}\right]^2 b - \left[\frac{3}{5}\right]\sqrt{\frac{3}{5}}b + \frac{3}{5}d - \sqrt{\frac{3}{5}}e + g \tag{5}$$

$$f(0) = g \tag{6}$$

$$f\left(\sqrt{\frac{3}{5}}\right) = +\left[\frac{3}{5}\right]^2\sqrt{\frac{3}{5}}a + \left[\frac{3}{5}\right]^2 b + \left[\frac{3}{5}\right]\sqrt{\frac{3}{5}}b + \frac{3}{5}d + \sqrt{\frac{3}{5}}e + g \tag{7}$$

식 (5)∼식 (7)에서 알 수 있듯이, 다음 식이 성립한다.

$$\frac{5}{9}f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9}f(0) + \frac{5}{9}f\left(\sqrt{\frac{3}{5}}\right) = \frac{2}{5}b + \frac{2}{3}d + 2g \tag{8}$$

식 (4)와 식 (8)에서 알 수 있듯이, 다음 식이 성립한다.

$$I = \frac{5}{9}f\left(-\sqrt{\frac{3}{5}}\right) + \frac{8}{9}f(0) + \frac{5}{9}f\left(\sqrt{\frac{3}{5}}\right) \tag{9}$$

어떤 $[2n-1]$차 멱함수의 계수들의 개수는 $2n$이다. 그러나, 그 중 n개는 적분구간 $[-1,1]$에서 기함수의 정적분이 0이라는 성질에 의해서 소거된다. 나머지 모수들은 n개 분할점들에서 함수값들 $\{f(x_k)|k = 1, 2, \cdots, n\}$을 이용해서 구할 수 있다. Gauss-Legendre구적법에서는 피적분함수를 $[2n-1]$차 멱함수로 근사시키므로 오차가 적다. ■

지금까지는 함수 $f(x)$를 적분구간 $[-1,1]$에서 수치적분하는 것을 다루었다. 그러나, 적분구간이 $[a,b]$인 경우에도 다음과 같은 Gauss-Legendre변환을 사용해서, Gauss-Legendre 구적법을 적용할 수 있다.

$$z = \frac{[b-a]x + a + b}{2} \tag{8.7.25}$$

다음 예제는 식 (8.7.25)과 Gauss-Legendre구적법을 사용해서 수치적분을 하는 예이다.

예제 8.7.9 함수 $f(x) = x[1-x]e^{-x}$의 정적분을 Gauss-Legendre구적법을 사용해서 계산하기 위해서, 다음 MATLAB프로그램 GaussLegendreQuad103.m을 사용해 보자.

```
% ------------------------------------------------------------------
%  Filename: GaussLegendreQuad103.m
%  Gauss Legendre Quadrature
%  Programmed by CBS
% ------------------------------------------------------------------
function GaussLegendreQuad103
```

```
clear all, close all, format long
f1 = @(x) x.*(1-x).*exp(-x);

% Calculating Gauss-Legendre Quadrature
a = 0, b = 4, n = 10;
syms xx
Absci = vpasolve(legendreP(n,xx) == 0)
Coeff = 2/(n+1)^2*(1-Absci.^2)./legendreP(n+1,Absci).^2
z = ((b-a)*Absci + a+b)/2;
GLQ = Coeff(1)*f1(z(1));
for kk = 2:1:n
    GLQ = GLQ + Coeff(kk)*f1(z(kk));
end
IntGLQ = GLQ*(b-a)/2

% Calculating true integral
syms xx
fx = xx*(1-xx)*exp(-xx);
IntTrue = int(fx,0,4);
IntTrue = double(IntTrue)

% Error and Save
err = IntTrue - IntGLQ
save('GaussLegendreQuad103.txt','IntTrue','IntGLQ')
end
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하기 위해서, MATLAB커맨드행에 다음 명령을 내리자.

```
>> [ Abscissa, Coefficient] = GaussLegendreQuad103(7)
```

이 MATLAB프로그램을 실행하면, 다음 정적분의 진짜값을 출력한다.

$$I = \int_0^4 x[1-x]e^{-x}dx = 21e^{-4} - 1 = -0.615371583336582 \tag{1}$$

차수가 10인 Gauss-Legendre구적법에 의한 수치적분값은 다음과 같다.

$$\int_0^4 f(x)dx = \sum_{k=1}^{10} A_{10,k} f(x_{10,k}) = -0.615371583336582 \tag{2}$$

즉, 이 수치적분값은 진짜 정적분과 같다. 즉, 배정밀도(double precision)까지 수치적분값은 오차가 0이다. 결론적으로, 앞에서 사용한 다른 수치적분값보다 적은 분할점들을 사용했으면서도 오차가 작아졌다. ■

예제 8.7.10 Python을 사용해서 예제 8.7.9을 다시 다루기 위해서, 다음 Python프로그램 GaussLegendreQuad103.Py를 실행해 보자.

```
"""
% Filename GaussLegendreQuad103.Py
% Gauss-Legendre Quadrature
% Programmed by CBS
"""

import numpy as np
from sympy import Symbol, integrate, exp

f1 = lambda x: x*(1-x)*np.exp(-x)

# Calculating Gauss-Legendre Quadrature
n = 10
Absci, Coeff = np.polynomial.legendre.leggauss(n)
a = 0; b = 4
z = ((b-a)*Absci + a+b)/2;
GLQ = Coeff[0]*f1(z[0]);
for kk in range(1,n):
    GLQ = GLQ + Coeff[kk]*f1(z[kk]);
IntGLQ = GLQ*(b-a)/2
print(IntGLQ)

# Calculating true integral
xx = Symbol('xx')
fx = xx*(1-xx)*exp(-xx);
IntTrue = integrate(fx,(xx,0,4));
print(IntTrue)
IntTrue.evalf()

# Error and Output
err = (IntTrue - IntGLQ).evalf()
Outp = [ IntGLQ, IntTrue, err ]
print(Outp)
np.savetxt(r'GaussLegendreQuad103Py.txt', Outp, fmt='%d')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.9의 결과와 같다. ■

8.7.2 Gauss-Laguerre구적법

다음 정적분 I를 수치적분하기로 하자.

$$I = \int_0^\infty e^{-x} f(x) dx \tag{8.7.26}$$

이 경우에 가중함수는 $w(x) = e^{-x}$이다. 제2.1.4소절에서 알 수 있듯이, 구간 $[0, \infty)$에서 가중함수 $w(x) = e^{-x}$에 대한 직교함수는 Laguerre함수이다.

식 (2.1.14)∼식 (2.1.18) 그리고 명제 2.1.10을 사용해서, 각 차수 n에 대한 Laguerre 함수 $L_n(x)$를 구할 수 있고, 또한 Laguerre함수 $L_n(x)$의 근을 구할 수 있다. 예를 들어,

2차 Laguerre함수 $L_2(x)$의 근들은 $2-\sqrt{2}, 2+\sqrt{2}$이다. 방정식 $L_n(x)=0$의 근들을 $x_{n,1}, x_{n,2}, \cdots, x_{n,n}$이라 하고, 이에 해당하는 계수들을 $A_{n,1}, A_{n,2}, \cdots, A_{n,n}$이라 하자. 즉, $x_{n,1}, x_{n,2}, \cdots, x_{n,n}$은 분할점들(abscissas)이다. 식 (8.7.5)∼식 (8.7.8)을 사용해서 미지수들 A_1과 A_2를 구하는 방법을 일반화하면, 다음 식들이 성립함을 알 수 있다.

$$\sum_{k=1}^{n} x_{n,k}^{j-1} A_{n,k} = \int_0^{\infty} e^{-x} x^{j-1} dx = \Gamma(j), \quad (j=1,2,\cdots,n) \tag{8.7.27}$$

연립방정식 (8.7.27)을 풀어서 계수들 $A_{n,1}, A_{n,1}, \cdots, A_{n,n}$을 구하면, 다음과 같다.

$$A_{n,k} = \frac{x_k}{[n+1]^2 [L_{n+1}(x_{n,k})]^2}, \quad (k=1,2,\cdots,n) \tag{8.7.28}$$

예제 8.7.11 Gauss-Laguerre구적법의 분할점과 계수를 구하기 위해, 다음 MATLAB프로그램 GaussLaguerreQuad101.m을 사용해 보자.

```
% ------------------------------------------------------------------
% Filename GaussLaguerreQuad101.m
% Gauss-Laguerre Quadrature
% Using MATLAB function roots.m
% Programmed by CBS
% ------------------------------------------------------------------
function  [ absci coeff ] = GaussLaguerreQuad101(n)
% Input n =  order of Laguerre function Pn
% Find abscissas
format long
if n <= 0, LP = 1;
elseif n == 1, LP = [ -1 1];
else LP0 = 1; LP1 = [ -1 1];
    for ii = 1:1:n-1
        LP = -1/(ii+1)*[LP1,0]+(2*ii+1)/(ii+1) ...
             *[0,LP1]-ii/(ii+1)*[0 0 LP0];
        LP0 = LP1; LP1 = LP;
    end
end
LP
abscissa = sort(roots(LP));
% Coeffcient formula is based on Vandermonde matrix
A(1,:) = ones(1,n);
b(1) = 1;
for ii = 2:1:n
    A(ii,:) = A(ii-1,:).*abscissa';
    b(ii) = (ii-1)*b(ii-1);
end
coeff = b/A';
absci = double(abscissa);
coeff = double(coeff');
[ absci coeff ]
save('GaussLaguerreQuad101.txt','coeff','absci','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 GaussLaguerreQuad101.m에서는 Laguerre함수의 계수들만 구한 다음, 이에 해당하는 멱함수의 근을 구하는 MATLAB함수 roots.m을 사용하였다. 이 MATLAB프로그램을 실행하기 위해서, MATLAB커맨드행에 다음 명령을 내려보자.

```
>> [ Abscissa, Coefficient] = GaussLaguerreQuad101(7)
```

이 MATLAB명령문을 실행하면, 차수가 $n = 7$인 Gauss-Laguerre구적법의 분할점과 계수를 출력한다. 그 결과는 다음과 같다.

```
-----------------------------------------
      Abscissa           Coefficient
   0.193043676560362   0.409318951701460
   1.026664895339190   0.421831277861426
   2.567876744950802   0.147126348657653
   4.900353084526287   0.020633514468668
   8.182153444563118   0.001074010143290
  12.734180291797697   0.000015865464348
  19.395727862262596   0.000000031703155
-----------------------------------------
```

■

예제 8.7.12 Python을 사용해서 예제 8.7.11을 다시 다루기 위해서, 다음 Python프로그램 GaussLaguerreQuad101.Py를 실행해 보자.

```
"""
% Filename GaussLaguerreQuad101.Py
% Gauss-Laguerre Quadrature
% Using Vandermonde matrix
% Programmed by CBS
"""

import numpy as np
import pandas as pd

# Find abscissas
zeroo = pd.Series([0])
n = 7
if n <= 0:
    LP = pd.Series([1]);
elif n == 1:
    LP = pd.Series([-1, 1]);
else:
    LP0 = pd.Series([1]);
    LP1 = pd.Series([-1, 1]);
    for ii in range(1,n):
        dum0 = pd.Series.copy(LP0);
        dum1f = pd.Series.copy(LP1);
```

```
        dum1b = pd.Series.copy(LP1);
        dum0 = pd.Series(np.concatenate(([0], \
                         dum0.values))).reset_index(drop=True);
        dum0 = pd.Series(np.concatenate(([0], \
                         dum0.values))).reset_index(drop=True);
        dum1f = pd.Series(np.concatenate(([0], \
                         dum1f.values))).reset_index(drop=True);
        dum1b = dum1b.append(zeroo).reset_index(drop=True);
        LP = -1/(ii+1)*dum1b + (2*ii+1)/(ii+1)*dum1f-ii/(ii+1)*dum0;
        LP0 = pd.Series.copy(LP1);
        LP1 = pd.Series.copy(LP);
abscis = np.roots(LP)
print(abscis)
abscissa = sorted(abscis); print(abscissa)

# Coeffcient formula is based on Vandermonde matrix
A = np.zeros((n,n))
b = np.zeros((n,1))
A[0,:] = 1;
b[0] = 1;
for ii in range(1,n):
    A[ii,:] = A[ii-1,:]*abscissa
    b[ii] = ii*b[ii-1];
coeff = np.linalg.solve(A,b); print(coeff)

# Output
absci = np.reshape(abscissa,(n,1))
Outp = np.concatenate((absci,coeff), axis=1)
print(Outp)
np.savetxt(r'GaussLaguerreQuad101Py.txt', Outp, fmt='%d')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.11의 결과와 같다. ■

다음 예제는 Gauss-Laguerre구적법을 사용해서 수치적분을 하는 예이다.

예제 8.7.13 함수 $f(x) = e^{-x}x^5$ 의 정적분을 Gauss-Laguerre구적법을 사용해서 계산하기 위해서, 다음 MATLAB프로그램 GaussLaguerreQuad102.m을 사용해 보자.

```
% ------------------------------------------------------------------------
%  Filename: GaussLaguerreQuad102.m
%  Gauss Laguerre Quadrature
%  Programmed by CBS
% ------------------------------------------------------------------------
function GaussLaguerreQuad102
clear all, close all, format long
f1 = @(x) x.^5
% Calculating Gauss-Laguerre Quadrature
n = 3;
syms xx
Abscissa = vpasolve(laguerreL(n,xx) == 0)
Coeff = Abscissa./(n+1)^2./laguerreL(n+1,Abscissa).^2
GLQ = Coeff(1)*f1(Abscissa(1));
for kk = 2:1:n
```

```
    GLQ = GLQ + Coeff(kk)*f1(Abscissa(kk));
end
GLQ
% Calculating true integral
trueftn = xx^5*exp(-xx);
IntTrue = int(trueftn,0,inf);
IntTrue = double(IntTrue)
err = IntTrue - GLQ
save('GaussLaguerreQuad102.txt','IntTrue','GLQ')
end
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 GaussLaguerreQuad102.m을 실행하면, 먼저 다음 정적분의 진짜 값을 출력한다.

$$I = \int_0^\infty e^{-x} x^5 dx = \Gamma(5) = 4! = 120 \tag{1}$$

차수가 3인 Gauss-Laguerre구적법에 의한 수치적분값은 다음과 같다.

$$\int_0^\infty f(x)dx = \sum_{k=1}^{3} A_{3,k} f(x_{3_k}) = 119.9999999999994 \tag{2}$$

즉, 이 수치적분값은 오차가 $5.8 \cdot 10^{-13}$이다. 이 예제는 명제 8.7.1을 예증한 것이다. ■

예제 8.7.14 Python을 사용해서 예제 8.7.13을 다시 다루기 위해서, 다음 Python프로그램 GaussLaguerreQuad102.Py를 실행해 보자.

```
"""
%  Filename: GaussLaguerreQuad102.Py
%  Gauss Laguerre Quadrature
%  Programmed by CBS
"""

import numpy as np
from sympy import Symbol, integrate, exp

f1 = lambda x: x**5

# Calculating Gauss-Laguerre Quadrature
n = 3
Abscissa, Coeff = np.polynomial.laguerre.laggauss(n)
GLQ = Coeff[0]*f1(Abscissa[0]);
for kk in range(1,n):
    GLQ = GLQ + Coeff[kk]*f1(Abscissa[kk]);
IntGLQ = GLQ
print(IntGLQ)

# Calculating true integral
xx = Symbol('xx')
fx = xx**5*exp(-xx);
IntTrue = integrate(fx,(xx,0,np.inf));
```

```
print(IntTrue)
IntTrue.evalf()

# Error and Output
err = (IntTrue - IntGLQ).evalf()
Outp = [ IntGLQ, IntTrue, err ]
print(Outp)
np.savetxt(r'GaussLaguerreQuad102Py.txt', Outp, fmt='%d')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.13의 결과와 같다. ■

8.7.3 Gauss-Hermite구적법

다음 정적분 I를 수치적분하기로 하자.

$$I = \int_{-\infty}^{\infty} e^{-x^2} f(x) dx \tag{8.7.29}$$

이 경우에 가중함수는 $w(x) = e^{-x^2}$ 이다. 제2.1.5소절에서 알 수 있듯이, 구간 $(-\infty, \infty)$ 에서 가중함수 $w(x) = e^{-x^2}$ 에 대한 직교함수는 Hermite함수이다.

식 (2.1.22)∼식 (2.1.25) 그리고 명제 2.1.14를 사용해서, 각 차수 n에 대한 Hermite함수 $H_n(x)$를 구할 수 있고, Hermite함수 $H_n(x)$의 근도 구할 수 있다. 예를 들어, 3차 Hermite 함수 $H_3(x)$의 근들은 $-\sqrt{1.5}, 0, \sqrt{1.5}$이다. 방정식 $H_n(x) = 0$의 근들을 $x_{n,1}, x_{n,2}, \cdots, x_{n,n}$ 이라 하고, 이에 해당하는 계수들을 $A_{n,1}, A_{n,2}, \cdots, A_{n,n}$ 이라 하자. 즉, $x_{n,1}, x_{n,2}, \cdots, x_{n,n}$ 은 분할점들(abscissas)이다. 식 (8.7.5)∼식 (8.7.8)을 사용해서 미지수들 A_1과 A_2를 구하는 방법을 일반화하면, 다음 식들이 성립함을 알 수 있다.

$$\sum_{k=1}^{n} x_k^{j-1} A_{n,k} = \int_{-\infty}^{\infty} e^{-x^2} x^{j-1} dx = \Gamma\left(\frac{j}{2}\right), \quad (j = 1, 2, \cdots, n) \tag{8.7.30}$$

여기서 변수변환 $z = x^2$을 적용하면 두 번째 등호가 성립함을 알 수 있다. 연립방정식 (8.7.30)를 풀어서 계수들 $A_{n,1}, A_{n,1}, \cdots, A_{n,n}$을 구하면, 다음과 같다. 다음 식들이 성립함을 알 수 있다.

$$A_{n,k} = \frac{\sqrt{\pi} 2^{n-1} n!}{n^2 [H_{n-1}(x_{n,k})]^2} \tag{8.7.31}$$

예제 8.7.15 Gauss-Hermite구적법의 분할점과 계수를 구하기 위해서, 다음 MATLAB 프로그램 GaussHermiteQuad101.m을 사용해 보자.

```
% ------------------------------------------------------------------
% Filename: GaussHermiteQuad101.m
% Gauss-Hermite Quadrature
% Using MATLAB function roots.m
% Programmed by CBS
% ------------------------------------------------------------------
function  [ absci coeff ] = GaussHermiteQuad101(n)
% Input n =  order of Hermite function Pn
% Find abscissas
format long
if n <= 0, HP = 1;
elseif n == 1, HP = [ 2 0 ];
else HP0 = 1; HP1 = [ 2 0 ];
    for ii = 1:1:n-1
        HP = 2*[HP1,0] - 2*ii*[0 0 HP0];
        HP0 = HP1; HP1 = HP;
    end
end
abscissa = sort(roots(HP));
% Coeffcient formula is based on Vandermonde matrix
A(1,:) = ones(1,n);
b(1) = sqrt(pi);
for ii = 2:1:n
    A(ii,:) = A(ii-1,:).*abscissa';
    if mod(ii,2) == 0, b(ii) = 0;
    else
    b(ii) = (ii-2)/2*b(ii-2);
    end
end
coeff = b/A';
absci = double(abscissa);
coeff = double(coeff');
[ absci coeff ]
save('GaussHermiteQuad101.txt','coeff','absci','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 GaussHermiteQuad101.m에서는 Hermite함수의 계수들만 구한 다음, 이에 해당하는 멱함수의 근을 구하는 MATLAB함수 roots.m을 사용하였다. 이 MATLAB 프로그램을 실행하기 위해서, MATLAB커맨드행에 다음 명령을 내려보자.

```
>> [ Abscissa, Coefficient] = GaussHermiteQuad101(7)
```

이 MATLAB명령문을 실행하면, 차수가 $n = 7$인 Gauss-Hermite구적법의 분할점과 계수를 출력한다. 그 결과는 다음과 같다.

```
-----------------------------------------
     Abscissa           Coefficient
 -2.651961356835237   0.000971781245099
 -1.673551628767471   0.054515582819130
 -0.816287882858964   0.425607252610121
                  0   0.810264617556817
  0.816287882858965   0.425607252610120
  1.673551628767469   0.054515582819130
  2.651961356835238   0.000971781245099
-----------------------------------------
```

■

예제 8.7.16 Python을 사용해서 예제 8.7.15을 다시 다루기 위해서, 다음 Python프로그램 GaussHermiteQuad101.Py를 실행해 보자.

```
"""
% Filename GaussHermiteQuad101.Py
% Gauss-Hermite Quadrature
% Using Vandermonde matrix
% Programmed by CBS
"""

import numpy as np
import pandas as pd

# Find abscissas
zeroo = pd.Series([0])
n = 7
if n <= 0:
    HP = pd.Series([1]);
elif n == 1:
    HP = pd.Series([2,0]);
else:
    HP0 = pd.Series([1]);
    HP1 = pd.Series([2,0]);
    for ii in range(1,n):
        dum0 = pd.Series.copy(HP0);
        dum1 = pd.Series.copy(HP1);
        dum0 = pd.Series(np.concatenate(([0], \
                        dum0.values))).reset_index(drop=True);
        dum0 = pd.Series(np.concatenate(([0], \
                        dum0.values))).reset_index(drop=True);
        dum1 = dum1.append(zeroo).reset_index(drop=True);
        HP = 2*dum1 - 2*ii*dum0;
        HP0 = pd.Series.copy(HP1);
        HP1 = pd.Series.copy(HP);
abscis = np.roots(HP)
print(abscis)
abscissa = sorted(abscis); print(abscissa)

# Coeffcient formula is based on Vandermonde matrix
A = np.zeros((n,n))
```

```
b = np.zeros((n,1))
A[0,:] = 1;
b[0] = np.sqrt(np.pi);
for ii in range(1,n):
    A[ii,:] = A[ii-1,:]*abscissa
    if np.mod(ii,2)==1:
        b[ii]=0;
    else:
        b[ii] = (ii-1)/2*b[ii-2];
coeff = np.linalg.solve(A,b); print(coeff)

# Output
absci = np.reshape(abscissa,(n,1))
Outp = np.concatenate((absci,coeff), axis=1)
print(Outp)
np.savetxt(r'GaussHermiteQuad101Py.txt', Outp, fmt='%d')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.15의 결과와 같다. ■

다음 예제는 Gauss-Hermite구적법을 사용해서 수치적분을 하는 예이다.

예제 8.7.17 함수 $f(x) = e^{-x^2}x^4$의 정적분을 Gauss-Hermite구적법을 사용해서 계산하기 위해서, 다음 MATLAB프로그램 GaussHermiteQuad102.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: GaussHermiteQuad102.m
%  Gauss Hermite Quadrature
%  Programmed by CBS
% ------------------------------------------------------------------------
clear all, close all, format long
f1 = @(x) x^4
% Calculating Gauss-Hermite Quadrature
n = 3;
syms xx
Abscissa = vpasolve(hermiteH(n,xx) == 0)
Coeff = sqrt(pi)*2^(n-1)*gamma(n+1)./n^2./hermiteH(n-1,Abscissa).^2
GLQ = Coeff(1)*f1(Abscissa(1));
for kk = 2:1:n
    GLQ = GLQ + Coeff(kk)*f1(Abscissa(kk));
end
GLQ
% Calculating true integral
syms xx
fx = exp(-xx^2)*xx^4;
IntTrue = int(fx,-inf,inf);
IntTrue = double(IntTrue)
err = IntTrue - GLQ
save('GaussHermiteQuad102.txt','IntTrue','GLQ')
% End of program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램 GaussHermiteQuad102.m을 실행하면, 다음 정적분의 진짜값을 출력한다.

$$I = \int_0^\infty e^{-x^2} x^4 dx = \Gamma\left(\frac{5}{2}\right) = \frac{3}{4}\Gamma\left(\frac{1}{2}\right) = \frac{3}{4}\sqrt{\pi} = 1.329340388179137 \tag{1}$$

차수가 3인 Gauss-Hermite구적법에 의한 수치적분값은 다음과 같다.

$$\int_0^\infty f(x)dx = \sum_{k=1}^{3} A_{3,k} f(x_{3,k}) = 1.329340388179137 \tag{2}$$

즉, 이 수치적분값은 오차가 0이다. 따라서, 이 예제는 명제 8.7.1을 예증한 것이다. ■

예제 8.7.18 Python을 사용해서 예제 8.7.17을 다시 다루기 위해서, 다음 Python프로그램 GaussHermiteQuad102.Py를 실행해 보자.

```
"""
%  Filename: GaussHermiteQuad102.Py
%  Gauss Hermite Quadrature
%  Programmed by CBS
"""

import numpy as np

f1 = lambda x: x**4

# Calculating Gauss-Hermite Quadrature
n = 3
Abscissa, Coeff = np.polynomial.hermite.hermgauss(n)
GLQ = Coeff[0]*f1(Abscissa[0]);
for kk in range(1,n):
    GLQ = GLQ + Coeff[kk]*f1(Abscissa[kk]);
IntGLQ = GLQ
print(IntGLQ)

# Calculating true integral
IntTrue = 3/4*np.sqrt(np.pi)
print(IntTrue)

# Error and Output
err = IntTrue - IntGLQ
Outp = [ IntGLQ, IntTrue, err ]
print(Outp)
np.savetxt(r'GaussHermiteQuad102Py.txt', Outp)

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.17의 결과와 같다. ■

8.7.4 Gauss-Chebyshev구적법

다음 정적분 I를 수치적분하기로 하자.

$$I = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)dx \tag{8.7.32}$$

여기서 가중함수는 $w(x) = [1-x^2]^{-1/2}$이다. 제2.1.3소절에서 알 수 있듯이, 구간 $(-1,1)$에서 가중함수 $w(x) = [1-x^2]^{-1/2}$에 대한 직교함수는 Chebyshev함수이다.

명제 2.1.7의 증명에서 알 수 있듯이, 각 차수 n에 대한 Chebyshev함수 $T_n(x)$의 근들은 다음과 같다.

$$x_{n,k} = \cos\left(\frac{\pi}{2n}[2k-1]\right), \quad (k = 1, 2, \cdots, n) \tag{8.7.33}$$

예를 들어, 3차 Chebyshev함수 $T_3(x)$의 근들은 $-\sqrt{3}/2, 0, \sqrt{3}/2$이다. 방정식 $T_n(x)$의 근들을 $x_{n,1}, x_{n,2}, \cdots, x_{n,n}$이라 하고, 이에 해당하는 계수들을 $A_{n,1}, A_{n,2}, \cdots, A_{n,n}$이라 하자. 즉, $x_{n,1}, x_{n,2}, \cdots, x_{n,n}$은 분할점들(abscissas)이다. 변수변환 $x = \sin\theta$를 이용해서, 다음 식이 성립함을 증명할 수 있다.

$$b_j \doteq \int_{-1}^{1} \frac{1}{\sqrt{2-x^2}} x^{j-1}dx = \begin{cases} \dfrac{1\cdot 3\cdots[2n-3][2n-1]}{2\cdot 4\cdots[2n-2][2n]}\dfrac{\pi}{2}, & (j = 2n+1) \\ 0, & (\text{otherwise}) \end{cases} \tag{8.7.34}$$

식 (8.7.5)∼식 (8.7.8)을 사용해서 미지수들 A_1과 A_2를 구하는 방법을 일반화하면, 다음 식들이 성립함을 알 수 있다.

$$\sum_{k=1}^{n} x_k^{j-1} A_{n,k} = b_j, \quad (j = 1, 2, \cdots, n) \tag{8.7.35}$$

연립방정식 (8.7.35)를 풀면, 계수들 $A_{n,1}, A_{n,2}, \cdots, A_{n,n}$을 구할 수 있다. 이 계수들을 구하는 다른 방법은 다음과 같다. Weisstein [67, pp. 401-402]에서 알 수 있듯이, 다음 식들이 성립한다.

$$A_{n,k} = -\frac{\pi}{T_{n+1}(x_{n,k})T_n'(x_{n,k})} \tag{8.7.36}$$

식 (2.1.7)과 식 (8.7.33)에서 알 수 있듯이, 다음 식들이 성립한다.

$$T_{n+1}(x_{n,k}) = [-1]^k \sin\left(\frac{\pi}{2n}[2k-1]\right), \quad (k = 1, 2, \cdots, n) \tag{8.7.37}$$

$$T_n'(x_{n,k}) = \frac{[-1]^{k+1} n}{\sin\left(\dfrac{\pi}{2n}[2k-1]\right)}, \quad (k = 1, 2, \cdots, n) \tag{8.7.38}$$

식 (8.7.37)과 식 (8.7.38)을 식 (8.7.36)에 대입하면, 다음 식들이 성립함을 알 수 있다.

$$A_{n,k} = \frac{\pi}{n}, \quad (k = 1, 2, \cdots, n) \tag{8.7.39}$$

예제 8.7.19 Gauss-Chebyshev구적법의 분할점과 계수를 구하기 위해서, 다음 MATLAB 프로그램 GaussChebyshevQuad101.m을 사용해 보자.

```
% ------------------------------------------------------------------
% Filename: GaussChebyshevQuad101.m
% Gauss-Chevyshev Quadrature
% Using Vandermonde matrix
% Programmed by CBS
% ------------------------------------------------------------------
function  [ absci coeff ] = GaussChebyshevQuad101(n)
% Input n =  order of Chevyshev function Pn
format long
kk = (1:1:n)';
abscissa = cos(pi/2/n.*(2*kk-1))
% Coeffcient formula is based on Vandermonde matrix
A(1,:) = ones(1,n);
b(1) = pi;
for ii = 2:1:n
    A(ii,:) = A(ii-1,:).*abscissa';
    if mod(ii,2) == 0, b(ii) = 0;
    else b(ii) = b(ii-2)*(ii-2)/(ii-1);
    end
end
coeff = b/A';
absci = double(abscissa);
coeff = double(coeff');
ErrorCoeff = coeff - pi/n
[ absci coeff ErrorCoeff]
save('GaussChebyshevQuad101.txt','coeff','absci','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 GaussChebyshevQuad101.m을 실행하기 위해서, MATLAB커맨드행에 다음 명령을 내려보자.

```
>> [ Abscissa, Coefficient] = GaussChebyshevQuad101(7)
```

이 MATLAB명령문을 실행하면, 차수가 $n = 7$인 Gauss-Chebyshev구적법의 분할점과 계수를 출력한다. 그 결과는 다음과 같다.

```
-----------------------------------------
      Abscissa           Coefficient
  0.974927912181824   0.448798950512827
  0.781831482468030   0.448798950512830
  0.433883739117558   0.448798950512822
  0.000000000000000   0.448798950512835
 -0.433883739117558   0.448798950512820
 -0.781831482468030   0.448798950512833
 -0.974927912181824   0.448798950512826
-----------------------------------------
```

■

예제 8.7.20 Python을 사용해서 예제 8.7.19을 다시 다루기 위해서, 다음 Python프로그램 GaussChebyshevQuad101.Py를 실행해 보자.

```
"""
% Filename GaussChebyshevQuad101.Py
% Gauss-Chebyshev Quadrature
% Programmed by CBS
"""

import numpy as np

# Find abscissas and Coefficients
n = 7
kk = np.arange(1,n+1)
abscissa = np.cos(np.pi/2/n*(2*kk-1))
coeff = np.pi/n*np.ones(n)

# Output
absci = np.reshape(abscissa,(n,1))
coef = np.reshape(coeff,(n,1))
Outp = np.concatenate((absci,coef), axis=1)
print(Outp)
np.savetxt(r'GaussChebyshevQuad101Py.txt', Outp, fmt='%d')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.19의 결과와 같다. ■

다음 예제는 Gauss-Chebyshev구적법을 사용해서 수치적분을 하는 예이다.

예제 8.7.21 함수 $f(x) = x^6/\sqrt{1-x^2}$의 정적분을 Gauss-Chebyshev구적법을 사용해서 계산하기 위해서, 다음 MATLAB프로그램 GaussChebyshevQuad102.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: GaussChebyshevQuad102.m
%  Gauss Chevyshev Quadrature
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, format long
ff = inline('x.^6')
% Calculating Gauss-Chebyshev Quadrature
n = 4
absissa = cos(((2*(1:n) - 1)/(2*n))*pi)';
GCQ = sum(pi/n*ff(absissa))
% Calculating true integral
syms xx
fx = xx^6/sqrt(1-xx.^2);
IntTrue = int(fx,-1,1)
IntTrue = double(IntTrue)
err = IntTrue - GCQ
save('GaussChebyshevQuad102.txt','IntTrue','GCQ','-ascii')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 GaussChebyshevQuad102.m을 실행하면, 다음 정적분의 진짜값을 출력한다.

$$I = \int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} x^6 dx = \frac{5\pi}{16} = 0.981747704246810 \tag{1}$$

차수가 4인 Gauss-Chebyshev구적법에 의한 수치적분값은 다음과 같다.

$$\int_{-1}^{1} f(x)dx = \sum_{k=1}^{4} A_{4,k} f(x_{4,k}) = 0.981747704246810 \tag{2}$$

즉, 배정밀도(double precision)까지 수치적분값은 오차가 0이다. 따라서, 이 예제는 명제 8.7.1을 예증한 것이다. ■

예제 8.7.22 Python을 사용해서 예제 8.7.21을 다시 다루기 위해서, 다음 Python프로그램 GaussChebyshevQuad102.Py를 실행해 보자.

```
"""
%  Filename: GaussChebyshevQuad102.Py
%  Gauss Chebyshev Quadrature
%  Programmed by CBS
"""

import numpy as np

f1 = lambda x: x**6

# Calculating Gauss-Chebyshev Quadrature
n = 4
```

```
Abscissa, Coeff = np.polynomial.chebyshev.chebgauss(n)
print(Abscissa, Coeff)
GLQ = Coeff[0]*f1(Abscissa[0]);
for kk in range(1,n):
    GLQ = GLQ + Coeff[kk]*f1(Abscissa[kk]);
IntGLQ = GLQ
print(IntGLQ)

# Calculating true integral
IntTrue = 5/16*np.pi
print(IntTrue)

# Error and Output
err = IntTrue - IntGLQ
Outp = [ IntGLQ, IntTrue, err ]
print(Outp)
np.savetxt(r'GaussChebyshevQuad102Py.txt', Outp)

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.21의 결과와 같다. ■

8.7.5 Gauss-Lobbato구적법

다음 정적분 I를 수치적분하기로 하자.

$$I = \int_{-1}^{1} f(x)dx \tag{8.7.40}$$

이 경우에 가중함수는 $w(x) \equiv 1$이다. 제2.1.2소절에서 알 수 있듯이, 구간 $[-1, 1]$에서 가중함수 $w(x) \equiv 1$에 대한 직교함수는 Legendre함수이다. Gauss-Lobbato구적법이 Legendre함수를 사용하는 Gauss-Legendre구적법과 다른 점은 끝점들 $x = -1$과 $x = 1$이 분할점이라는 것이다. 따라서, 차수가 n인 Gauss-Lobbato구적법에서 분할점을 $[n-2]$개만 선택하면 된다. 이 $[n-2]$개 점들은 차수가 $[n-1]$인 Legendre함수의 도함수 $P'_{n-1}(x)$의 근이다. 방정식 $P'_{n-1}(x) = 0$의 근들을 $x_{n,2}, x_{n,3}, \cdots, x_{n,n-1}$이라 하고, 이에 해당하는 계수들을 $A_{n,1}, A_{n,2}, \cdots, A_{n,n}$이라 하자. 즉, $x_{n,1}, x_{n,2}, \cdots, x_{n,n}$은 분할점들(abscissas)이다. Weisstein [67, pp. 1786-1787]에서 알 수 있듯이, 다음 식들이 성립한다.

$$A_{n,k} = \begin{cases} \dfrac{2}{n[n-1]}, & (k = 1, n) \\ \dfrac{2}{n[n-1][P_{n-1}(x_{n,k})]^2}, & (k = 2, 3, \cdots, n-1) \end{cases} \tag{8.7.41}$$

예제 8.7.23 MATLAB함수 quadl.m을 사용해서 적응Lobbato구적법(adaptive Gauss-Lobbato quadrature)을 적용할 수 있다. 이 함수의 사용법을 살펴보기 위해서, 다음 MATLAB 프로그램 USEquadl101.m("use quad el one zero one dot m")을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: USEquadl101.m (Use quad el one zero one dot m)
%  Adaptive Gauss-Lobbato Quadrature
%  Programmed by CBS
% ------------------------------------------------------------------
function USEquadl101
clear all, close all, format long
% (Example 1)
f1 = @(x) x.*(1-x).*exp(-x);
a = 0, b = 4, tol = 1.0e-08, trace = 1
Int1 = quadl(f1,a,b,tol,trace)
% Plotting
xx = 0:0.001:4;
yy = f1(xx);
x0 = [ 0.0000000000    0.0678950000    0.1357900000  ...
       0.2036850000    0.2715800000    0.4073700000  ...
       0.5431600000    0.6789500000    0.8147400000  ...
       0.9505300000    1.0863200000    1.2005300000  ...
       1.3147400000    1.4289500000    1.5431600000  ...
       1.7715800000    2.0000000000    2.2284200000  ...
       2.4568400000    2.6852600000    2.9136800000  ...
       3.1852600000    3.4568400000    3.7284200000  4.0 ]
plot(xx,yy,'k',x0,0,'ro','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('function','calculating points',1)
hold on
plot([0 4],[0,0],'k-','LineWidth',1.5)
hold off
saveas(gcf,'USEquadl101','jpg')
save('USEquadl101.txt','Int1','Int2','-ascii')
end
% End of program
% ------------------------------------------------------------------
```

MATLAB함수 quadl.m의 표준적 사용법은 다음과 같다.

```
>> q = quadl(fun,a,b)
>> q = quadl(fun,a,b,tol,trace)
>> [q,fcnt] = quadl(fun,a,b,tol,trace)
```

첫 번째 입력변수 fun은 피적분함수이고, 두 번째와 세 번째 입력변수들 a와 b는 적분구간의 하한과 상한을 나타낸다. 옵션인 네 번째 입력변수 tol은 오차허용값이다. 또한, 다섯 번째 입력변수 trace에 0이 아닌 값을 할당하면, 함수값을 계산하는 분할점들에 대한 정보를 출력한다. 첫 번째 출력변수 q는 수치적분값이고, 두 번째 출력변수 fcnt는 함수값을 계산하는 횟수이다.

MATLAB프로그램 USEquadl101.m을 실행하면, 적응Simpson알고리즘을 적용해서 다음 정적분을 계산한다.

$$q = \int_0^4 x[1-x]e^{-x}dx \tag{1}$$

이 quadl.m을 실행한 결과, 정적분 q의 수치적분값은 -0.6154이다. 옵션 trace에 1을 지정하였으므로, 함수를 계산하는 횟수 fcnt, 적분하는 소구간의 시작점 a, 소구간의 길이 b-a 그리고 수치적분값 q를 출력한다. 이 값들을 바탕으로 그린 그래프가 그림 8.7.1이다. 그림 8.7.1에서 알 수 있듯이, 곡선의 기울기 변화가 심한 부분에서는 분할점들이 촘촘히 분포되고, 그렇지 않으면 성기게 분포되어 있다. ■

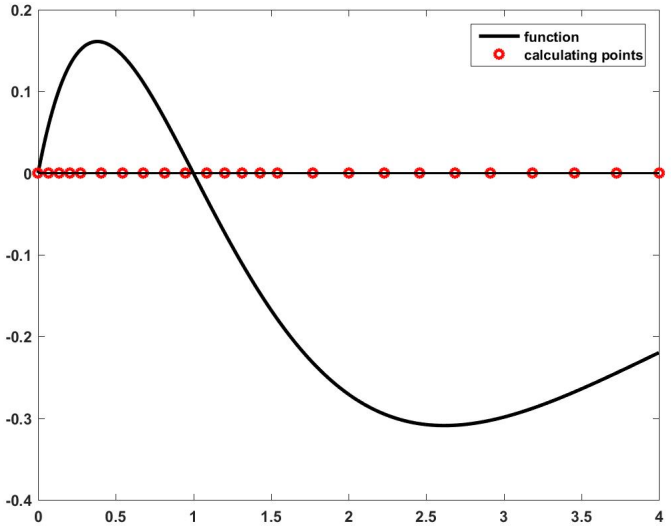


그림 8.7.1. MATLAB함수 quadl.m

예제 8.7.24 Python을 사용해서 예제 8.7.23을 다시 다루기 위해서, 다음 Python프로그램 USEquad101.Py를 실행해 보자.

```
"""
%  Filename: USEquad101.Py
% https://en.wikipedia.org/wiki/Adaptive_Simpson%27s_method
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import quad

def simpsons_rule(f,a,b):
    c = (a+b) / 2.000
    h3 = abs(b-a) / 6.0
```

```
    return h3*(f(a) + 4.0*f(c) + f(b))

def recursive_asr(f,a,b,eps,whole):
    "Recursive implementation of adaptive Simpson's rule."
    c = (a+b) / 2.0
    left = simpsons_rule(f,a,c)
    right = simpsons_rule(f,c,b)
    if abs(left + right - whole) <= 15*eps:
        return left + right + (left + right - whole)/15.0
    return recursive_asr(f,a,c,eps/2.0,left) \
           + recursive_asr(f,c,b,eps/2.0,right)

def adaptive_simpsons_rule(f,a,b,eps):
    "Calculate integral of f from a to b with max error of eps."
    return recursive_asr(f,a,b,eps,simpsons_rule(f,a,b))

## (Example 1)
f1 = lambda x: x*(1-x)*np.exp(-x)
a = 0; b = 4; tol = 1.0e-08;
Int1 = adaptive_simpsons_rule(f1, a, b, tol)
print(Int1)

# Plotting
xx =  np.linspace(0,4,4001)
yy = f1(xx);
x0 = [ 0.0000000000,     0.0678950000,     0.1357900000,
       0.2036850000,     0.2715800000,     0.4073700000,
       0.5431600000,     0.6789500000,     0.8147400000,
       0.9505300000,     1.0863200000,     1.2005300000,
       1.3147400000,     1.4289500000,     1.5431600000,
       1.7715800000,     2.0000000000,     2.2284200000,
       2.4568400000,     2.6852600000,     2.9136800000,
       3.1852600000,     3.4568400000,     3.7284200000,   4.0 ]
zero0 = np.zeros((len(x0),1))
fig = plt.figure()
plt.plot(xx,yy,'k', linewidth=2.0, label="y=f(x)")
plt.plot(x0,zero0,"r*", linewidth=2.5, label="Integral Points")
plt.legend(loc='lower left')
plt.plot(x0,zero0,"g-", linewidth=1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
fig.savefig('USEquad101aPy.jpg')

## (Example 2)
def ff(x):
    return x*(1-x)*np.exp(-x)

Int2 = quad(ff, 0, 4)
print(Int2)

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.7.23의 결과와 같다. ■

8.7.6 MATLAB과 Python의 적분함수들

MATLAB함수 trapz.m을 사용해서 사다리꼴공식을 적용할 수 있다. MATLAB함수들 quad.m이나 quadl.m은 모두 Gander & Gautschi (2000)이 제시한 알고리즘을 바탕으로 한다. 피적분함수가 평활하지 않고 또한 높은 정밀도를 요구하지 않는 경우에는 quad.m을 사용하고, 반면에 피적분함수가 평활하거나 높은 정밀도를 요구하는 경우에는 quadl.m을 사용한다. 또한, 단변량함수의 정적분을 구하기 위해서, MATLAB함수 integral.m을 사용할 수 있다. Python에서는 scipy패키지의 함수 integrate를 사용해서 정적분을 구할 수 있다.

예제 8.7.25 MATLAB함수 integral.m을 사용하는 예로서 다음 MATLAB프로그램 UseIntegral101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: UseIntegral101.m
%  Use MATLAB function Integral.m
%  https://kr.mathworks.com/help/matlab/ref/integral.html#examples
% ------------------------------------------------------------------
clear all, clear all, format long
% Example 1: Improper Integral
fun = @(x) exp(-x.^2).*log(x).^2;
q1 = integral(fun,0,Inf)

% Example 2: Parameterized Function
fun = @(x,c) 1./(x.^3-2*x-c);
  % Evaluate the integral from x=0 to x=2 at c=5.
q2 = integral(@(x)fun(x,5),0,2)

% Example 3: Singularity at Lower Limit
fun = @(x)log(x);
  % Evaluate with the default error tolerances.
q31 = integral(fun,0,1)
  % Evaluate with 12 decimal places of accuracy.
q32 = integral(fun,0,1,'RelTol',0,'AbsTol',1e-12)

% Example 4: Complex Contour Integration Using Waypoints
fun = @(z) 1./(2*z-1);
  % Integrate over the triangular path from 0 to 1+1i to 1-1i to 0
  % by specifying waypoints.
q4 = integral(fun,0,0,'Waypoints',[1+1i,1-1i])

% Example 5: Vector-Valued Function
fun = @(x)sin((1:5)*x);
q5 = integral(fun,0,1,'ArrayValued',true)

% Example 6: Improper Integral of Oscillatory Function
fun = @(x)x.^5.*exp(-x).*sin(x);
  % Adjusting the absolute and relative tolerances
q6 = integral(fun,0,Inf,'RelTol',1e-8,'AbsTol',1e-13)
% End of program
% ------------------------------------------------------------------
```

첫 번째 예제는 다음 특이적분(improper integral)을 하기 위한 것이다.

$$I_1 = \int_0^{\infty} \exp\left(-\frac{x^2}{2}\right) [\ln(x)]^2 \, dx \tag{1}$$

이 적분값은 $I_1 = 1.947522220295560$ 이다.

두 번째 예제는 모수(parameter)를 포함하는 함수를 적분하기 하기 위한 것이다.

$$I_2(c) = \int_0^2 \frac{1}{x^3 - 2x - c} dx \tag{2}$$

적분값 $I_2(5)$ 는 -0.460501533846733 이다.

세 번째 예제는 다음 정적분을 계산할 때 오차허용범위(error tolerance)를 조정하기 위한 것이다.

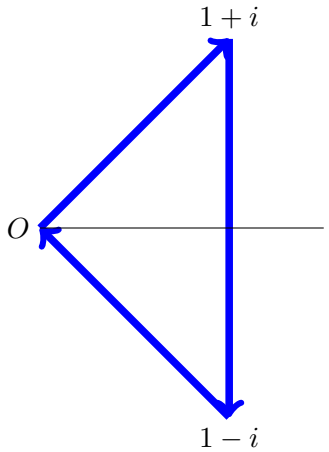
$$I_3 = \int_0^1 \ln(x) dx \tag{3}$$

디폴트 오차적용범위를 사용할 때, 이 적분값은 $I_{31} = -1.000000010959678$ 이다. 오차허용범위가 10^{-12} 일 때, 이 적분값은 $I_{32} = -1.000000000000010$ 이다.

네 번째 예제는 다음 복소적분(contour integral)을 하기 위한 것이다.

$$I_4 = \int_C \frac{1}{2z - 1} dz \tag{4}$$

여기서 원점 $0 + 0i$ 를 출발해서 점 $1 + i$ 와 점$1 - i$ 을 거쳐 다시 원점으로 돌아오는 적분경로(contour) C 는 다음과 같다.



이 적분값은 $I_4 = -0.000000000000000 - 3.141592653589799i = -\pi i$ 이다.

다섯 번째 예제는 벡터값을 갖는 다음과 같은 적분을 하기 위한 것이다.

$$I_5 = \begin{bmatrix} \int_0^1 \sin(x)dx \\ \int_0^1 \sin(2x)dx \\ \int_0^1 \sin(3x)dx \\ \int_0^1 \sin(4x)dx \\ \int_0^1 \sin(5x)dx \end{bmatrix} \tag{5}$$

이 벡터적분값은 다음과 같다.

$$I_5 = \begin{bmatrix} 0.4597, & 0.7081, & 0.6633, & 0.4134, & 0.1433 \end{bmatrix}^t \tag{6}$$

여섯 번째 예제는 다음과 같이 진동하는 함수를 적분하기 위한 것이다.

$$I_6 = \int_0^\infty x^5 \exp(-x) \sin(x)dx \tag{7}$$

여기서는 상대오차허용범위를 10^{-8} 그리고 절대오차허용범위를 10^{-13} 으로 제한하였다. 적분값 I_6 는 -14.999999999998364 이다. ■

예제 8.7.26 Python을 사용해서 예제 8.7.25을 다시 다루기 위해서, 다음 Python프로그램 UseIntegrate101.Py를 실행해 보자.

```
"""
%  Filename: UseIntegrate101.Py
%  Use Python function scipy.integrate
%  https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html
"""

import numpy as np
import scipy.integrate as integrate

# Example 1: Improper Integral
fun1 = lambda x: np.exp(-x**2)*np.log(x)**2
q1 = integrate.quad(fun1, 0, np.inf)
print(q1)

# Example 2: Parameterized Function
def fun2(x,c): return 1/(x**3-2*x-c)
# E valuate the integral from x=0 to x=2 at c=5.
c = 5
q2 = integrate.quad(fun2,0,2,args=(c))
```

```
print(q2)

# Example 3: Singularity at Lower Limit
fun3 = lambda x: np.log(x);
 # Evaluate with the default error tolerances.
q31 = integrate.quad(fun3,0,1)
print(q31)
 # Evaluate with 12 decimal places of accuracy.
q32 = integrate.quad(fun3,0,1,epsabs=1e-12,epsrel=0)
print(q32)

# Example 5: Vector-valued function
def fun5(x,n): return np.sin(n*x)
q5vec = np.zeros(5)
for nn in range(0,5):
    q5vec[nn] = integrate.quad(fun5, 0, 1, args=(nn+1))[0]
print(q5vec)

# Example 6: Improper Integral of Oscillatory Function
fun6 = lambda x: x**5*np.exp(-x)*np.sin(x);
  # Adjusting the absolute and relative tolerances
q6 = integrate.quad(fun6,0,np.Inf,epsabs=1e-13,epsrel=1e-8)
print(q6)

# Example 11
import scipy.special as special
q11 = integrate.quad(lambda x: special.jv(2.5,x), 0, 4.5)
print(q11)

# Example 12
q12 = np.sqrt(2/np.pi)*(18.0/27*np.sqrt(2)*np.cos(4.5) -
      4.0/27*np.sqrt(2)*np.sin(4.5)  +
      + np.sqrt(2*np.pi)*special.fresnel(3/np.sqrt(np.pi))[0])
print(q12)

# Example 13
def fun13(x, a, b): return a*x**2 + b
a = 2
b = 1
q13 = integrate.quad(fun13, 0, 1, args=(a,b))
print(q13)

# Example 14
def fun14(t, n, x):
    return np.exp(-x*t) / t**n
def expint(n, x):
    return integrate.quad(fun14, 1, np.inf, args=(n, x))[0]
vec_expint = np.vectorize(expint)
q14a = vec_expint(3, np.arange(1.0, 4.0, 0.5))
print(q14a)
q14b = special.expn(3, np.arange(1.0,4.0,0.5))
print(q14b)

# Example 15
q15 = integrate.quad(lambda x: expint(3, x), 0, np.inf)
print(q15)
q15True = 1/3
error15 = q15True - q15[0]
print(error15)

# End of Program
```

이 Python프로그램 UseIntegrate101.Py와 MATLAB프로그램 UseIntegral101.m은 동일하지 않다. 그러나, 공통된 문제들은 동일한 결과를 보여준다. ■

제8.8절 다중적분

다변수함수의 정적분을 구하는 다중적분을 수치적으로 구하기 위해서도 Newton-Cotes법이나 Gauss구적법을 사용할 수 있다. 다중적분법은 원자산이 복수인 금융파생상품이나 포트폴리오를 평가하는데 유용하다. 이 절에서는 다중적분에 대해 간단히 살펴보자.

먼저, MATLAB의 내장함수들을 사용해서 다중적분을 하는 방법들을 살펴보자. MATLAB함수 int.m을 사용해서 다중적분의 부정적분과 수치적분을 구할 수 있다.

예제 8.8.1 MATLAB함수 int.m을 다중적분에 적용하는 방법들을 살펴보기 위해서, 다음 MATLAB프로그램 MultipleInteg101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: MultipleInteg101.m
%  Multiple Integration 1
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, format long
syms x y a b c
Cube1 = int(int(30,x,0,10),y,0,20)
Cube2 = int(int(c,x,0,a),y,0,b)
Cone1 = int(int(30*(1-x/10-y/20),x,0,10*(1-y/20)),y,0,20)
dum = int(c*(1-x/a-y/b),x,0,a*(1-y/b))
Cone2 = int(dum,y,0,b)
% End of Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 MultipleInteg101.m의 첫 번째 명령문은 가로가 10, 세로가 20 그리고 높이가 30인 직육면체의 부피를 다중적분을 이용해서 구하는 것이다. 그 결과는 다음과 같다.

$$\int_0^{20}\left[\int_0^{10} 30dx\right]dy = 6000 \tag{1}$$

두 번째 명령문은 가로가 a, 세로가 b 그리고 높이가 c인 직육면체의 부피를 다중적분을 이용해서 구하는 것이다. 그 결과는 다음과 같다.

$$\int_0^{b}\left[\int_0^{a} cdx\right]dy = abc \tag{2}$$

세 번째 명령문은 가로가 10, 세로가 20, 그리고 높이가 30인 삼각뿔의 부피를 다중적분을 이용해서 구하는 것이다. 이 삼각뿔은 다음 평면들에 둘러싸인 형태이다.

$$\frac{x}{10} + \frac{y}{20} + \frac{z}{30} = 1, \quad x \geq 0, \quad y \geq 0, \quad x \geq 0 \tag{3}$$

이 삼각뿔의 부피는 다음과 같다.

$$\begin{aligned} &\int_0^{20} \left\{ \int_0^{10[1-y/20]} 30 \left[1 - \frac{x}{10} - \frac{y}{20} \right] dx \right\} dy \\ &= \int_0^{20} \frac{300}{2} \left[1 - \frac{y}{20} \right]^2 dy = -\frac{300}{6} \left[1 - \frac{y}{20} \right]^3 \Big|_0^{20} = 1000 \end{aligned} \tag{4}$$

네 번째 명령문은 가로가 a, 세로가 b, 그리고 높이가 c인 삼각뿔의 부피를 다중적분을 이용해서 구하는 것이다. 이 삼각뿔은 다음 평면들에 둘러싸인 형태이다.

$$\frac{x}{a} + \frac{y}{b} + \frac{z}{c} = 1, \quad x \geq 0, \quad y \geq 0, \quad x \geq 0 \tag{5}$$

이 삼각뿔의 부피는 다음과 같다.

$$\begin{aligned} &\int_0^{b} \left\{ \int_0^{a[1-y/b]} c \left[1 - \frac{x}{a} - \frac{y}{b} \right] dx \right\} dy \\ &= \int_0^{b} -\frac{ac}{2b^2} \left[b - y \right]^2 dy = -\frac{ac}{6b^2} \left[b - y \right]^3 \Big|_0^{b} = \frac{abc}{6} \end{aligned} \tag{4}$$

■

예제 8.8.2 Python을 사용해서 예제 8.8.1을 다시 다루기 위해서, 다음 Python프로그램 MultipleIntegral101.Py를 실행해 보자.

```
"""
%  Filename: MultipleIntegral101.m
%  Use Python function scipy.integrate
%  https://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html
"""

import numpy as np
import sympy as sp
import scipy.integrate as integrate


# Example 1
q1 = integrate.dblquad(lambda x, y: 30, 0, 10, lambda x: 0, lambda x: 20)
```

```
print(q1)

# Example 2
xx = sp.Symbol('xx')
yy = sp.Symbol('yy')
a = sp.Symbol('a'); b = sp.Symbol('b'); c = sp.Symbol('c')
q2a = sp.integrate(c,(xx,0,a))
print(q2a)
q2  = sp.integrate(q2a,(yy,0,b))
print(q2)

# Example 3
def f(x,y): return 30*(1-x/10-y/20)
def bounds_y(): return [0, 20]
def bounds_x(y): return [0, 10*(1-y/20)]
q3 = integrate.nquad(f, [bounds_x, bounds_y])
print(q3)

# Example 4
xx = sp.Symbol('xx')
yy = sp.Symbol('yy')
a = sp.Symbol('a'); b = sp.Symbol('b'); c = sp.Symbol('c')
q4a = sp.integrate(c*(1-xx/a-yy/b),(xx,0,a*(1-yy/b)))
print(q4a)
q4  = sp.integrate(q4a,(yy,0,b))
print(q4)

# Example 11
q11 = integrate.dblquad(lambda x, y: x*y, 0, 0.5,
                                lambda x: 0, lambda x: 1-2*x)
print(q11)

# Example 12
def f(x, y): return x*y
def bounds_y(): return [0, 0.5]
def bounds_x(y): return [0, 1-2*y]
q12 = integrate.nquad(f, [bounds_x, bounds_y])
print(q12)

# Example 13
n = 5
def f13(t, x): return np.exp(-x*t) / t**n
q13 = integrate.nquad(f13, [[1, np.inf],[0, np.inf]])
print(q13)

# Example 14
n = 5
f14 = lambda t, x: np.exp(-x*t)/t**n
q14 = integrate.dblquad(f14, 0, np.inf, lambda x: 1, lambda x: np.inf)
print(q14)

# Example 15
def q15(n):
    return integrate.dblquad(lambda t, x: np.exp(-x*t)/t**n, 0, np.inf,
                                          lambda x: 1, lambda x: np.inf)
for ii in range(1,11):
    print(q15(ii))

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 8.8.1의 결과와 같다. ■

다음 예제는 다중적분을 사용해서 극좌표로 주어진 적분문제를 푸는 것이다.

예제 8.8.3 극좌표에 관한 적분문제를 다중적분으로 푸는 방법을 살펴보기 위해서, 다음 MATLAB프로그램 MultipleInteg102.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: MultipleInteg102.m
%  Multiple Integration 2: Polar Coordinate System
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, format long
% (Example 1) Polar Rose
% Polar coordinate system
theta = linspace(0,2*pi,361);
r = 5*cos(3*theta);
subplot(2,2,1)
polar(theta,r,'r')
set(gcf,'color','w')
% Cartesian coordinate system
[x y] = pol2cart(theta,r);
[x2 y2] = pol2cart(theta,5);
subplot(2,2,2)
hold on
plot(x2,y2,'k',x,y,'r','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
axis equal off
hold off
syms rr tt a
A = 3*int(int(rr,rr,0,a*cos(3*tt)),tt,-pi/6,pi/6)
% (Example 2) Cycloid
x = 3*(theta-sin(theta)); y = 3*(1-cos(theta));
subplot(2,2,3)
plot(x,y,'r',[-0.1*pi,6.1*pi],[0,0],'b--','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
axis equal
xx = a*(tt - sin(tt)); yy = a*(1-cos(tt));
dxx = diff(xx)
B = int(yy*dxx,tt,0,2*pi)
% (Example 3) Astroid
x = 2*cos(theta).^3; y = 2*sin(theta).^3;
subplot(2,2,4)
plot(x,y,'r',[-2.3 2.3],[0 0],'b--',[0 0], ...
     [-2.3 2.3],'b--','LineWidth',2)
set(gca,'fontsize',11,'fontweigh','bold')
set(gcf,'color','w')
axis equal off
xx = a*cos(tt)^3; yy = a*sin(tt)^3;
dxx = diff(xx)
C = 4*int(yy*dxx,tt,pi/2,0)
saveas(gcf,'MultipleInteg102','jpg')
% End of Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램의 첫 번째 예제는 다음과 같이 극좌표로 주어진 장미그림(polar

rose) 의 넓이를 구하는 것이다.

$$r = 5\cos(2\Theta), \quad (0 \le \Theta \le 2\pi) \tag{1}$$

먼저 MATLAB함수 polar.m을 사용해서 이 장미그림을 그린 것이 그림 8.8.1의 좌측상단에 있는 그래프이다. 이 장미그림에는 꽃잎이 3개이다. 즉, 꽃잎이 3개인 장미 (rose with 3 petals) 이다. MATLAB함수 pol2cart.m을 사용해서 식 (1) 을 직교좌표계로 나타낸 다음, 이 장미그림을 그린 것이 그림 8.8.1의 우측상단에 있는 그래프이다. 이 꽃잎들의 넓이를 구하기 위해서, 5 대신 a 를 사용해서 다음과 같이 다중적분을 한다.

$$A = 3\int_{-\pi/6}^{\pi/6}\left[\int_0^{a\cos(3\Theta)} r dr\right] d\Theta = 3\int_{-\pi/6}^{\pi/6}\frac{1}{2}[a\cos(3\Theta)]^2 d\Theta = \frac{\pi}{4}a^2 \tag{2}$$

두 번째 예제는 다음 식들로 주어지는 사이클로이드 (cycloid) 의 면적을 구하는 것이다.

$$x = 3[\Theta - \sin(\Theta)], \quad y = 3[1 - \cos(\Theta)], \quad (0 \le \Theta \le 2\pi) \tag{3}$$

이 사이클로이드를 그린 것이 그림 8.8.1의 좌측하단에 있는 그래프이다. 이 사이클로이드와 X 축으로 둘러 쌓인 넓이를 구하기 위해서, 3 대신 a 를 사용해서 다음과 같이 적분을 한다.

$$B = \int_0^{2\pi} y(\Theta)dx(\Theta) = \int_0^{2\pi} a[1-\cos(\Theta)]\cdot a[1-\cos(\Theta)]d\Theta = 3\pi a^2 \tag{4}$$

세 번째 예제는 다음 식들로 주어지는 아스트로이드 (astroid) 의 면적을 구하는 것이다.

$$x = 2\cos^3(\Theta), \quad y = 2\sin^3(\Theta), \quad (0 \le \Theta \le 2\pi) \tag{5}$$

이 아스트로이드를 그린 것이 그림 8.8.1의 우측하단에 있는 그래프이다. 이 아스트로이드로 둘러 쌓인 넓이를 구하기 위해서, 다음과 같은 적분을 한다.

$$C = 4\int_{2\pi}^{0} y(\Theta)dx(\Theta) = 4\int_{2\pi}^{0} a\sin^3(\Theta)\cdot[-3a\cos^2(\Theta)\sin(\Theta)d\Theta] = \frac{3}{8}\pi a^2 \tag{6}$$

■

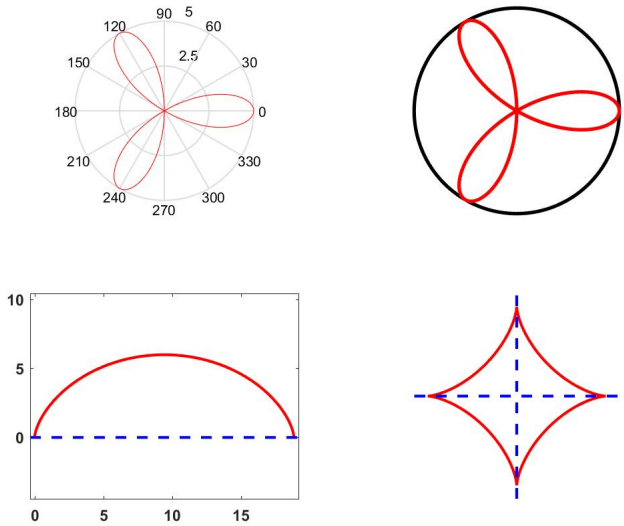


그림 8.8.1. 극좌표와 적분

예제 8.8.4 Python을 사용해서 예제 8.8.3을 다시 다루기 위해서, 다음 Python프로그램 MultipleIntegral102.Py를 실행해 보자.

```
"""
%  Filename: MultipleIntegral102.Py
%  Multiple Integration 2: Polar Coordinate System
%  Programmed by CBS
"""

import numpy as np
import sympy as sp
# import scipy.integrate as integrate
import matplotlib.pyplot as plt

# Polar Rose
fig = plt.figure()

## Polaar Coordinates
theta = np.arange(0,2*np.pi,0.01);
r = 5*np.cos(3*theta);
ax = plt.subplot(221, projection='polar')
ax.plot(theta, r, lw=2,color='r')
ax.set_rmax(5)
ax.set_rticks(range(1,6))      # less radial ticks
ax.set_rlabel_position(180)  # get radial labels
ax.grid(True)

## Cartesian coordinates
def pol2cart(rho,phi):
    x = rho * np.cos(phi);  y = rho * np.sin(phi)
    return(x, y)
x2, y2 = pol2cart(r,theta);
x2a, y2a = pol2cart(5,theta);
ax = plt.subplot(222)
ax.plot(x2,y2,lw=2,color='r')
ax.plot(x2a,y2a,lw=1.5,color='b')
```

```
ax.axis('equal')

## Cycloid
ax = plt.subplot(223)
x3 = 3*(theta-np.sin(theta)); y3 = 3*(1-np.cos(theta));
ax.plot(x3,y3,'r',lw=2)
ax.plot([-0.1*np.pi,6.1*np.pi],[0,0],'b--',lw=2)
ax.axis('equal')

## Astroid
ax = plt.subplot(224)
x4 = 2*(np.sin(theta))**3; y4 = 2*(np.cos(theta))**3;
ax.plot(x4,y4,'r',lw=2)
ax.plot([-2.3, 2.3],[0, 0],'b--',lw=2)
ax.plot([0, 0],[-2.3, 2.3],'b--',lw=2)
ax.axis('equal')

plt.show()
fig.savefig('MultipleIntegral102Py.jpg')

# Integration

## Area of Petals
rr = sp.Symbol('rr')
tt = sp.Symbol('tt')
a = sp.Symbol('a')
qAa = sp.integrate(rr,(rr,0,a*sp.cos(3*tt)))
print(qAa)
qA  = 3*sp.integrate(qAa,(tt,-np.pi/6,np.pi/6))
print(qA)
print(4*qA)

## Area of Cycloid
xx = a*(tt - sp.sin(tt)); yy = a*(1-sp.cos(tt));
dxx = sp.diff(xx,tt); print(dxx)
qB  = sp.integrate(yy*dxx,(tt,0,2*np.pi))
print(qB)
print(qB/3)

## Area of Astroid
xx = a*sp.cos(tt)**3; yy = a*sp.sin(tt)**3;
dxx = sp.diff(xx,tt); print(dxx)
qC  = 4*sp.integrate(yy*dxx,(tt,np.pi/2,0))
print(qC)
print(8/3*qC)

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 8.8.3의 결과와 같다. ■

다음은 MATLAB함수 dblquad.m을 사용해서 2중적분을 구하거나 MATLAB함수 triplequad.m을 사용해서 3중적분을 구하는 예제이다.

예제 8.8.5 MATLAB함수 dblquad.m을 사용해서 적분문제를 다중적분으로 푸는 방법을 살펴보기 위해서, 다음 MATLAB프로그램 USEdblquad101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: USEdblquad101.m
%  Use dblquad.m
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, format long
% (Example 1)
F1 = @(x,y) 1/(2*pi)*exp(-(x.^2+y.^2)/2)
DQ1 = dblquad(F1,-4,4,-4,4)
% (Example 2)
DQ2 = dblquad(@(x,y) sqrt(max(1-(x.^2+y.^2),0)), -1, 1, -1, 1)
% (Example 3)
DQ3 = dblquad(@(x,y) 1/(2*pi)*exp(-(x.^2+y.^2)/2) ...
              .*(x.^2+y.^2<=64/pi),-5,5, -5,5)
% (Example 4)
DQ4 = dblquad(F1,0,1,0,1)
% (Example 5)
F5 = @(x,y,z) (x.^2 + y.^2 + z.^2 <= 9 )
DQ5 = triplequad(F5,-3,3, -3,3, -3,3)
% End of program
%------------------------------------------------------------------
```

첫 번째 예제는 다음 정적분을 계산하는 것이다.

$$I_1 = \int_{-4}^{4}\int_{-4}^{4} \frac{1}{2\pi}\exp\left(-\frac{1}{2}[x^2+y^2]\right) dxdy \tag{1}$$

이 정적분은 2변량 정규분포에 관한 것이다. 정적분 I_1은 1에 가까워야 할 것이다. 첫 번째 MATLAB함수 dblquad.m을 사용해서 계산한 수치적분값은 다음과 같다.

$$I_1 = 0.999868868602910 \tag{2}$$

두 번째 예제는 다음 정적분을 계산하는 것이다.

$$I_2 = \int_{-1}^{1}\int_{-1}^{1} \sqrt{1-[x^2+y^2]}dxdy \tag{3}$$

이 정적분 I_2는 반경이 1인 반구의 부피 $2\pi/3 = 2.094395102393195$이다. 두 번째 MATLAB 함수 dblquad.m을 사용해서 계산한 수치적분값은 다음과 같다.

$$I_2 = 2.094410945109232 \tag{4}$$

세 번째 예제는 다음 정적분을 계산하는 것이다.

$$I_3 = \int_{-5}^{5}\int_{-5}^{5} \frac{1}{2\pi}\exp\left(-\frac{1}{2}[x^2+y^2]\right) 1_{\{x^2+y^2\leq\frac{64}{\pi}\}}(x,y)\ dxdy \tag{5}$$

이 정적분은 2변량 정규분포에 관한 것이다. 정적분 I_3은 1에 가까워야 할 것이다. 세 번째 MATLAB함수 dblquad.m을 사용해서 계산한 수치적분값은 다음과 같다.

$$I_3 = 0.999963284466323 \tag{6}$$

네 번째 예제는 다음 정적분을 계산하는 것이다.

$$I_4 = \int_0^1 \int_0^1 \frac{1}{2\pi} \exp\left(-\frac{1}{2}[x^2 + y^2]\right) dxdy \tag{7}$$

이 정적분은 2변량 정규분포에 관한 것이다. 네 번째 MATLAB함수 dblquad.m을 사용해서 계산한 수치적분값은 다음과 같다.

$$I_4 = 0.116516233168631 \tag{8}$$

다섯 번째 예제는 다음 정적분을 계산하는 것이다.

$$I_5 = \iiint\limits_{x^2+y^2+z^2 \le 9} 1 dxdydz \tag{9}$$

이 정적분 I_5는 반경이 3인 반구의 부피 $36\pi = 113.0973355292326$이다. MATLAB함수 triplequad.m을 사용해서 계산한 수치적분값은 다음과 같다.

$$I_5 = 113.0974114288376 \tag{10}$$

■

예제 8.8.6 Python을 사용해서 예제 8.8.5을 다시 다루기 위해서, 다음 Python프로그램 USEdblquad101.Py를 실행해 보자.

```
"""
% Filename: USEdblquad101.Py
% Double integral using scipy.integrate.dblquad
% Triple integral using scipy.integrate.tplquad
% Programmed by CBS
"""

import numpy as np
from scipy.integrate import dblquad, tplquad
```

```
10
11 # Example 1
12 f1 = lambda x, y: 1/(2*np.pi)*np.exp(-(x**2 + y**2)/2)
13 q1 = dblquad(f1, -4, 4, lambda x: -4, lambda x: 4)
14 print(q1)
15
16 # Example 2
17 def f2(x, y): return np.sqrt(1-(x**2 + y**2))
18 q2 = dblquad(f2,-1,1,lambda x:-np.sqrt(1-x**2),lambda x:np.sqrt(1-x**2))
19 print(q2)
20 q2X = dblquad(f2, -1, 1, lambda x: -1, lambda x: 1)
21 print(q2X)
22
23 # Example 3
24 f3 = lambda x, y: 1/(2*np.pi)*np.exp(-(x**2 + y**2)/2)
25 maxY = lambda x: np.sqrt(64/np.pi-x**2)
26 maxX = np.sqrt(64/np.pi)
27 q3 = dblquad(f3, -maxX, maxX, lambda x: -np.sqrt(64/np.pi-x**2),
28              lambda x: np.sqrt(64/np.pi-x**2))
29 print(q3)
30 f3X = lambda x, y: 1/(2*np.pi)*np.exp(-(x**2 + y**2)/2) \
31                    *(x**2 + y**2 <= 64/np.pi)
32 q3X = dblquad(f3X, -5, 5, lambda x: -5, lambda x: 5)
33 print(q3X)
34
35 # Example 4
36 def f4(x, y, z): return 1
37 q4 = tplquad(f4,-3,3,lambda x:-np.sqrt(9-x**2), lambda x:np.sqrt(9-x**2),
38              lambda x,y:-np.sqrt(9-x**2-y**2),
39              lambda x,y:np.sqrt(9-x**2-y**2) )
40 print(q4)
41
42 # End of program
```

이 Python프로그램을 실행한 결과는 예제 8.8.5의 결과와 같다. ■

함수 $f(x,y)$의 적분영역 $R=\{(x,y)|a\le x\le b,\ \ c(x)\le y\le d(x)\}$에서 정적분을 다음과 같이 표기한다.

$$I=\int_a^b\int_{c(x)}^{d(x)} f(x,y)dxdy \tag{8.8.1}$$

이 정적분의 수치적분은 다음과 같은 형태이다.

$$I_Q=\sum_{m=0}^{M}\beta_m\sum_{n=0}^{N}\alpha_n f(x_m,y_{m,n}) \tag{8.8.2}$$

여기서 계수들 α_n과 β_m은 앞에서 다룬 여러 수치적분법 중 어느 것을 택하느냐에 따라 정해진다. 유의할 점은 X축에 대한 적분과 Y축에 관한 적분이 동일한 수치적분법을 택할 필요는 없다는 것이다.

예제 8.8.7 함수 $f(x,y)=x^2y$의 적분영역 $R=\{(x,y)|-1\le x\le 1,\ -1\le y\le 1\}$에서 정적분은 다음과 같다.

$$I=\int_{-1}^{1}\int_{-1}^{1}x^2ydxdy=\int_{-1}^{1}x^2\left[\int_{-1}^{1}ydy\right]dx=0 \tag{1}$$

이 정적분을 Gauss-Legendre구적법으로 구해보자. 차수가 2인 Legendre함수는 다음과 같다.

$$P_2(x)=\frac{3}{2}\left[x^2-\frac{1}{3}\right] \tag{2}$$

따라서, 방정식 $P_2(x)=0$의 해는 다음과 같다.

$$x=-\frac{1}{\sqrt{3}},\quad \frac{1}{\sqrt{3}} \tag{3}$$

다음 식들이 성립한다.

$$f\left(-\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{3}}\right)=-\frac{1}{3\sqrt{3}},\qquad f\left(-\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}\right)=\frac{1}{3\sqrt{3}} \tag{4}$$

$$f\left(\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{3}}\right)=-\frac{1}{3\sqrt{3}},\qquad f\left(\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}\right)=\frac{1}{3\sqrt{3}} \tag{5}$$

따라서, 다음 식이 성립한다.

$$\begin{aligned}&1\cdot 1\cdot f\left(-\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{3}}\right)+1\cdot 1\cdot f\left(-\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}\right)\\&+1\cdot 1\cdot f\left(\frac{1}{\sqrt{3}},-\frac{1}{\sqrt{3}}\right)+1\cdot 1\cdot f\left(\frac{1}{\sqrt{3}},\frac{1}{\sqrt{3}}\right)=0\end{aligned} \tag{6}$$

식 (6)에서 알 수 있듯이, Gauss-Legendre구적법으로 계산한 2중적분은 0이다. 이 수치적분값은 식 (1)의 정적분과 일치한다. 즉, 오차가 없다. ■

예제 8.8.8 사다리꼴공식을 사용해서 2중적분을 구해보기 위해서, 다음 MATLAB프로그램 DoubleTraperzoidal101.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: DoubleTraperzoidal101.m
%  Double Integration 1 using Traperzoidal Rule
%  Programmed by CBS
% ------------------------------------------------------------------
clear all, close all, format long
```

```
f = @(x,y) 1/(2*pi)*exp(-(x.^2+y.^2)/2)
a = -4, b = 4
for Ndum = 1:1:20
    N(Ndum) = Ndum*5;
    Nx = N(Ndum); Ny = N(Ndum);
    hx = (b-a)/Nx;
    xm = linspace(a,b,Nx+1);
    for nn = 1:1:Nx
        d = sqrt(64/pi-(xm(nn) + xm(nn+1))^2/4);
        c = -d;
        hy = (d-c)/Ny;
        ym = linspace(c,d,Ny+1);
        fmn = feval(f,xm(nn),ym);
        sumY(nn) = hy*(sum(fmn(1:Ny)) + sum(fmn(2:Ny+1)))/2;
    end
    IntT(Ndum) = hx*sum(sumY);
end
IntT
plot(N,IntT,'g-',N,IntT,'k.','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf Number of Subintevals/Axis','fontsize',12)
ylabel('\bf Traperzoidal Integral','fontsize',12)
axis([0 90 0.9950 1.0005])
saveas(gcf,'DoubleTraperzoidal101','jpg')
save('DoubleTraperzoidal101.txt','IntT','-ascii')
% End of Program
% --------------------------------------------------------------------
```

이 MATLAB프로그램 DoubleTraperzoidal101.m은 다음 정적분을 계산하는 것이다.

$$I = \int_{-4}^{4}\int_{-4}^{4} \frac{1}{2\pi} \exp\left(-\frac{1}{2}[x^2+y^2]\right) dxdy \tag{1}$$

MATLAB함수 dblquad.m을 사용해서 계산한 수치적분값은 다음과 같다.

$$I = 0.999868868602910 \tag{2}$$

이 MATLAB프로그램 DoubleTraperzoidal101.m을 수행한 결과, 각 축별 소구간들 개수 N에 대한 사다리꼴공식에 의한 수치적분값은 다음과 같다. 이 결과물에서 있듯이, $N = 10$ 정도만 되어도 이 수치적분값은 식 (2)의 값과 아주 가까움을 알 수 있다. 그림 8.8.2에는 소구간들 개수 N에 대한 수치적분값이 그려져 있다. 그림 8.8.2에서 알 수 있듯이, 정적분 (1)을 계산할 때는 사다리꼴공식을 사용하는 2중적분이 유용하다.

```
--------------------------------------
    N                Numerical Integral
    5                 0.995560530492315
   10                 0.999831537019735
   15                 0.999875151319439
   20                 0.999891105499553
   25                 0.999898640373151
   30                 0.999902776622551
   35                 0.999905285951327
--------------------------------------
```

■

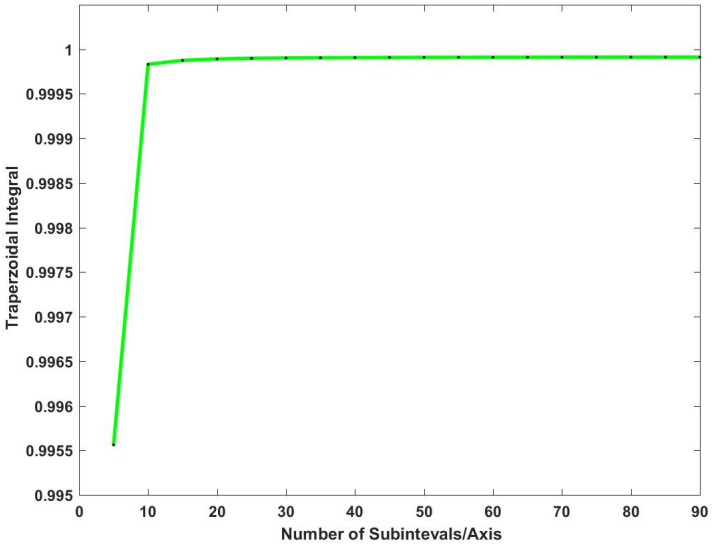


그림 8.8.2. 사다리꼴공식을 이용한 2중적분

예제 8.8.9 Python을 사용해서 예제 8.8.8을 다시 다루기 위해서, 다음 Python프로그램 DoubleTraperzoidal101.Py를 실행해 보자.

```
"""
%  Filename: DoubleTraperzoidal101.Py
%  Double Integration 1 using Traperzoidal Rule
%  Programmed by CBS
"""

import numpy as np
import matplotlib.pyplot as plt

f = lambda x,y: 1/(2*np.pi)*np.exp(-(x**2+y**2)/2)
fvec = np.vectorize(f)

```

```
a = -4; b = 4
N = [1]*20
sumY = np.zeros(100)
IntT = np.zeros(20)
for Ndum in range(0,20):
    N[Ndum] = np.int((Ndum+1)*5);
    Nx = N[Ndum];
    Ny = N[Ndum];
    hx = (b-a)/Nx;
    xm = np.linspace(a,b,Nx+1);
    for nn in range(0,Nx):
        d = np.sqrt(64/np.pi-(xm[nn] + xm[nn+1])**2/4);
        c = -d;
        hy = (d-c)/Ny;
        ym = np.linspace(c,d,Ny+1);
        fmn = fvec(xm[nn],ym)
        sumY[nn] = hy*(np.sum(fmn[0:Ny]) + np.sum(fmn[1:Ny+1]))/2;
    IntT[Ndum] = hx*np.sum(sumY);
IntT

# Plotting
fig = plt.figure()
plt.plot(N,IntT,'g-',lw=2.5)
plt.plot(N,IntT,'k.',lw=2.5)
plt.xlabel('Number of Subintevals/Axis')
plt.ylabel('Traperzoidal Integral')
plt.axis([0, 90, 0.9950, 1.0005])
plt.show()
fig.savefig('DoubleTraperzoidal101Py.jpg')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.8.8의 결과와 같다. ■

이변량함수의 정적분을 구하기 위해서, MATLAB함수 integral2.m을 사용할 수 있다.

예제 8.8.10 MATLAB함수 integral2.m을 사용하는 예로서 다음 MATLAB프로그램 UseIntegral201.m을 실행해 보자.

```
% ----------------------------------------------------------------
%  Filename: UseIntegral201.m
%  Use MATLAB function Integral2.m for Double Integral
%  https://kr.mathworks.com/help/matlab/ref/integral.html#examples
% ----------------------------------------------------------------
clear all, clear all, format long
% Example 1: Integrate Triangular Region with Singularity
fun1 = @(x,y) 1./( sqrt(x + y) .* (1 + x + y).^2 )
ymax = @(x) 1 - x
q1 = integral2(fun1,0,1,0,ymax)

% Example 2: Evaluate Double Integral in Polar Coordinates
fun2 = @(x,y) 1./( sqrt(x + y) .* (1 + x + y).^2 )
polarfun = @(theta,r) fun2(r.*cos(theta),r.*sin(theta)).*r
rmax = @(theta) 1./(sin(theta) + cos(theta));
q2 = integral2(polarfun,0,pi/2,0,rmax)
```

```
17
18 % Example 3: Evaluate Double Integral of Parameterized Function
19 a = 3;  b = 5;
20 fun3 = @(x,y) a*x.^2 + b*y.^2
21 q3 = integral2(fun3,0,5,-5,0,'Method','iterated', ...
22                 'AbsTol',0,'RelTol',1e-10)
23 % End of program
24 % ------------------------------------------------------------------
```

첫 번째 예제는 다음 2중적분을 하기 위한 것이다.

$$I_1 = \int_0^1 \int_0^{1-x} \frac{1}{\sqrt{x+y}[1+x+y]^2} dydx \tag{1}$$

점 $(0,0)$는 특이점임에 유의하라. 이 적분값은 $I_1 = 0.285398175390866$이다.

두 번째 예제는 다음과 같은 극좌표함수를 적분하기 위한 것이다.

$$I_2(c) = \int_0^{\pi/2} \int_0^{1/[\sin\theta+\cos\theta]} \frac{r}{\sqrt{r\cos\theta + r\sin\theta}\,[1 + r\cos\theta + r\sin\theta]^2} drd\theta \tag{2}$$

이 적분값 I_2는 0.285398163347462이다.

세 번째 예제는 다음과 같은 매개모수들을 사용한 함수를 적분하기 위한 것이다.

$$I_3 = \int_{-5}^0 \int_0^5 \left[ax^2 + by^2\right] dxdy, \quad a = 3, \;\; b = 5 \tag{3}$$

이 적분값 I_3는 1666.666666666666이다. ■

예제 8.8.11 Python을 사용해서 예제 8.8.10을 다시 다루기 위해서, 다음 Python프로그램 UseIntegrate201.Py를 실행해 보자.

```
1  """
2  %  Filename: UseIntegrate201.Py
3  %  Double Integral
4  % Programmed by CBS
5  """
6
7  import numpy as np
8  import scipy.integrate as integrate
9
10 # Example 1: Integrate Triangular Region with Singularity
11 fun1 = lambda x, y: 1/( np.sqrt(x + y)*(1 + x + y)**2)
12 q1 = integrate.dblquad(fun1,0,1, lambda x:0, lambda x:1-x)
13 print(q1)
14
15 # Example 2: Evaluate Double Integral in Polar Coordinates
16 fun2 = lambda x, y: 1/( np.sqrt(x + y)*(1 + x + y)**2 )
```

```
fun2p = lambda r, theta: fun2(r*np.cos(theta),r*np.sin(theta))*r
q2 = integrate.dblquad(fun2p, 0,np.pi/2, lambda theta:0, \
                       lambda theta:1/(np.sin(theta) + np.cos(theta)) )
print(q2)
def bounds_theta(): return [0, np.pi/2]
def bounds_r(theta): return [0, 1/(np.sin(theta) + np.cos(theta))]
q2b = integrate.nquad(fun2p, [bounds_r,bounds_theta])
print(q2b)

# Example 3: Evaluate Double Integral of Parameterized Function
a = 3;  b = 5;
fun3 = lambda x,y: a*x**2 + b*y**2
q3 = integrate.dblquad(fun3,0,5, lambda x:-5, lambda x:0)
print(q3)

# End of Program
```

이 Python프로그램을 실행한 결과는 예제 8.8.10의 결과와 같다. ■

삼변량함수의 정적분을 구하기 위해서, MATLAB함수 integral3.m을 사용할 수 있다.

예제 8.8.12 MATLAB함수 integral3.m을 사용하는 예로서 다음 MATLAB프로그램 UseIntegral301.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: UseIntegral301.m
%  Use MATLAB function Integral3.m for Triple Integral
%  https://kr.mathworks.com/help/matlab/ref/integral.html#examples
% ------------------------------------------------------------------
clear all, clear all, format long
% Example 1: Triple Integral with Finite Limits
fun1 = @(x,y,z) y.*sin(x)+z.*cos(x)
q1 = integral3(fun1,0,pi,0,1,-1,1)

% Example 2: Integral Over the Unit Sphere in Cartesian Coordinates
fun2 = @(x,y,z) x.*cos(y) + x.^2.*cos(z)
xmin = -1;
xmax = 1;
ymin = @(x) -sqrt(1 - x.^2);
ymax = @(x) sqrt(1 - x.^2);
zmin = @(x,y)-sqrt(1 - x.^2 - y.^2);
zmax = @(x,y) sqrt(1 - x.^2 - y.^2);
q2 = integral3(fun2,xmin,xmax,ymin,ymax,zmin,zmax,'Method','tiled')

% Example 3: Evaluate Improper Triple Integral
a = 2;
fun3 = @(x,y,z) 10./(x.^2 + y.^2 + z.^2 + a);
q31 = integral3(fun3,-Inf,0,-100,0,-100,0)
  % Specify accuracy to approximately 9 significant digits.
q32 = integral3(fun3,-Inf,0,-100,0,-100,0,'AbsTol', 0,'RelTol',1e-9)
% End of program
% ------------------------------------------------------------------
```

첫 번째 예제는 다음 3중적분을 하기 위한 것이다.

$$I_1 = \int_{-1}^{1}\int_{0}^{1}\int_{0}^{\pi} [y\sin x + z\cos x]\, dxdydz \tag{1}$$

이 적분값 I_1은 2.000000000000000이다.

두 번째 예제는 다음 3중적분을 하기 위한 것이다.

$$I_2 = \iiint\limits_{x^2+y^2+z^2\le 1} \left[x\cos y + x^2\cos z\right] dxdydz \tag{2}$$

이 적분값 I_2은 0.779555454656150이다.

세 번째 예제는 다음과 같은 매개모수를 사용한 함수를 적분하기 위한 것이다.

$$I_3 = \int_{-\infty}^{0}\int_{-100}^{0}\int_{-100}^{0} \frac{10}{x^2+y^2+z^2+a} dxdydz, \quad a = 2 \tag{3}$$

이 적분값 I_3는 2734.244598320928이다. 유효숫자 9자리로 계산한 적분값은 2734.244599944285이다. ■

예제 8.8.13 Python을 사용해서 예제 8.8.12을 다시 다루기 위해서, 다음 Python프로그램 UseIntegrate301.Py를 실행해 보자.

```
"""
% Filename: UseIntegrate301.Py
% Triple Integral
% Programmed by CBS
"""

import numpy as np
import scipy.integrate as integrate

# Example 1: Triple Integral with Finite Limits
fun1 = lambda z,y,x: y*np.sin(x)+z*np.cos(x)
q1 = integrate.tplquad(fun1, 0, np.pi, lambda x:0, lambda x:1,
                       lambda x,y:-1, lambda x,y:1)
print(q1)

# Example 2: Integral Over the Unit Sphere in Cartesian Coordinates
fun2 = lambda z,y,x: x*np.cos(y) + x**2*np.cos(z)
q2 = integrate.tplquad(fun2,-1,1, lambda x:-np.sqrt(1 - x**2),
     lambda x:np.sqrt(1 - x**2), lambda x,y:-np.sqrt(1 - x**2 - y**2),
     lambda x,y:np.sqrt(1 - x**2 - y**2))
print(q2)

# Example 3: Evaluate Improper Triple Integral
a = 2;
```

```
fun3 = lambda z,y,x: 10/(x**2 + y**2 + z**2 + a);
q3 = integrate.tplquad(fun3, -100, 0, lambda x:-100, lambda x:0,
                       lambda x,y:-np.inf, lambda x,y:0)
print(q3)
q3a = integrate.tplquad(fun3, -100, 0, lambda x:-100, lambda x:0,
                       lambda x,y:-np.inf, lambda x,y:0,
                       epsabs=0, epsrel=1e-9)
print(q3a)

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.8.12의 결과와 같다. ■

제8.9절 미분방정식의 수치해

8.9.1 Euler법

다음 초기값문제를 생각해보자.

$$y' = f(t, y), \quad y(a) = A \tag{8.9.1}$$

구간 $[a, b]$의 분할 Π를 다음과 같이 정의하자.

$$\Pi \doteq \{a = t_0 < t_1 < \cdots < t_{N-1} < t_N = b\} \tag{8.9.2}$$

다음 식들이 성립한다.

$$t_n = t_0 + \frac{b-a}{N}h, \quad (n = 0, 1, \cdots, N-1, N) \tag{8.9.3}$$

초기값문제 (8.9.1)의 근사해 $y_n \approx y(t_n)$을 분할 Π 상에서 $y(t_0)$부터 $y(t_n)$까지 차례로 계산하자.

다음 식을 만족하는 p, A_j와 α_j가 존재한다고 하자.

$$\int_{t_n}^{t_n+h} f\left(x, y(x)\right) dx = h\sum_{j=1}^{s} A_j f\left(t_{n,j}, y(t_{n,j})\right) + O\left(h^{p+1}\right) \tag{8.9.4}$$

$$t_{n,j} \doteq t_n + \alpha_j h, \quad 0 < \alpha_j < 1 \tag{8.9.5}$$

중간값 $t_{n,j}$를 모르기때문에 식 (8.9.4)이 큰 도움이 되지 않는 것 같다. 따라서, 식 (8.9.4)를 약간 변형한 근사식들을 살펴보자.

사다리꼴공식(traperzoidal rule)의 첫 번째 형태는 다음과 같다.

$$\int_{t_n}^{t_n+h} f\left(x, y(x)\right) dx = hf\left(t_n, y(t_n)\right) + O\left(h^2\right) = hf\left(t_n, y_n\right) + O\left(h^2\right) \tag{8.9.6}$$

이에 해당하는 전향Euler법에 의한 전향미분공식(forward differentiation formula; FDF)은 다음과 같다.

$$y_{n+1} = y_n + hf\left(t_n, y_n\right) \tag{8.9.7}$$

사다리꼴공식의 두 번째 형태는 다음과 같다.

$$\int_{t_n}^{t_n+h} f\left(x, y(x)\right) dx = hf\left(t_{n+1}, y(t_{n+1})\right) + O\left(h^2\right) = hf\left(t_{n+1}, y_{n+1}\right) + O\left(h^2\right) \tag{8.9.8}$$

이에 해당하는 후향Euler법에 의한 후향미분공식(backward differentiation formula; BDF)은 다음과 같다.

$$y_{n+1} = y_n + hf\left(t_{n+1}, y_{n+1}\right) \tag{8.9.9}$$

식 (8.9.9)의 양변에 모두 y_{n+1}항이 들어가 있으므로, 이 후향미분공식은 음Euler법(implicit Euler method)이다.

사다리꼴공식의 세 번째 형태는 다음과 같다.

$$\begin{aligned}\int_{t_n}^{t_n+h} f\left(x, y(x)\right) dx &= h\frac{f\left(t_n, y(t_n)\right) + f\left(t_{n+1}, y(t_{n+1})\right)}{2} + O\left(h^2\right) \\ &= h\frac{f\left(t_n, y_n\right) + f\left(t_{n+1}, y_{n+1}\right)}{2} + O\left(h^2\right)\end{aligned} \tag{8.9.10}$$

이에 해당하는 중심Euler법에 의한 중심미분공식(central differentiation formula; CDF)은 다음과 같다.

$$y_{n+1} = y_n + \frac{h}{2}\left[f\left(t_n, y_n\right) + f\left(t_{n+1}, y_{n+1}\right)\right] \tag{8.9.11}$$

식 (8.9.11)의 양변에 모두 y_{n+1}항이 들어가 있으므로, 이 중심미분공식 역시 음Euler법이다.

예제 8.9.1 다음 초기값문제를 살펴보자.

$$\dot{y} = xy^2 + y, \quad y(-3) = \frac{1}{4} \tag{1}$$

Euler법을 사용해서 이 초기값문제를 풀기 위해서, 다음 MATLAB프로그램 ODE_Euler101.m을 살펴보자.

```
% ------------------------------------------------------------------
% Filename ODE_Euler101.m
% Euler's method
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary ODE_Euler101.txt
f = @(t,y) t*y^2+y;                          % function f
% True solution
yTrue = dsolve('Dy=t*y^2+y','y(0)=1','t')  % True solution
t0 = -3;           % Initial time
tN = 0.8;          % Final time
N = 38;            % Number of time steps
t = linspace(t0,tN,N+1);
yValue = eval(vectorize(yTrue));
% Euler method
h = (tN-t0)/N;           % Time step
yn(1) = yValue(1);       % Initial value
for nn=1:N
    t(nn+1)=t(nn)+h;
    yn(nn+1)=yn(nn)+h*f(t(nn),yn(nn));
end
% Plotting
plot(t,yValue,'g-',t,yn,'k:','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[t0 tN])
xlabel('\bf t','fontsize',12)
ylabel('y','fontsize',12','rotation',0)
legend('\bf Euler','\bf True value','location','NW')
saveas(gcf,'ODE_Euler101.jpg')
% End of program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 ODE_Euler101.m을 실행하면, 이 초기값문제의 닫힌해가 다음과 같음을 알 수 있다.

$$y = \frac{1}{1-t} \tag{2}$$

시간구간 $[-3, 0.8]$을 38개 소구간들로 나누어 Euler법을 적용한다. 이 수치해를 그린 그래프가 그림 8.9.1에 실려있다. 그림 8.9.1에서 녹색 실선은 Euler법에 의한 수치해를 나타내고, 흑색 점선은 해석해를 나타낸다. 그림 8.9.1에서 알 수 있듯이, 시점 t가 증가할수록 해석해와 수치해의 차이가 점점 커진다. ■

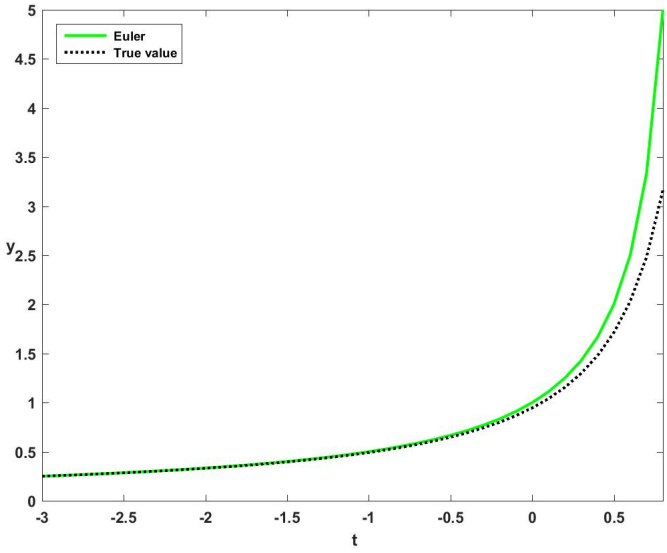


그림 8.9.1. Euler법

예제 8.9.2 Python을 사용해서 예제 8.9.1을 다시 다루기 위해서, 다음 Python프로그램 ODE_Euler101.Py를 실행해 보자.

```
"""
% Filename ODE_Euler101.Py
% Euler's method
% Programmed by CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from sympy import Function, dsolve
from sympy.abc import t
from IPython.display import *
init_printing(use_latex=True)

# True solution
var('C1')
f = Function('f')(t)
DE = Eq(f.diff(t)-t*f**2-f)
print('DE:',DE)
yTrueG = dsolve(DE)
print('yTrueG =',yTrueG)
display(yTrueG)
yTrueP = yTrueG.subs(C1,0)
print('yTrueP =',yTrueP)
display(yTrueP)
y = Lambda(t,yTrueP.rhs)

# Euler method
g = lambda td, yd: td*yd**2 + yd
t0 = -3;           # Initial time
tN = 0.8;          # Final time
N = 38;            # Number of time steps
h = (tN-t0)/N;           # Time step
```

```
ts = np.zeros(N+1)
yn = np.zeros(N+1)
ts[0] = -3; yn[0] = 1/4;       # Initial values
for nn in range(0,N):
    ts[nn+1] = ts[nn] + h;
    yn[nn+1] = yn[nn] + h*g(ts[nn],yn[nn]);

# Plotting
fig = plt.figure()
x = np.linspace(t0,tN,101)
plt.grid(True)
plt.xlabel('Time $t$',fontsize=12)
plt.ylabel('$f(t)$',fontsize=16)
plt.plot(x,[y(t) for t in x],color='#008000')
plt.plot(ts,yn,'r*', lw=2.0)
plt.show()
fig.savefig('ODE_Euler101Py.jpg')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.9.1의 결과와 같다. ■

8.9.2 Runge-Kutta법

시점 t_n에서 함수값 y_n으로부터 시점 t_{n+1}에서 함수값 y_{n+1}을 유도하기로 하자. 다음과 같은 새로운 변수들을 정의하자.

$$y_{n,1} = y_n \quad f_{n,1} = f\left(t_n, y_{n,1}\right) \tag{8.9.12}$$

각 $j(=2,3,\cdots,s)$에 대해서 다음과 같은 중간값들을 계산하자.

$$y_{n,j} = y_n + h_n \sum_{k=1}^{j-1} \beta_{j,k}\, f_{n,k} \tag{8.9.13}$$

$$f_{n,j} = f\left(t_n + \alpha_j h_n, y_{n,j}\right) \tag{8.9.14}$$

이 값들을 사용해서 시점 t_{n+1}에서 함수값 y_{n+1}을 다음과 같이 계산한다.

$$y_{n+1} = y_n + h_n \sum_{j=1}^{s} \gamma_j\, f_{n,j} \tag{8.9.15}$$

식 (8.9.12)∼식 (8.9.15)가 Runge-Kutta공식을 구성한다. 식 (8.9.4)에서 사용된 오차의 차수가 p이기 위한 Runge-Kutta공식의 계수들에 대한 조건방정식은 다음과 같다.

$$\sum_{j=1}^{s} \gamma_j \alpha_j^{k-1} = \frac{1}{k}, \quad (k = 1, 2 \cdots, p) \tag{8.9.16}$$

식 (8.9.16)은 필요조건이지만 충분조건은 아니다.

예제 8.9.3 다음 초기값문제를 살펴보자.

$$y' = f(t, y), \quad y(t_0) = y_0 \tag{1}$$

이 초기값문제에 제3차 Runge-Kutta법(the fourth order Runge-Kutta method: RK3)을 적용하기로 하자. 각 $n(= 0, 1, 2, \cdots)$에 대해서, RK3의 제$[n+1]$번째 단계는 다음과 같이 구성된다.

(a) $k_1 = hf\,(t_n, y_n)$

(b) $k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$

(c) $k_3 = hf\,(t_n + h, y_n - k_1 + 2k_2)$

(e) $y_{n+1} = y_n + \frac{1}{6}\,[k_1 + 4k_2 + k_3]$

(f) 이렇게 계산된 y_{n+1}을 $y(t_{n+1})$의 근사값으로 취한다.

제3차 Runge-Kutta법인 RK3를 실행하는 예로서, 다음 MATLAB프로그램 RungeKutta3A.m을 살펴보자.

```
% ------------------------------------------------------------------
% Filename: RungeKutta3A.m
% Runge-Kutta of order 3; Example 1
% Programmed by CBS
% ------------------------------------------------------------------
function RungeKutta3A
clear all, close all
f = @(t,y) y - t^2 +1;
ytrue = @(t) t^2 +2*t +1 - 0.5*exp(t)
h = 0.02;         % step size
t = 0;            % Initial time
tFinal = 2.0      % Final time
ys = 0.5;         % Initial value y(0)
fprintf('Step 0: t = %6.4f, y = %13.8f\n', t, ys);
for nn=1:round(tFinal/h)
    k1 = h*f(t,ys);
```

```
    k2 = h*f(t+h/2, ys+k1/2);
    k3 = h*f(t+h, ys-k1+2*k2);
    ys = ys + (k1+4*k2+k3)/6;
    t = t + h;
    tt(nn) = t;
    yn(nn) = ys;
    yValue(nn) = ytrue(t);
    error(nn) = yValue(nn) - yn(nn);
    fprintf('Step %d: t=%6.4f,y=%13.8f\n,error = %13.10f\n',...
             nn, t, ys, error(nn));
end
errorMax = max(abs(error));
display1 = ['The max error is ', num2str(errorMax)];
disp(display1)
% Plotting
plot(tt,yValue,'g-',tt,yn,'k:','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0 tFinal])
xlabel('\bf t','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
legend('\bf Runge Kutta Order 3','\bf True value','location','SE')
axis([0 2 0 6 ])
saveas(gcf,'RungeKutta3A.jpg')
end
% ------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta3A.m은 다음 초기값문제를 제3차 Runge-Kutta법으로 푸는 것이다.

$$y' = y - t^2 + 1, \quad y(0) = -.5 \tag{2}$$

이 초기값문제의 닫힌해는 다음과 같다.

$$y = t^2 + 2t + 1 - \frac{1}{2}e^t \tag{3}$$

이 MATLAB프로그램 RungeKutta3A.m을 실행하면, 시간구간 $[0, 2]$를 100개 소구간들로 나누어 Runge-Kutta법을 적용한다. 그 결과 $y(2)$의 근사값은 5.30547016이고, 시간구간 $[0, 2]$에서 최대오차는 1.793×10^{-6}임을 알 수 있다. 이 수치해와 해석해를 그린 그래프가 그림 8.9.2에 실려있다. ■

예제 8.9.4 Python을 사용해서 예제 8.9.3을 다시 다루기 위해서, 다음 Python프로그램 RungeKutta3A.Py를 실행해 보자.

```
"""
# Filename: RungeKutta3A.Py
```

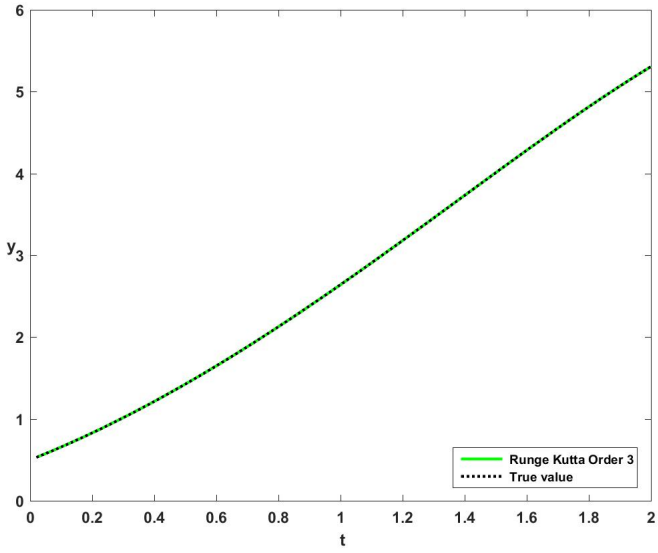


그림 8.9.2. 제3차 Runge-Kutta법

```
# Runge-Kutta of order 3; Example 1
# Programmed by CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from sympy import Function, dsolve
from sympy.abc import t
from IPython.display import *
init_printing(use_latex=True)

f = lambda t,y: y - t**2 +1;
ytrue = lambda t: t**2 +2*t +1 - 0.5*np.exp(t)
h = 0.02;         # step size
t = 0;            # Initial time
tFinal = 2.0      # Final time
ys = 0.5;         # Initial value y(0)
print('Step 0', t, ys);
N = round(tFinal/h);
tt = np.zeros(N);  yn = np.zeros(N);
yValue = np.zeros(N);   error = np.zeros(N);
for nn in range(0,N):
    k1 = h*f(t,ys);
    k2 = h*f(t+h/2, ys+k1/2);
    k3 = h*f(t+h, ys-k1+2*k2);
    ys = ys + (k1+4*k2+k3)/6;
    t = t + h;
    tt[nn] = t;
    yn[nn] = ys;
    yValue[nn] = ytrue(t);
    error[nn]= yValue[nn] - yn[nn];
    print(nn, t, ys, error[nn]);
errorMax = np.max(np.abs(error));
print('The max error is ', errorMax)

# Plotting
fig = plt.figure()
```

```
plt.plot(tt,yValue,'g-',lw=2.0, label='Runge Kutta Order 3')
plt.plot(tt,yn,'k:',lw=2.0, label='True value')
plt.xlabel('Time $t$',fontsize=12)
plt.ylabel('$y$',fontsize=16)
plt.legend(loc='lower right')
plt.axis([0, 2, 0, 6])
plt.show()
fig.savefig('RungeKutta3APy.jpg')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.9.3의 결과와 같다. ■

예제 8.9.5 다음 초기값문제를 살펴보자.

$$y' = f(t, y), \quad y(t_0) = y_0 \tag{1}$$

이 초기값문제에 제4차 Runge-Kutta법 (the fourth order Runge-Kutta method: RK4)을 적용하기로 하자. 각 $n(= 0, 1, 2, \cdots)$에 대해서, RK4의 제$[n+1]$번째 단계는 다음과 같이 구성된다.

(a) $k_1 = hf\,(t_n, y_n)$

(b) $k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$

(c) $k_3 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$

(d) $k_4 = hf\,(t_n + h, y_n + k_3)$

(e) $y_{n+1} = y_n + \frac{1}{6}\,[k_1 + 2k_2 + 2k_3 + k_4]$

(f) 이렇게 계산된 y_{n+1}을 $y(t_{n+1})$의 근사값으로 취한다.

제4차 Runge-Kutta법인 RK4를 실행하는 예로서, 다음 MATLAB프로그램 RungeKutta4A.m을 살펴보자.

```
% ------------------------------------------------------------------
% Filename: RungeKutta4A.m
% Runge-Kutta of order 4; Example 1
% Programmed by CBS
% ------------------------------------------------------------------
function RungeKutta4A
clear all, close all
f = @(t,y) y - t^2 +1;
ytrue = @(t) t^2 +2*t +1 - 0.5*exp(t)
```

```
h = 0.02;          % step size
t = 0;             % Initial time
tFinal = 2.0       % Final time
ys = 0.5;          % Initial value y(0)
fprintf('Step 0: t = %6.4f, y = %13.8f\n', t, ys);
for nn=1:round(tFinal/h)
    k1 = h*f(t,ys);
    k2 = h*f(t+h/2, ys+k1/2);
    k3 = h*f(t+h/2, ys+k2/2);
    k4 = h*f(t+h, ys+k3);
    ys = ys + (k1+2*k2+2*k3+k4)/6;
    t = t + h;
    tt(nn) = t;
    yn(nn) = ys;
    yValue(nn) = ytrue(t);
    error(nn) = yValue(nn) - yn(nn);
    fprintf('Step %d: t=%6.4f,y=%13.8f\n,error = %13.10f\n',...
             nn, t, ys, error(nn));
end
errorMax = max(abs(error));
display1 = ['The max error is ', num2str(errorMax)];
disp(display1)
% Plotting
plot(tt,yValue,'g-',tt,yn,'k:','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0 tFinal])
xlabel('\bf t','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
legend('\bf Runge Kutta Order 4','\bf True value','location','SE')
axis([0 2 0 6 ])
saveas(gcf,'RungeKutta4A.jpg')
end
% ------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta4A.m은 다음 초기값문제를 제4차 Runge-Kutta법으로 푸는 것이다.

$$y' = y - t^2 + 1, \quad y(0) = -\frac{1}{2} \tag{2}$$

이 초기값문제의 닫힌해는 다음과 같다.

$$y = t^2 + 2t + 1 - \frac{1}{2}e^t \tag{3}$$

이 MATLAB프로그램 RungeKutta4A.m을 실행하면, 시간구간 $[0, 2]$를 100개 소구간들로 나누어 Runge-Kutta법을 적용한다. 그 결과 $y(2)$의 근사값은 5.30547194이고, 시간구간 $[0, 2]$에서 최대오차는 1.1395×10^{-8}임을 알 수 있다. 이 수치해와 해석해를 그린 그래프가 그림 8.9.3에 실려있다. ■

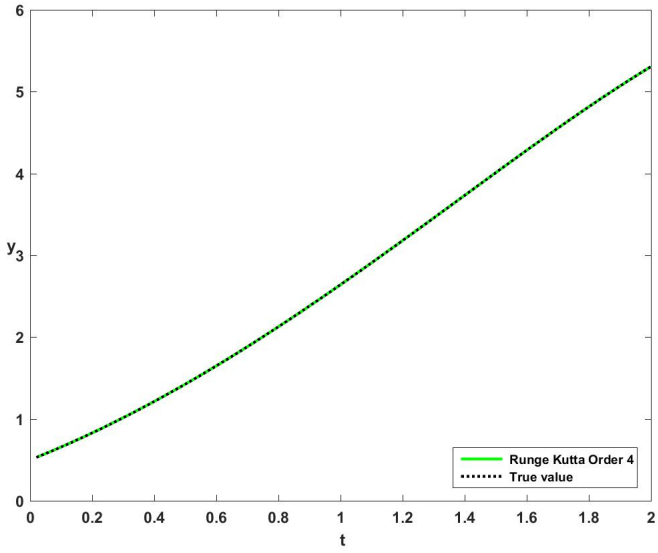


그림 8.9.3. 제4차 Runge-Kutta법

예제 8.9.6 Python을 사용해서 예제 8.9.5을 다시 다루기 위해서, 다음 Python프로그램 RungeKutta4A.Py를 실행해 보자.

```
"""
# Filename: RungeKutta4A.Py
# Runge-Kutta of order 4; Example 1
# Programmed by CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from sympy import Function, dsolve
from sympy.abc import t
from IPython.display import *
init_printing(use_latex=True)

f = lambda t,y: y - t**2 +1;
ytrue = lambda t: t**2 +2*t +1 - 0.5*np.exp(t)
h = 0.02;         # step size
t = 0;            # Initial time
tFinal = 2.0      # Final time
ys = 0.5;         # Initial value y(0)
print('Step 0', t, ys);
N = round(tFinal/h);
tt = np.zeros(N);  yn = np.zeros(N);
yValue = np.zeros(N);   error = np.zeros(N);
for nn in range(0,N):
    k1 = h*f(t,ys);
    k2 = h*f(t+h/2, ys+k1/2);
    k3 = h*f(t+h/2, ys+k2/2);
    k4 = h*f(t+h, ys+k3);
    ys = ys + (k1+2*k2+2*k3+k4)/6;
    t = t + h;
    tt[nn] = t;
    yn[nn] = ys;
    yValue[nn] = ytrue(t);
```

```
    error[nn]= yValue[nn] - yn[nn];
    print(nn, t, ys, error[nn]);
errorMax = np.max(np.abs(error));
print('The max error is ', errorMax)

# Plotting
fig = plt.figure()
plt.plot(tt,yValue,'g-',lw=2.0, label='Runge Kutta Order 4')
plt.plot(tt,yn,'k:',lw=2.0, label='True value')
plt.xlabel('Time $t$',fontsize=12)
plt.ylabel('$y$',fontsize=16)
plt.legend(loc='lower right')
plt.axis([0, 2, 0, 6])
plt.show()
fig.savefig('RungeKutta4APy.jpg')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.9.5의 결과와 같다. ■

예제 8.9.7 다음 초기값문제를 살펴보자.

$$y' = f(t, y), \quad y(t_0) = y_0 \tag{1}$$

이 초기값문제에 제5차 Runge-Kutta 법 (the fifth order Runge-Kutta method: RK5)을 적용하기로 하자. 각 $n(= 0, 1, 2, \cdots)$에 대해서, RK5의 제$[n+1]$번째 단계는 다음과 같이 구성된다.

(a) $k_1 = hf\left(t_n, y_n\right)$

(b) $k_2 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$

(c) $k_3 = hf\left(t_n + \frac{h}{4}, y_n + \frac{3k_1}{16} + \frac{k_2}{16}\right)$

(d) $k_4 = hf\left(t_n + \frac{h}{2}, y_n + \frac{k_3}{2}\right)$

(e) $k_5 = hf\left(t_n + \frac{3h}{4}, y_n - \frac{3k_2}{16} + \frac{3k_3}{8} + \frac{9k_4}{16}\right)$

(f) $k_6 = hf\left(t_n + h, y_n + \frac{k_1}{7} + \frac{4k_2}{7} + \frac{6k_3}{7} - \frac{12k_4}{7} + \frac{8k_5}{7}\right)$

(g) $y_{n+1} = y_n + \frac{1}{90}\left[7k_1 + 32k_3 + 12k_4 + 32k_5 + 7k_6\right]$

(h) 이렇게 계산된 y_{n+1}을 $y(t_{n+1})$의 근사값으로 취한다.

제5차 Runge-Kutta법인 RK5를 실행하는 예로서, 다음 MATLAB프로그램 RungeKutta5A.m을 살펴보자.

```
% ------------------------------------------------------------------
% Filename: RungeKutta5A.m
% Runge-Kutta of order 5; Example 1
% Programmed by CBS
% ------------------------------------------------------------------
function RungeKutta5A
clear all, close all
f = @(t,y) y - t^2 +1;
ytrue = @(t) t^2 +2*t +1 - 0.5*exp(t)
h = 0.02;          % step size
t = 0;             % Initial time
tFinal = 2.0       % Final time
ys = 0.5;          % Initial value y(0)
fprintf('Step 0: t = %6.4f, y = %13.8f\n', t, ys);
for nn=1:round(tFinal/h)
    k1 = h*f(t,ys);
    k2 = h*f(t+h/2, ys+k1/2);
    k3 = h*f(t+h/4, ys+(3*k1+k2)/16);
    k4 = h*f(t+h/2, ys+k3/2);
    k5 = h*f(t+3*h/4, ys+(-3*k2+6*k3+9*k4)/16);
    k6 = h*f(t+h, ys+(k1+4*k2+6*k3-12*k4+8*k5)/7);
    ys = ys + (7*k1+32*k3+12*k4+32*k5+7*k6)/90;
    t = t + h;
    tt(nn) = t;
    yn(nn) = ys;
    yValue(nn) = ytrue(t);
    error(nn) = yValue(nn) - yn(nn);
    fprintf('Step %d: t=%6.4f,y=%13.8f\n,error = %13.10f\n',...
             nn, t, ys, error(nn));
end
errorMax = max(abs(error));
display1 = ['The max error is ', num2str(errorMax)];
disp(display1)
% Plotting
plot(tt,yValue,'g-',tt,yn,'k:','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0 tFinal])
xlabel('\bf t','fontsize',12)
ylabel('y','fontsize',12','rotation',0)
legend('\bf Runge Kutta Order 5','\bf True value','location','SE')
axis([0 2 0 6 ])
saveas(gcf,'RungeKutta5A.jpg')
end
% ------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta5A.m은 다음 초기값문제를 제5차 Runge-Kutta법으로 푸는 것이다.

$$y' = y - t^2 + 1, \quad y(0) = -\frac{1}{2} \tag{2}$$

이 초기값문제의 닫힌해는 다음과 같다.

$$y = t^2 + 2t + 1 - \frac{1}{2}e^t \tag{3}$$

이 MATLAB프로그램 RungeKutta5A.m을 실행하면, 시간구간 $[0, 2]$를 100개 소구간들로 나누어 Runge-Kutta법을 적용한다. 그 결과 $y(2)$의 근사값은 5.30547195이고, 시간구간 $[0, 2]$에서 최대오차는 7.0717×10^{-12}임을 알 수 있다. 이 수치해와 해석해를 그린 그래프가 그림 8.9.4에 실려있다. ■

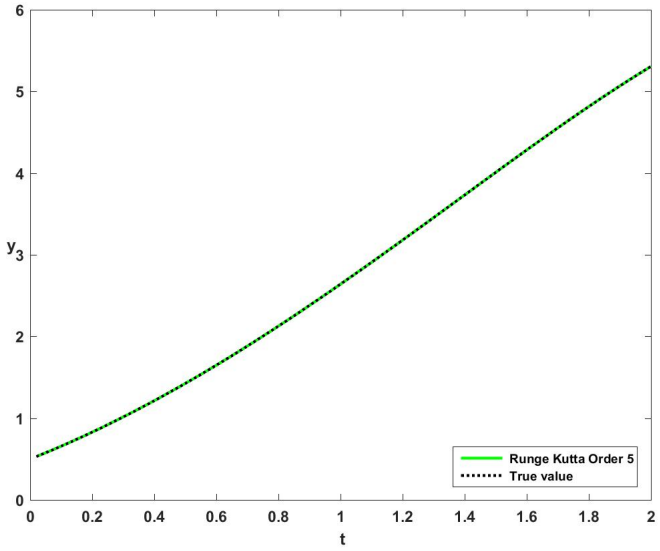


그림 8.9.4. 제5차 Runge-Kutta법

예제 8.9.8 Python을 사용해서 예제 8.9.7을 다시 다루기 위해서, 다음 Python프로그램 RungeKutta5A.Py를 실행해 보자.

```
"""
# Filename: RungeKutta5A.Py
# Runge-Kutta of order 5; Example 1
# Programmed by CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from sympy import Function, dsolve
from sympy.abc import t
from IPython.display import *
init_printing(use_latex=True)

f = lambda t,y: y - t**2 +1;
```

```
ytrue = lambda t: t**2 +2*t +1 - 0.5*np.exp(t)
h = 0.02;          # step size
t = 0;             # Initial time
tFinal = 2.0       # Final time
ys = 0.5;          # Initial value y(0)
print('Step 0', t, ys);
N = round(tFinal/h);
tt = np.zeros(N);  yn = np.zeros(N);
yValue = np.zeros(N);   error = np.zeros(N);
for nn in range(0,N):
    k1 = h*f(t,ys);
    k2 = h*f(t+h/2, ys+k1/2);
    k3 = h*f(t+h/4, ys+(3*k1+k2)/16);
    k4 = h*f(t+h/2, ys+k3/2);
    k5 = h*f(t+3*h/4, ys+(-3*k2+6*k3+9*k4)/16);
    k6 = h*f(t+h, ys+(k1+4*k2+6*k3-12*k4+8*k5)/7);
    ys = ys + (7*k1+32*k3+12*k4+32*k5+7*k6)/90;
    t = t + h;
    tt[nn] = t;
    yn[nn] = ys;
    yValue[nn] = ytrue(t);
    error[nn]= yValue[nn] - yn[nn];
    print(nn, t, ys, error[nn]);
errorMax = np.max(np.abs(error));
print('The max error is ', errorMax)

# Plotting
fig = plt.figure()
plt.plot(tt,yValue,'g-',lw=2.0, label='Runge Kutta Order 5')
plt.plot(tt,yn,'k:',lw=2.0, label='True value')
plt.xlabel('Time $t$',fontsize=12)
plt.ylabel('$y$',fontsize=16)
plt.legend(loc='lower right')
plt.axis([0, 2, 0, 6])
plt.show()
fig.savefig('RungeKutta5APy.jpg')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.9.7의 결과와 같다. ■

예제 8.9.9 다음 초기값문제를 살펴보자.

$$y' = f(t,y), \quad y(t_0) = y_0 \tag{1}$$

이 초기값문제에 Runge-Kutta-Fehlberg법인 RK45를 적용하기로 하자. 각 $n(=0,1,2,\cdots)$에 대해서, RK45의 제$[n+1]$번째 단계는 다음과 같이 구성된다.

(a) $k_1 = hf\left(t_n, y_n\right)$

(b) $k_2 = hf\left(t_n + \frac{h}{4}, y_n + \frac{k_1}{4}\right)$

(c) $k_3 = hf\left(t_n + \frac{3h}{8}, y_n + \frac{3k_1}{32} + \frac{9k_2}{32}\right)$

(d) $k_4 = hf\left(t_n + \frac{12h}{13}, y_n + \frac{1932k_1}{2197} - \frac{7200k_2}{2197} + \frac{7296k_3}{2197}\right)$

(e) $k_5 = hf\left(t_n + h, y_n + \frac{439k_1}{216} - 8k_2 + \frac{3680k_3}{513} - \frac{845k_4}{4104}\right)$

(f) $k_6 = hf\left(t_n + \frac{h}{2}, y_n - \frac{8k_1}{27} + 2k_2 - \frac{3544k_3}{2565} + \frac{1859k_4}{4104} - \frac{11k_5}{40}\right)$

(g) $y_{n+1} = y_n + \frac{25k_1}{216} + \frac{1408k_3}{2565} + \frac{2197k_4}{4104} - \frac{k_5}{5}$

(h) $\tilde{y}_{n+1} = y_n + \frac{16k_1}{135} + \frac{6656k_3}{12825} + \frac{28561k_4}{56430} - \frac{9k_5}{50} + \frac{2k_6}{55}$

(i) 만약 주어진 작은 양수 ϵ에 대해서 $R = |\tilde{y}_{n+1} - y_{n+1}|/h \le \epsilon$이면, 현재 단계 해로서 y_{n+1}을 사용하고 다음 단계의 스텝크기를 δh로 한다. 그렇지 않으면, 스텝크기를 δh로 해서 현재 단계를 다시 계산한다. 여기서 $\delta = 0.84[\epsilon/R]^{1/4}$이다.

(j) 이렇게 계산된 y_{n+1}을 $y(t_{n+1})$의 근사값으로 취한다.

Runge-Kutta법인 RK45를 실행하는 예로서, 다음 MATLAB프로그램 RungeKuttaFehlberg45A.m을 살펴보자.

```
% ------------------------------------------------------------------------
% Filename: RungeKuttaFehlberg45A.m
% Runge-Kutta-Felberg 45; Example 1
% Programmed by CBS
% ------------------------------------------------------------------------
function RungeKutta45A
clear all, close all
f = @(t,y) y - t^2 +1;
ytrue = @(t) t^2 +2*t +1 - 0.5*exp(t)
epsil = 1.0e-10;
h = 0.02;          % step size
t = 0;             % Initial time
ys = 0.5;          % Initial value y(0)
nn = 0
tFinal = 2.0       % Final time
fprintf('Step 0: t = %6.4f, y = %13.8f\n', t, ys);
while t < 2
    h = min(h,2-t);
    k1 = h*f(t,ys);
    k2 = h*f(t+h/4, ys+k1/4);
    k3 = h*f(t+3*h/8, ys+3*k1/32+9*k2/32);
    k4 = h*f(t+12*h/13, ys+1932*k1/2197-7200*k2/2197+7296*k3/2197);
    k5 = h*f(t+h, ys+439*k1/216-8*k2+3680*k3/513-845*k4/4104);
    k6 = h*f(t+h/2, ys-8*k1/27+2*k2-3544*k3/2565+1859*k4/4104-11*k5/40);
    ys1 = ys + 25*k1/216+1408*k3/2565+2197*k4/4104-k5/5;
    ys2 = ys + 16*k1/135+6656*k3/12825+28561*k4/56430-9*k5/50+2*k6/55;
    R = abs(ys1-ys2)/h;
    delta = 0.84*(epsil/R)^(1/4);
    if R <= epsil
       t = t+h;
```

```
        ys = ys1;
        nn = nn+1;
        h = delta*h;
    else
        h = delta*h;
    end
    if nn >= 1
        tt(nn) = t;
        yn(nn) = ys;
        yValue(nn) = ytrue(t);
        error(nn) = yValue(nn) - yn(nn);
        fprintf('Step %d: t=%6.4f,y=%13.8f\n,error = %13.10f\n',...
                nn, t, ys, error(nn));
    end
end

% Plotting
plot(tt,yValue,'g-',tt,yn,'k:','LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[0 2])
xlabel('\bf t','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
legend('\bf Runge-Kutta-Fehlberg','\bf True value','location','SE')
axis([0 2 0 6 ])
saveas(gcf,'RungeKuttaFehlberg45A.jpg')
end
% ---------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKuttaFehlberg45A.m은 다음 초기값문제를 제4차 Runge-Kutta법으로 푸는 것이다.

$$y' = y - t^2 + 1, \quad y(0) = -.5 \tag{2}$$

이 초기값문제의 닫힌해는 다음과 같다.

$$y = t^2 + 2t + 1 - \frac{1}{2}e^t \tag{3}$$

이 MATLAB프로그램 RungeKuttaFehlberg45A.m을 실행하면, 시간구간 $[0, 2]$를 100개 소구간들로 나누어 Runge-Kutta법을 적용한다. 그 결과 $y(2)$의 근사값은 5.30547194이고, 시간구간 $[0, 2]$에서 최대오차는 1.1395×10^{-8}임을 알 수 있다. 이 수치해와 해석해를 그린 그래프가 그림 8.9.5에 실려있다. ■

예제 8.9.10 Python을 사용해서 예제 8.9.9을 다시 다루기 위해서, 다음 Python프로그램 RungeKuttaFehlberg45A.Py를 실행해 보자.

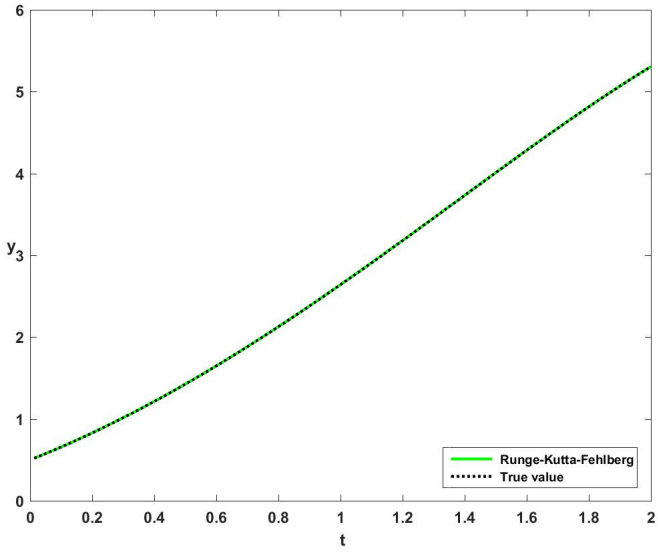


그림 8.9.5. Runge-Kutta법 RK45

```
"""
# Filename: RungeKuttaFehlberg45A.Py
# Runge-Kutta-Felberg 45; Example 1
# Programmed by CBS
"""

import numpy as np
import matplotlib.pyplot as plt

f = lambda t,y: y - t**2 +1;
ytrue = lambda t: t**2 +2*t +1 - 0.5*np.exp(t)
# epsil = 1.0e-5;
epsi = 1.0e-5

tInitial = 0;    # Initial time
yInitial = 0.5;        # Initial value y(0)
tFinal = 2.0     # Final time
print('Step 0', tInitial, yInitial);
h = 0.02;        # step size
N = int(round(tFinal/h*2.0));
tt = 0*np.ones(N);
yTrueValue = yInitial*np.ones(N);        # True value
yValue = yInitial*np.ones(N);    # RKF value
error = np.zeros(N);
t = tInitial
ys = yInitial
for nn in range(0,N-1):
    h = 0.02;        # step size
    for jj in range(0,100):
        h = min(h,2-t);
        k1 = h*f(t,ys);
        k2 = h*f(t+h/4, ys+k1/4);
        k3 = h*f(t+3*h/8, ys+3*k1/32+9*k2/32);
        k4 = h*f(t+12*h/13, ys+1932*k1/2197-7200*k2/2197+7296*k3/2197);
        k5 = h*f(t+h, ys+439*k1/216-8*k2+3680*k3/513-845*k4/4104);
        k6 = h*f(t+h/2, ys-8*k1/27+2*k2-3544*k3/2565+1859*k4/4104-11*k5/40);
```

```
        ys1 = ys + 25*k1/216+1408*k3/2565+2197*k4/4104-k5/5;
        ys2 = ys + 16*k1/135+6656*k3/12825+28561*k4/56430-9*k5/50+2*k6/55;
        R = abs(ys1-ys2)/max(h,1e-11)
        print('R',R)
        if np.abs(R) <= epsi:
            nn1 = nn+1
            t = t+h
            ys = ys1
            tt[nn1] = t;
            yTrueValue[nn1] = ytrue(t);
            yValue[nn1] = ys
            error[nn1] = yValue[nn1] - yTrueValue[nn1];
            print(nn1, tt[nn1], yValue[nn1], yTrueValue[nn1], error[nn1])
            break
        else:
            delta = 0.84*(epsi/R)**(1/4);
            h = delta*h;
            t = t+h;
            ys = ys1;
errorMax = np.max(np.abs(error));
print('The max error is ', errorMax)

# Plotting
fig = plt.figure()
plt.plot(tt,yValue,'g-',lw=1.0, label='Runge-Kutta-Fehlberg')
plt.plot(tt,yTrueValue,'k:',lw=2.0, label='True value')
plt.xlabel('Time $t$',fontsize=12)
plt.ylabel('$y$',fontsize=16)
plt.legend(loc='lower right')
plt.axis([0, 2, 0, 6])
plt.show()
fig.savefig('RungeKuttaFehlberg45APy.png')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.9.9의 결과와 같다. ■

8.9.3 Euler법과 Runge-Kutta법

이 소절에서 주로 다루게 될 미분방정식들은 다음과 같다.

첫째, Van der Pol의 진동방정식(oscillator equation)은 다음과 같다.

$$\ddot{x} - \mu[1 - x^2]\dot{x} + x = 0 \tag{8.9.17}$$

이 진동방정식을 다음과 같은 연립미분방정식으로 쓸 수 있다.

$$\dot{x} = y \tag{8.9.18}$$

$$\dot{y} = \mu[1 - x^2]y - x \tag{8.9.19}$$

둘째, 카오스(chaotic behavior)를 발생시키는 시스템으로 유명한 Lorenz시스템은 다음과 같다 [37].

$$\dot{x} = -\sigma x + \sigma y \tag{8.9.20}$$

$$\dot{y} = \rho x - y - xz \tag{8.9.21}$$

$$\dot{z} = -\beta z + xy \tag{8.9.22}$$

셋째, 포식자(predator)와 먹잇감(prey)의 관계를 나타내는 Lotka-Volterra모형은 다음과 같은 가정을 바탕으로 한다.

(a) 포식자에게는 항상 충분한 먹잇감이 있다.

(b) 포식자의 음식량은 먹잇감의 수에 전적으로 의존한다.

(c) 객체수(population)의 변화율(rate of change)은 객체수 그 자체에 비례한다.

(d) 먹잇감의 수와 포식자의 수가 변화하는 동안 다른 환경은 변화하지 않는다.

(e) 포식자의 식욕은 끝이 없다.

이러한 가정 하에서 먹잇감 수 x와 포식자 수 y를 다음과 같은 연립미분방정식으로 나타낼 수 있다.

$$\dot{x} = x\,[\alpha - \beta y] \tag{8.9.23}$$

$$\dot{y} = y\,[-\gamma + \delta x] \tag{8.9.24}$$

여기서 α, β, γ 와 δ는 양(positive)인 상수들이다. Lotka [38] 는 화학반응을 나타내기 위해서 이 모형을 사용했고 Volterra [64]는 1차세계대전 중 Adriatic해에서 어획량이 급증한 이유를 설명하기 위해서 이 모형을 사용했다. 경제학에서도 오랫동안 이 모형을 사용해왔다 [24]. 이 모형은 로지스틱모형과 밀접한 관계가 있다.

예제 8.9.11 Euler법을 사용해서 Van der Pol의 진동방정식을 풀기 위해서, 다음 MATLAB프로그램 Euler_VDP101.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename: Euler_VDP101.m
% Euler Method for van der Pol oscillator Problem with mu=1
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
diary EulerVDPa.txt

% Euler Method
timee = 0; tend = 349;
x = [0 1/4]';
k1 = zeros(2,1); k2 = k1;
u = zeros(1,tend+1); v = u;
h = 0.1;
while timee <= tend
   k1(1) = h*(x(1)*(1-x(2)^2)-x(2));
   k1(2) = h*x(1);
   y = x + k1;
   k2(1) = h*(y(1)*(1-y(2)^2)-y(2));
   k2(2) = h*y(1);
   x = x+(k1+k2)/2;
   u(timee+1) = x(2);
   v(timee+1) = x(1);
   timee = timee+1;
end

% Plotting
tspan = h*(1:timee);
plot(tspan,u,'r-',tspan,v,'k-.','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('x','y','location','SW')
xlabel('\bf Time','fontsize',12)
ylabel('\bf Solution','fontsize',12)
saveas(gcf,'Euler_VDP101','jpg')
diary off
% End oh Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 Euler_VDP101.m은 다음 초기값문제를 Euler법으로 풀기 위한 것이다.

$$\dot{x} = y, \quad \dot{y} = [1 - x^2]y - x, \quad x(0) = \frac{1}{4}, \quad y(0) = 0 \tag{1}$$

이 MATLAB프로그램 Euler_VDP101.m을 실행하면, 시간구간 $[0, 35]$를 350개 소구간들로 나누어 Euler법을 적용한다. 이 수치해를 그린 그래프가 그림 8.9.6에 실려있다. 그림 8.9.6에서 적색 실선이 x를 나타내고, 흑색 반점선은 y를 나타낸다. ■

예제 8.9.12 Python을 사용해서 예제 8.9.11을 다시 다루기 위해서, 다음 Python프로그램 Euler_VDP101.Py를 실행해 보자.

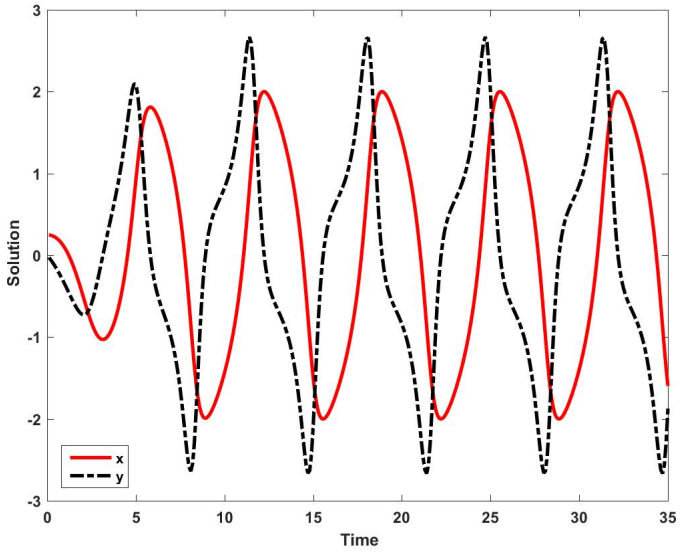


그림 8.9.6. Van der Pol의 진동방정식과 Euler법

```
"""
% Filename: Euler_VDP101.Py
% Euler Method for van der Pol oscillator Problem with mu=1
% Programmed by CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from IPython.display import *
# init_printing(use_latex=True)

timee = 0; tend = 350;
x = [0, 1/4]
k1 = np.zeros(2); k2 = np.zeros(2);
u = np.zeros(tend);  v = np.zeros(tend);
h = 0.1;
while (timee < tend):
    k1[0] = h*(x[0]*(1-x[1]**2)-x[1]);
    k1[1] = h*x[0];
    y = x + k1;
    k2[0] = h*(y[0]*(1-y[1]**2)-y[1]);
    k2[1] = h*y[0];
    x = x+(k1+k2)/2;
    u[timee] = x[1];
    v[timee] = x[0];
    timee = timee+1;
tspan = h*np.arange(0,timee);

# Plotting
fig = plt.figure()
plt.plot(tspan,u,'r-',lw=2.5,label='x')
plt.plot(tspan,v,'k-.',lw=2.5,label='y')
plt.xlabel('Time $t$',fontsize=12)
plt.ylabel('Solution',fontsize=16)
plt.legend(loc='lower left')
plt.axis([0, 35, -3, 3])
```

```
plt.show()
fig.savefig('Euler_VDP101Py.jpg')

# End of program
```

이 Python프로그램을 실행한 결과는 예제 8.9.11의 결과와 같다. ■

예제 8.9.13 Euler법을 사용해서 Lorenz시스템을 풀기 위해서, 다음 Python프로그램 Euler_Lorenz101.Py를 실행해 보자.

```
"""
% Filename: Euler_Lorenz101.Py
% Plot of the Lorenz Attractor based on Edward Lorenz's 1963
% "Deterministic Nonperiodic Flow" publication.
% https://matplotlib.org/examples/mplot3d/lorenz_attractor.html
"""

import numpy as np
import matplotlib.pyplot as plt
# from mpl_toolkits.mplot3d import Axes3D

def lorenz(x, y, z, sigma=10, rho=28, beta=2.667):
    x_dot = sigma*(y - x)
    y_dot = rho*x - y - x*z
    z_dot = x*y - beta*z
    return x_dot, y_dot, z_dot

dt = 0.01
stepCnt = 10000

# Need one more for the initial values
xs = np.empty((stepCnt + 1,))
ys = np.empty((stepCnt + 1,))
zs = np.empty((stepCnt + 1,))

# Setting initial values
xs[0], ys[0], zs[0] = (-7.0, 7.0, 25.0)

# Stepping through "time".
for i in range(stepCnt):
    # Derivatives of the X, Y, Z state
    x_dot, y_dot, z_dot = lorenz(xs[i], ys[i], zs[i])
    xs[i + 1] = xs[i] + (x_dot * dt)
    ys[i + 1] = ys[i] + (y_dot * dt)
    zs[i + 1] = zs[i] + (z_dot * dt)

# Plotting
fig = plt.figure()
ax = fig.gca(projection='3d')
ax.plot(xs, ys, zs, 'r', lw=0.5)
ax.set_xlabel("X Axis")
ax.set_ylabel("Y Axis")
ax.set_zlabel("Z Axis")
ax.set_title("Lorenz Attractor")
ax.set_xlim(-20,20)
ax.set_ylim(-50,50)
```

```
ax.set_zlim(0,60)
plt.show()
fig.savefig('Euler_Lorenz101Py.jpg')

# End of program
```

이 Python프로그램 Euler_Lorenz101.Py는 다음 초기값문제를 Euler법으로 풀기 위한 것이다.

$$\dot{x} = -10x + 10y \tag{1}$$

$$\dot{y} = 28x - y - xz \tag{2}$$

$$\dot{z} = -2.667z + xy \tag{3}$$

이 Python프로그램 Euler_Lorenz101.Py를 실행하면, 그림 8.9.7이 그려진다. ■

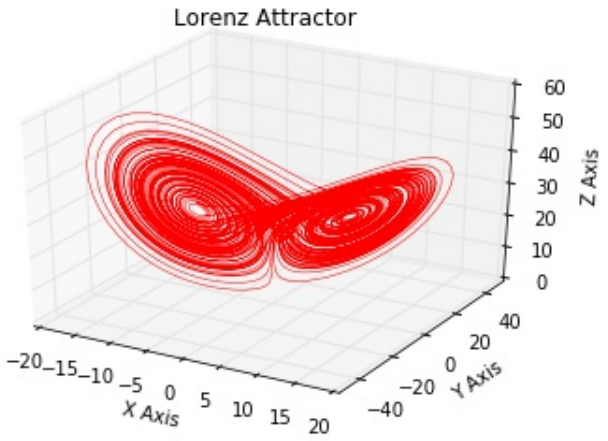


그림 8.9.7. Lorenz시스템과 Euler법

예제 8.9.14 제4차 Runge-Kutta법을 사용해서 Van der Pol의 진동방정식을 풀기 위해서, 다음 MATLAB프로그램 RungeKutta4VDPa.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename RungeKutta4VDPa.m
% Runge-Kutta Method RK4
% for van der Pol oscillator Problem with mu=1
% Programmed by CBS
% ------------------------------------------------------------------
clear, close all
```

```
diary RungeKutta4VDPa.txt

% Runge-Kutta Method
timee = 0; tend = 349;
x = [0 1/4]';
k1 = zeros(2,1); k2 = k1; k3 = k1; k4 = k1;
u = zeros(1,tend+1); v = u;
h = 0.1;
while timee <= tend
   k1(1) = h*(x(1)*(1-x(2)^2)-x(2));
   k1(2) = h*x(1);
   y = x + k1/2;
   k2(1) = h*(y(1)*(1-y(2)^2)-y(2));
   k2(2) = h*y(1);
   y = x + k2/2;
   k3(1) = h*(y(1)*(1-y(2)^2)-y(2));
   k3(2) = h*y(1);
   y = x + k3;
   k4(1) = h*(y(1)*(1-y(2)^2)-y(2));
   k4(2) = h*y(1);
   x = x+(k1+2*k2+2*k3+k4)/6;
   u(timee+1) = x(2);
   v(timee+1) = x(1);
   timee = timee+1;
end

% Plotting
tspan = h*(1:timee);
plot(tspan,u,'r-',tspan,v,'k-.','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
legend('x','y','location','SW')
xlabel('\bf Time','fontsize',12)
ylabel('\bf Solution','fontsize',12)
saveas(gcf,'RungeKutta4VDPa','jpg')
diary off
% End of Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta4VDPa.m은 다음 초기값문제를 제4차 Runge-Kutta 법으로 풀기 위한 것이다.

$$\dot{x} = y, \quad \dot{y} = [1 - x^2]y - x, \quad x(0) = \frac{1}{4}, \quad y(0) = 0 \tag{2}$$

이 MATLAB프로그램 RungeKutta4VDPa.m을 실행하면, 시간구간 $[0, 35]$를 350개 소구간들로 나누어 제4차 Runge-Kutta법을 적용한다. 이 수치해를 그린 그래프가 그림 8.9.8에 실려있다. 그림 8.9.8에서 적색 실선이 x를 나타내고, 흑색 반점선은 y를 나타낸다. ■

MATLAB함수 ode45.m은 Runge-Kutta-Fehlberg법을 적용해서 미분방정식을 푸는 MATLAB의 ODE루틴이다. 이 ode45는 경도(stiffness)가 낮은 미분방정식에 적용하는

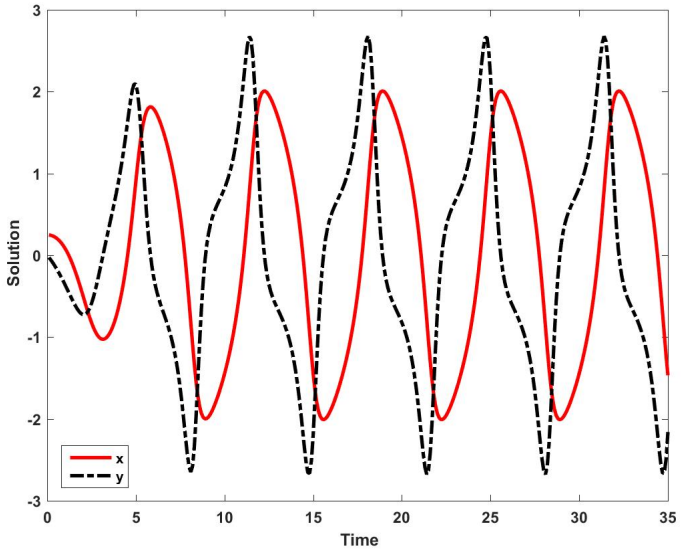


그림 8.9.8. Van der Pol의 진동방정식과 제4차 Runge-Kutta법

함수로서 정확도(order of accuracy)는 보통(medium)이다. 미분방정식에 대부분 적용되는 함수로서 독자가 문제를 풀 때 처음 시도해야하는 ODE루틴이다.

예제 8.9.15 Runge-Kutta-Fehlberg법을 사용해서 Van der Pol의 진동방정식을 풀기 위해서, 다음 MATLAB프로그램 RungeKutta45VDPa.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename: RungeKutta45VDPa.m
% Runge-Kutta-Fehlberg Method through MATLAB function ode45.m
% for van der Pol oscillator Problem with mu=1
% Presented by MathWorks
% ------------------------------------------------------------------
clear, close all
diary  RungeKutta45VDPa.txt

% Runge_Kutta method
tspan = [0, 35];
y0 = [1/4 ; 0 ];
Mu = 1;
ode = @(t,y) vanderpoldemo(t,y,Mu)
[t,y] = ode45(ode, tspan, y0);

% Plot of the solution
plot(t,y(:,1),'r-',t,y(:,2),'k-.','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold','xlim',tspan)
legend('x','y','location','SW')
xlabel('\bf Time','fontsize',12)
ylabel('\bf Solution','fontsize',12)
saveas(gcf,'RungeKutta45VDPa','jpg')
diary off
% End of Program
% ------------------------------------------------------------------
```

이 MATLAB 프로그램 RungeKutta45VDPa.m은 다음 초기값문제를 Runge-Kutta-Fehlberg법으로 풀기 위한 것이다.

$$\dot{x} = y, \quad \dot{y} = [1 - x^2]y - x, \quad x(0) = \frac{1}{4}, \quad y(0) = 0 \tag{2}$$

이 MATLAB프로그램 RungeKutta45VDPa.m을 실행하면, Runge-Kutta-Fehlberg법을 적용한다. 이 수치해를 그린 그래프가 그림 8.9.9에 실려있다. 그림 8.9.9에서 적색 실선이 x를 나타내고, 흑색 반점선은 y를 나타낸다. ■

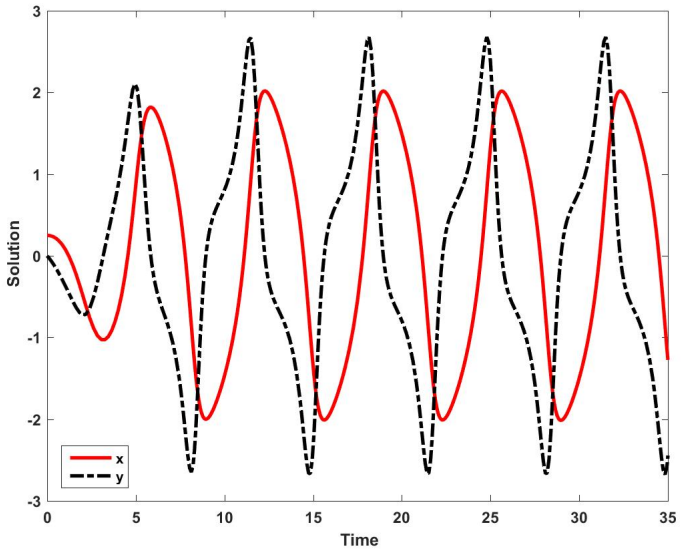


그림 8.9.9. Runge-Kutta-Fehlberg법과 Van der Pol의 진동방정식

예제 8.9.16 Runge-Kutta-Fehlberg법을 사용해서 Lorenz방정식을 풀기 위해서, 다음 MATLAB프로그램 RungeKutta45Lorenz.m을 실행해 보자.

```
% ------------------------------------------------------------------
% Filename: RungeKutta45Lorenz.m
% Runge-Kutta-Fehlberg Method through MATLAB function ode45.m
% for Lorenz system
% Programmed by CBS
% ------------------------------------------------------------------
function RungeKutta45Lorenz
clear, close all
diary  RungeKutta45Lorenz.txt
y0 = [-7 7 25 ]          % Initial values
tt = [0,30]              % Time span
sigma = 10, rho = 28, beta = 2.667  % Parameters
para = [ sigma beta rho ];
```

```
[t,y] = ode45(@lorenz,tt,y0,[],para);

% Plot of the solution
subplot(2,2,1)
plot(y(:,1),y(:,2),'r-','LineWidth',1.0)
set(gca,'fontsize',11,'fontweigh','bold')
axis equal
xlabel('x','fontsize',12)
ylabel('y','fontsize',12,'rotation',0)
subplot(2,2,2)
plot(y(:,1),y(:,3),'g-','LineWidth',1.0)
set(gca,'fontsize',11,'fontweigh','bold')
axis equal
xlabel('x','fontsize',12)
ylabel('z','fontsize',12,'rotation',0)
subplot(2,2,3)
plot(y(:,2),y(:,3),'b-','LineWidth',1.0)
set(gca,'fontsize',11,'fontweigh','bold')
axis equal
xlabel('y','fontsize',12)
ylabel('z','fontsize',12,'rotation',0)
subplot(2,2,4)
plot3(y(:,1),y(:,2),y(:,3),'k','LineWidth',1);
grid on
set(gca,'fontsize',11,'fontweigh','bold')
xlabel('\bf x','fontsize',12), ylabel('\bf y','fontsize',12)
zlabel('\bf z','fontsize',12)
saveas(gcf,'RungeKutta45Lorenz','jpg')
diary off
end
% End of Program
% --------------------------------------------------------------------
function yprime = lorenz(t,y,p);
% LORENZ: Computes the derivatives involved in Lorenz equations.
sig = p(1);
beta = p(2);
rho = p(3);
yprime = [ -sig*y(1) + sig*y(2); rho*y(1) - y(2) - y(1)*y(3); ...
           -beta*y(3) + y(1)*y(2) ];
end
% --------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta45Lorenz.m을 실행하면, 모수들이 다음과 같은 모수들을 갖는 Lorenz시스템을 Runge-Kutta-Fehlberg법으로 푼다.

$$\sigma = 10, \quad \beta = 2.667, \quad \rho = 28 \tag{4}$$

이 MATLAB프로그램 RungeKutta45Lorenz.m을 실행하면, 시간구간 $[0, 30]$에서 Runge-Kutta-Fehlberg법을 적용한다. 이렇게 구한 수치해를 그린 그래프들이 그림 8.9.10에 실려 있다. 그림 8.9.10에서 좌측상단 그래프는 x대 y를, 우측상단 그래프는 x대 z를, 좌측하단 그래프는 y대 z를, 우측하단 그래프는 x대 y대 z를 그린 것이다. ■

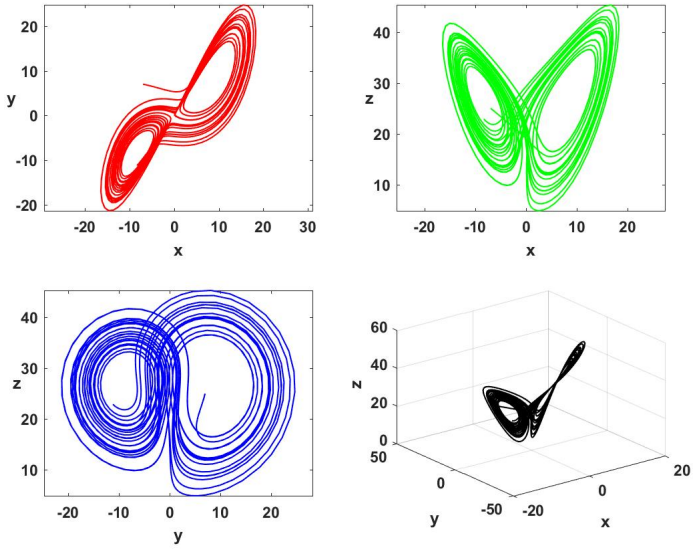


그림 8.9.10. Runge-Kutta-Fehlberg법과 Lorenz Curves

예제 8.9.17 Python패키지 scipy의 integrate모듈의 함수 odeint는 FORTRAN라이브러리 odepack의 LSODA (Ordinary Differential Equation Solver for Stiff or Non-Stiff System)를 사용해서 미분방정식을 수치적으로 푼다. 함수 odeint를 적용해서 Lorenz방정식을 풀기 위해서, 다음 Python프로그램 LSODA2Lorenz.Py을 실행해 보자.

```
"""
% Filename: LSODA2Lorenz.Py
% Solve Lorenz system using scipy.integrate.odeint
% http://ipywidgets.readthedocs.io/en/latest/examples
%              /Lorenz%20Differential%20Equations.html
"""
from ipywidgets import interact, interactive
from IPython.display import clear_output, display, HTML
import numpy as np
from scipy import integrate
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.colors import cnames
from matplotlib import animation

def solve_lorenz(N=10, angle=0.0, max_time=4.0, sigma=10.0, beta=8./3, rho
    =28.0):

    fig = plt.figure()
    ax = fig.add_axes([0, 0, 1, 1], projection='3d')
    ax.axis('off')

    # prepare the axes limits
    ax.set_xlim((-25, 25))
    ax.set_ylim((-35, 35))
    ax.set_zlim((5, 55))

    def lorenz_deriv(x_y_z, t0, sigma=sigma, beta=beta, rho=rho):
```

```
        """Compute the time-derivative of a Lorenz system."""
        x, y, z = x_y_z
        return [sigma * (y - x), x * (rho - z) - y, x * y - beta * z]

    # Choose random starting points, uniformly distributed from -15 to 15
    np.random.seed(1)
    x0 = -15 + 30 * np.random.random((N, 3))

    # Solve for the trajectories
    t = np.linspace(0, max_time, int(250*max_time))
    x_t = np.asarray([integrate.odeint(lorenz_deriv, x0i, t)
                      for x0i in x0])

    # choose a different color for each trajectory
    colors = plt.cm.viridis(np.linspace(0, 1, N))

    for i in range(N):
        x, y, z = x_t[i,:,:].T
        lines = ax.plot(x, y, z, '-', c=colors[i])
        plt.setp(lines, linewidth=2)

    ax.view_init(30, angle)
    plt.show()
    fig.savefig('LSODA2LorenzPy.jpg')
    return t, x_t

# Call the function solve_lorenz
t, x_t = solve_lorenz(angle=0, N=10)

# End of Program
```

이 Python프로그램 LSODA2Lorenz.Py를 실행하면, 모수들이 다음과 같은 모수들을 갖는 Lorenz시스템을 푼다.

$$\sigma = 10, \quad \beta = 2.667, \quad \rho = 28 \tag{4}$$

이 Python프로그램 LSODA2Lorenz.Py을 실행하면, 그림 8.9.11이 그려진다. ■

MATLAB함수 ode23.m은 미분방정식을 푸는 데 사용되는 MATLAB ODE루틴의 하나이다. 이 ode23.m은 경도(stiffness)가 낮은 미분방정식에 적용하는 함수로서 정확도(order of accuracy)는 낮다. 오차허용범위(error tolerance)가 크거나 경도가 중도적인(moderately) 미분방정식을 푸는 데 사용한다.

예제 8.9.18 Runge-Kutta법을 사용해서 Van der Pol의 진동방정식을 풀기 위해서, 다음 MATLAB프로그램 RungeKutta23VDPa.m을 살펴보자.

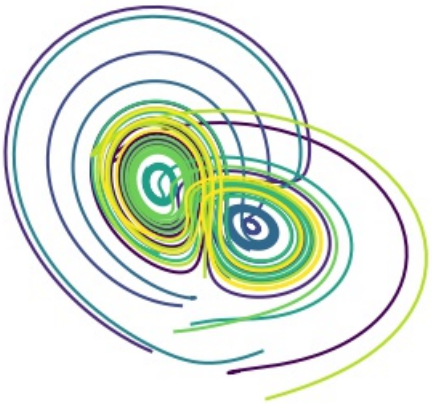


그림 8.9.11. LSODA와 Lorenz Curves

```
% ------------------------------------------------------------------
% Filename RungeKutta23VDPa.m
% Runge-Kutta Method using MATLAB function ode23.m
% for van der Pol equation
% Programmed by CBS
% ------------------------------------------------------------------
function RungeKutta23VDPa
diary RungeKutta23VDPa.txt
tspan = [0,35];
x0 = [0; 1/4];
[t,x] = ode23(@VanDerPol,tspan,x0);
% Plot of the solution
plot(t,x(:,1),'k-.',t,x(:,2),'r-.','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold','xlim',tspan)
legend('t','x','location','SW')
xlabel('\bf Time','fontsize',12)
ylabel('\bf Solution','fontsize',12)
saveas(gcf,'RungeKutta23VDPa','jpg')
diary off
end
% End of Program
% ------------------------------------------------------------------
function xdot=VanDerPol(t,x)
  xdot = zeros(2,1);
  xdot(1) = x(1)*(1-x(2)^2)-x(2);
  xdot(2) = x(1);
end
% ------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta23VDPa.m은 다음 초기값문제를 Runge-Kutta법으로 풀기 위한 것이다.

$$\dot{x} = y, \quad \dot{y} = [1 - x^2]y - x, \quad x(0) = \frac{1}{4}, \quad y(0) = 0 \tag{2}$$

이 MATLAB프로그램 RungeKutta23VDPa.m을 실행하면, 시간구간 $[0, 35]$에 Runge-Kutta법 ode23.m을 적용한다. 이렇게 구한 수치해를 그린 그래프가 그림 8.9.12에 실려있다.

그림 8.9.12에서 적색 실선이 x를 나타내고, 흑색 반점선은 y를 나타낸다. ■

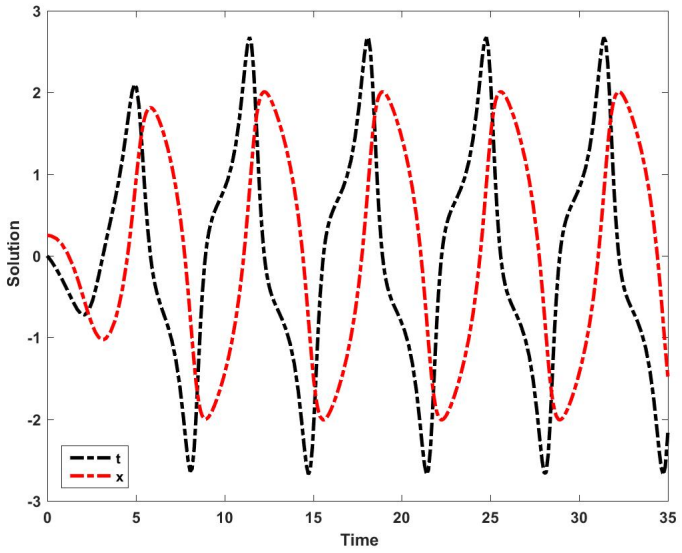


그림 8.9.12. Van der Pol의 진동방정식과 MATLAB함수 ODE23.m

예제 8.9.19 Runge-Kutta방법 RK23을 사용해서 Lotka-Volterra의 미분방정식을 푸는 방법을 살펴보기 위해서, 다음 MATLAB프로그램 RungeKutta23LotkaVolterra.m을 살펴보자.

```
% ---------------------------------------------------------------------
% Filename: RungeKutta23LotkaVolterra.m
% Solve Lotka-volterra equation using Runge-Kutta Method RK23
% Programmed by CBS
% ---------------------------------------------------------------------
function RungeKutta23LotkaVolterra
clear all, close all
diary LotkaVolterra101.txt
global alpha beta gamma delta
alpha = 1;   beta = .05;   gamma = .5;  delta = .02;
tspan = [0,39];
x0 = [30 30];
options = odeset('RelTol', 1e-6, 'NonNegative', [1 2]);
[t,x] = ode23(@LotkaVolterra,tspan,x0,options);
prey = x(:,1);
predators = x(:,2);

% Plots of the solution
subplot(2,1,1)
plot(t,prey,'b-',t,predators,'r--','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold','xlim',tspan,'ylim',[0,50])
legend('prey', 'predator','location','NW')
xlabel('\bf Time','fontsize',12)
ylabel('\bf Population','fontsize',12)
subplot(2,2,3)
```

```
plot(prey,predators,'k-','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold')
axis([0 50 0 50]), axis equal
xlabel('\bf Prey','fontsize',12)
ylabel('\bf Predator','fontsize',12)
subplot(2,2,4)
ic = linspace(1.0,5.0,11);
for r=1.0:0.4:5.0
    hold all
    x0 = [r, 1.0];
    options = odeset('RelTol', 1e-6, 'NonNegative', [1 2]);
    [t,x] = ode23(@LotkaVolterra,tspan,x0,options);
    plot(x(:,1),x(:,2),'-')
end
    xlabel('\bf Prey','fontsize',12)
    ylabel('\bf Predator','fontsize',12)
    axis([-20 230 -20 230]), axis equal
    hold off
saveas(gcf,'RungeKutta23LotkaVolterra','jpg')
diary off
end
% End of Program
% --------------------------------------------------------------------
function xdot = LotkaVolterra(t,x)
  global alpha beta gamma delta
  xdot = [0;0];
  xdot(1) = alpha * x(1) - beta * x(1) * x(2);
  xdot(2) = delta * x(1) * x(2) - gamma * x(2);
end
% --------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta23LotkaVolterra.m은 모수들이 다음과 같은 Lotka-Volterra의 미분방정식을 풀기 위한 것이다.

$$\alpha = 1\,, \quad \beta = 0.05\,, \quad \gamma = 0.5\,, \quad \delta = 0.02 \tag{3}$$

이 MATLAB프로그램 RungeKutta23LotkaVolterra.m을 실행하면, 시간구간 $[0, 39]$에서 Runge-Kutta법 ode23.m을 적용한다. 이렇게 구한 수치해를 그린 시점 t에 대한 시계열산점도가 그림 8.9.13의 상단 그래프에 실려있다. 이 상단 그래프에서 적색 실선이 포식자의 수를 나타내고, 흑색 반점선은 먹잇감의 수를 나타낸다. 또한 먹잇감 수와 포식자 수의 2차원 산점도가 그림 8.9.13의 하단좌측 그래프에 그려져 있다. 이 하단좌측 그래프에서 알 수 있듯이, 먹잇감 수와 포식자 수는 단조함수 형태가 아닌 타원 형태를 이룬다. 또한, 하단우측 그래프는 먹잇감 수의 다양한 초기값들에 대한 먹잇감과 포식자의 산점도이다. ■

예제 8.9.20 Python을 사용해서 예제 8.9.19를 다시 다루기 위해서, 다음 Python프로그램 RungeKutta23LotkaVolterra.Py를 실행해 보자.

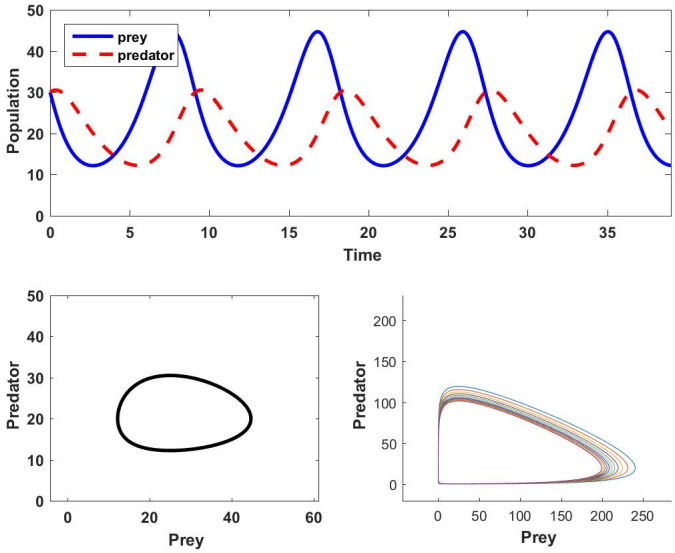


그림 8.9.13. Lotka-Volterra 방정식과 MATLAB함수 ODE23.m

```
"""
% Filename: RungeKutta23LotkaVolterra.m
% Solve Lotka-volterra equation using scipy.integrate.odeint
% Programmed by CBS
@author: CBS
"""

import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

alpha, beta, gamma, delta = 1.0, 0.05, 0.5, 0.02
def dP_dt(P, t):
    return [P[0]*(alpha - beta*P[1]), -P[1]*(gamma - delta*P[0])]

tt = np.linspace(0, 39, 391)
P0 = [30, 30]
Ps = odeint(dP_dt, P0, tt)
prey = Ps[:,0]
predators = Ps[:,1]

fig = plt.figure()
plt.subplot(2,1,1)
plt.plot(tt, prey, "b-", linewidth=2.5, label="Rabbits")
plt.plot(tt, predators, "r--", linewidth=2.5, label="Foxes")
plt.xlabel("Time")
plt.ylabel("Population")
plt.legend(loc='upper left');
plt.axis([0, 39, 0, 50])

plt.subplot(2,2,3)
plt.plot(prey, predators, "k-")
plt.xlabel("Rabbits")
plt.ylabel("Foxes")
plt.title("Rabbits vs Foxes");
plt.axis([0, 50, 0, 50])
```

```
37
38 plt.subplot(2,2,4)
39 ic = np.linspace(1.0, 5.0, 11)
40 for r in ic:
41     P0 = [r, 1.0]
42     Ps = odeint(dP_dt, P0, tt)
43     plt.plot(Ps[:,0], Ps[:,1], "-")
44 plt.xlabel("Rabbits")
45 plt.ylabel("Foxes")
46 plt.title("Rabbits vs Foxes");
47 plt.axis([-20, 230, -20, 230])
48
49 plt.show()
50 fig.savefig('RungeKutta23LotkaVolterraPy.jpg')
51
52 # End of Program
```

이 Python프로그램을 실행하면, Python패키지 scipy의 integrate모듈의 함수 odeint를 적용해서 Lotka-Volterra의 미분방정식을 푼다. 이 Python프로그램을 실행한 결과는 예제 8.9.19의 결과와 같다. ■

Runge-Kutta법을 실현하는 MATLAB함수 ode113.m은 미분방정식을 푸는 데 사용되는 MATLAB ODE루틴의 하나이다. 이 MATLAB함수 ode113.m은 경도(stiffness)가 낮은 미분방정식에 적용하는 함수로서 정확도(order of accuracy)는 낮은 것으로부터 높은 것까지 적용할 수 있다. 오차허용범위(error tolerance)가 작거나(stringent) 계산량을 많이 필요로 하는 미분방정식을 푸는 데 사용한다.

예제 8.9.21 Runge-Kutta법 ode113.m을 사용해서 Van der Pol의 진동방정식을 풀기 위해서, 다음 MATLAB프로그램 RungeKutta113VDPa.m을 살펴보자.

```
1  % --------------------------------------------------------------------
2  % Filename RungeKutta113VDPa.m
3  % Runge-Kutta Method using MATLAB function ode113.m
4  % for van der Pol equation
5  % Presented by MathWorks
6  % --------------------------------------------------------------------
7  clear, close all
8  diary RungeKutta113VDPa.txt
9  tspan = [0, 35];
10 y0 = [1/4 ; 0 ];
11 Mu = 1;
12 ode = @(t,y) vanderpoldemo(t,y,Mu);
13 [t,y] = ode113(ode, tspan, y0);
14 % Plot of the solution
15 plot(t,y(:,1),'r-',t,y(:,2),'k-.','LineWidth',2.5)
16 set(gca,'fontsize',11,'fontweigh','bold','xlim',tspan)
17 legend('x','y','location','SW')
18 xlabel('\bf Time','fontsize',12)
19 ylabel('\bf Solution','fontsize',12)
```

```
saveas(gcf,'RungeKutta113VDPa','jpg')
diary off
% End of Program
% ------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta113VDPa.m은 다음 초기값문제를 MATLAB함수 ode113.m으로 풀기 위한 것이다.

$$\dot{x} = y, \quad \dot{y} = [1 - x^2]y - x, \quad x(0) = \frac{1}{4}, \quad y(0) = 0 \tag{2}$$

이 MATLAB프로그램 RungeKutta113VDPa.m을 실행하면, 시간구간 $[0, 35]$에 Runge-Kutta법인 ode113.m을 적용한다. 이렇게 구한 수치해를 그린 그래프가 그림 8.9.14에 실려 있다. 그림 8.9.14에서 적색 실선이 x를 나타내고, 흑색 반점선은 y를 나타낸다. ■

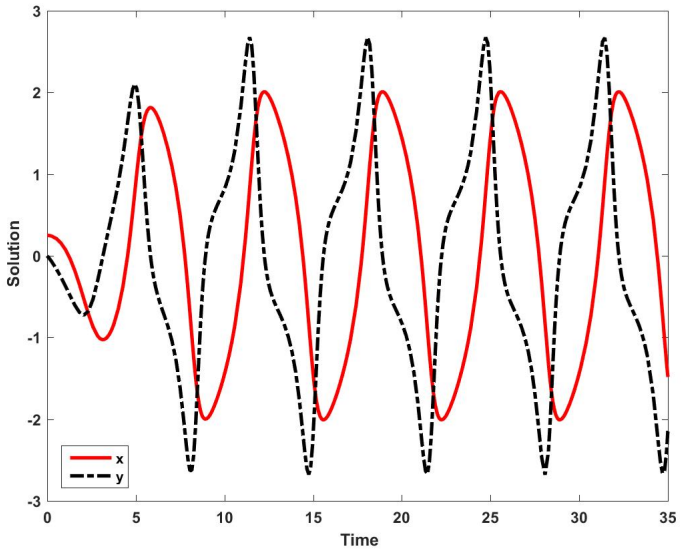


그림 8.9.14. Runge-Kutta함수 ODE113.m

Runge-Kutta법을 실현하는 MATLAB함수 ode15s.m은 미분방정식을 푸는 데 사용되는 MATLAB ODE루틴의 하나이다. 이 MATLAB함수 ode15s.m은 경도(stiffness)가 높은 미분방정식에 적용하는 함수로서 정확도(order of accuracy)는 낮은 것으로부터 중간 것까지 적용할 수 있다. 미분방정식 자체의 경도가 높아서 MATLAB함수 ode45.m을 적용할 때 많은 계산항을 필요로 하면 ode15s.m을 적용한다.

예제 8.9.22 Runge-Kutta법 ode15s.m을 사용해서 Van der Pol의 진동방정식을 풀기 위해서, 다음 MATLAB프로그램 RungeKutta15sVDPa.m을 살펴보자.

```
% ------------------------------------------------------------------
% Filename: RungeKutta15sVDPa.m
% Runge-Kutta Method using MATLAB function ode15s.m
% for van der Pol
% Presented by MathWorks
% ------------------------------------------------------------------
function RungeKutta15sVDPa
clear, close all
diary RungeKutta15sA.txt
Mu = 1000;
tspan = [0; 4000];              % Several periods
y0 = [2; 0];
options = odeset('Jacobian',@J);
[t,y] = ode15s(@f,tspan,y0,options,Mu);

% Plot of the solution
plot(t,y(:,2),'g-',t,y(:,1),'r:','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold','xlim',tspan)
legend('y','x','location','SW')
axis([tspan(1) tspan(end) -2.5 2.5]);
xlabel('\bf Time','fontsize',12)
ylabel('\bf Solution','fontsize',12)
saveas(gcf,'RungeKutta15sVDPa','jpg')
diary off
% End of Program
% ------------------------------------------------------------------
function dydt = f(t,y,mu)
dydt = [ y(2); mu*(1-y(1)^2)*y(2)-y(1) ];
return
% ------------------------------------------------------------------
function dfdy = J(t,y,mu)
dfdy = [ 0 1 ; -2*mu*y(1)*y(2)-1    mu*(1-y(1)^2) ];
return
% ------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta15sVDPa.m은 다음 초기값문제를 MATLAB함수 ode15s.m으로 풀기 위한 것이다.

$$\dot{x} = y, \quad \dot{y} = 1000[1 - x^2]y - x, \quad x(0) = 2, \quad y(0) = 0 \tag{2}$$

이 MATLAB프로그램 RungeKutta15sVDPa.m을 실행하면, 시간구간 $[0, 4000]$에 Runge-Kutta법인 ode15s.m을 적용한다. 이렇게 구한 수치해를 그린 그래프가 그림 8.9.15에 실려 있다. 그림 8.9.15에서 적색 점선이 x를 나타내고, 녹색 실선은 y를 나타낸다. ■

Runge-Kutta법을 실현하는 MATLAB함수들 ode23s.m, ode23t.m, 그리고 ode23tb.m은 미분방정식을 푸는 데 사용되는 MATLAB ODE루틴들이다. 이중 MATLAB함수 ode23s.m은 경도(stiffness)가 높은 미분방정식에 적용하는 함수로서 정확도(order of accuracy)는 낮은 것에 적용할 수 있다. 만약 미분방정식의 경도가 심해서 오차허용도(error tolerance)

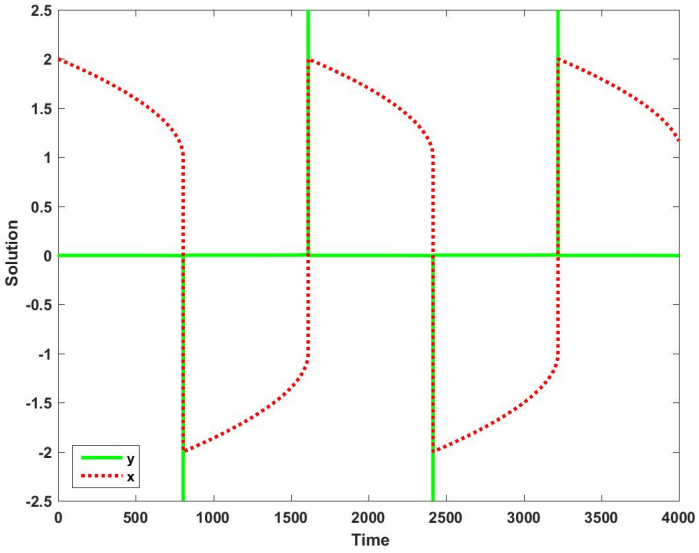


그림 8.9.15. Runge-Kutta함수 ODE15s.m

가 크면, MATLAB함수 ode23s.m을 적용한다. 연립미분방정식 $M(t,y)y' = f(t,y)$에서 계수행렬 $M(t,y)$를 매스행렬(mass matrix)라 부른다. 이 매스행렬이 함수 t에 의존하지 않는 행렬이면, MATLAB함수 ode23s.m을 적용한다. MATLAB함수 ode23t.m은 경도가 약간 높은 미분방정식에 적용하는 함수로서 정확도는 낮은 것에 적용할 수 있다. 수치적으로 크게 변화하지 않는 해를 구하고자 할 때 MATLAB함수 ode23t.m을 적용한다. MATLAB함수 ode23tb.m은 경도가 높은 미분방정식에 적용하는 함수로서 정확도는 낮은 것에 적용할 수 있다. 만약 미분방정식의 경도가 크고 오차허용도가 크면, MATLAB함수 ode23tb.m을 적용한다.

예제 8.9.23 다음 초기값문제를 살펴보자.

$$\dot{x} = 2x + y + 5z + e^{-2t} \tag{1}$$

$$\dot{y} = -3x - 2y - 8z + 2e^{-2t} - \cos 3t \tag{2}$$

$$\dot{z} = 3x + 3y + 2z + \cos 3t \tag{3}$$

$$x(0) = 1\,, \quad y(0) = -1 \quad z(0) = 0 \tag{4}$$

Runge-Kutta법 ode23s.m을 사용해서 이 연립미분방정식을 풀기 위해서, 다음 MATLAB 프로그램 RungeKutta23sA.m을 살펴보자.

```
% ------------------------------------------------------------------
% Filename: RungeKutta23sA.m
% Runge-Kutta Method using MATLAB function ode23t.m
% Programmed by CBS
% ------------------------------------------------------------------
function RungeKutta23sA
% clear, close all
diary RK23sA.txt
% Closed form solution
Eqn1 = 'Dx=2*x+y+5*z+exp(-2*t)'
Eqn2 = 'Dy=-3*x-2*y-8*z+2*exp(-2*t)-cos(3*t)'
Eqn3 = 'Dz=3*x+3*y+2*z+cos(3*t)'
Init = 'x(0)=1, y(0)=-1.0, z(0)=0.0'
Sol23 = dsolve(Eqn1,Eqn2,Eqn3 ,Init,'t')
Y11 = simplify(Sol23.x)
Y21 = simplify(Sol23.y)
Y31 = simplify(Sol23.z)
t = linspace(0,pi/2,201);
y1Val = eval(vectorize(Y11));
y2Val = eval(vectorize(Y21));
y3Val = eval(vectorize(Y31));
% Numerical solution through RK23s
init = [1.0  -1.0  0.0]
TimeSpan = [0  pi/2]
[tt,yn] = ode23s(@fsys,TimeSpan,init);

% Plot of the solution
plot(tt,yn(:,1),'r:',tt,yn(:,2),'k-.',tt,yn(:,3),'b--', ...
     'LineWidth',2.0)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[tt(1) tt(end)])
hold on
grid
legend('y_1','y_2','y_3','location','SW')
plot(t,y1Val,'g-',t,y2Val,'g-',t,y3Val,'g-','LineWidth',2.0)
plot(tt,yn(:,1),'r:',tt,yn(:,2),'k-.',tt,yn(:,3),'b--', ...
     'LineWidth',2.0)
xlabel('\bf Time','fontsize',12)
ylabel('\bf Solution','fontsize',12)
hold off
saveas(gcf,'RungeKutta23sA','jpg')
diary off
% End of Program
% ------------------------------------------------------------------
function Ftn = fsys(t,Y);
Ftn(1,1) = 2*Y(1)+Y(2)+5*Y(3)+exp(-2*t);
Ftn(2,1) = -3*Y(1)-2*Y(2)-8*Y(3)+2*exp(-2*t)-cos(3*t);
Ftn(3,1) = 3*Y(1)+3*Y(2)+2*Y(3)+cos(3*t);
return
% ------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta23sA.m을 실행하면, 시간구간 $[0, \pi/2]$에 MATLAB의 Runge-Kutta함수 ode23s.m을 적용한다. 이렇게 구한 수치해를 그린 그래프가 그림 8.9.16에 실려있다. 그림 8.9.16에서 적색 점선은 y_1을 나타내고, 흑색 반점선은 y_2를 나타내고, 청색 긴점선은 y_3를 나타낸다. ■

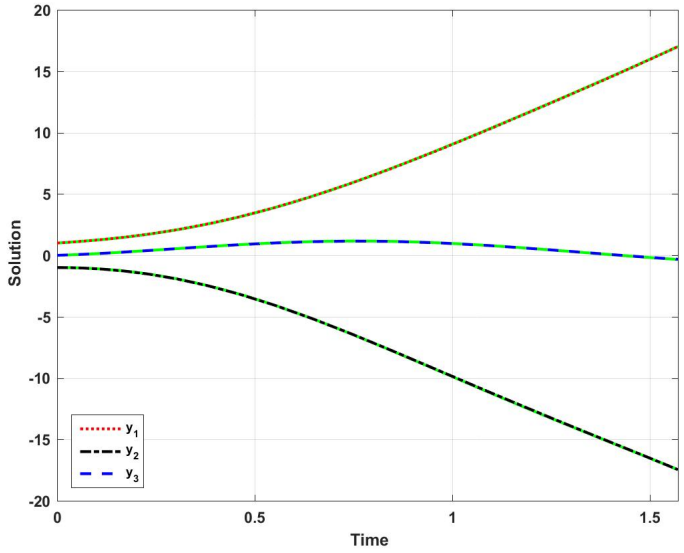


그림 8.9.16. Runge-Kutta함수 ODE23s.m

예제 8.9.24 Runge-Kutta법 ode23t를 사용해서 Van der Pol의 진동방정식을 풀기 위해서, 다음 MATLAB프로그램 RungeKutta23tVDPa.m을 살펴보자.

```
% ------------------------------------------------------------------
% Filename RungeKutta23tVDPa.m
% Runge-Kutta Method using MATLAB function ode23t.m and ode23tb.m
% Programmed by CBS
% ------------------------------------------------------------------
function RungeKutta23tVDPa
% clear, close all
diary RK23tA.txt
dydt = @(t) [y(2); 1000*(1-y(1)^2)*y(2)-y(1)]
[tt,yt] = ode23t(@vdp1000,[0 4000],[2 0]);       % ode23t
[ttb,ytb] = ode23t(@vdp1000,[0 4000],[2 0]);     % ode23tb
% Plot of the solution
plot(tt,yt(:,1),'g-',ttb,ytb(:,1),'k:','LineWidth',2.5)
set(gca,'fontsize',11,'fontweigh','bold','xlim',[tt(1) tt(end)])
legend('ode23t','ode23tb','location','SW')
xlabel('\bf Time','fontsize',12)
ylabel('\bf Solution','fontsize',12)
saveas(gcf,'RungeKutta23tVDPa','jpg')
diary off
% End of Program
% ------------------------------------------------------------------
function dydt = vdp1000(t,y)
dydt = [y(2); 1000*(1-y(1)^2)*y(2)-y(1)];
return
% ------------------------------------------------------------------
```

이 MATLAB프로그램 RungeKutta23tVDPa.m은 다음 초기값문제를 RungeKutta법으

로 풀기 위한 것이다.

$$\dot{x} = y, \quad \dot{y} = 1000[1 - x^2]y - x, \quad x(0) = 2, \quad y(0) = 0 \tag{2}$$

이 MATLAB 프로그램 RungeKutta23tVDPa.m을 실행하면, 시간구간 $[0, 4000]$에서 Runge-Kutta법 ode23t.m와 ode23tb.m을 적용한다. 이렇게 구한 수치해를 그린 그래프가 그림 8.9.17에 실려있다. 그림 8.9.17에서 녹색 실선은 Runge-Kutta법 ode23t.m에 의한 수치해이고, 흑색 점선은 Runge-Kutta법 ode23tb.m에 의한 수치해이다. ■

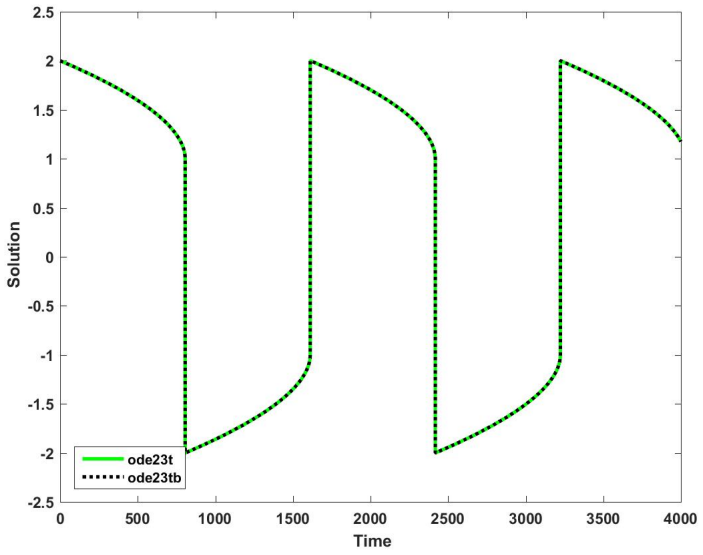


그림 8.9.17. Runge-Kutta 23t

제8.10절 예제들

예제 8.10.1 이 예제는 Ascher & Grief [11, p. 532]에서 인용한 것이다. 극좌표를 사용해서 원을 그리는 데는 많은 계산량을 필요로 한다. 좀 더 적은 계산량으로 원을 그리기 위해서, 다음과 같은 연립미분방정식을 이용할 수도 있다.

$$\dot{x} = -y, \qquad x(0) = 1 \tag{1}$$

$$\dot{y} = x, \qquad y(0) = 0 \tag{2}$$

전향차분근사, 중심차분근사 그리고 후향차분근사를 사용해서 이 연립미분방정식을 풀기 위해서, 다음 MATLAB프로그램 CircleEuler101DY.m을 실행해 보자.

```
% ------------------------------------------------------------------
%  Filename: CircleEuler101DY.m
%  Euler Methods
%  Programmed by CDY
%------------------------------------------------------------------
clear all, close all, clc
format long g
% Analytic Solution
h = 0.02;
t = 0:h:120;
xtrue = cos(t); ytrue = sin(t);
n = length(t)
A = [ 0 -1
      1 0 ]
  % Forward
Bf = eye(2) + h*A
[ Qf Lambdaf ] = eig(Bf)
DoubleCheck = Bf*Qf -Qf*Lambdaf
zF(:,1) = [ 1 0 ]';
for ii=1:n-1
    zF(:,ii+1) = zF(:,ii) + h*A*zF(:,ii);
end
% Backward
Bb = inv(eye(2) - h*A)
[ Qb Lambdab ] = eig(Bb)
DoubleCheck = Bb*Qb -Qb*Lambdab
zB(:,1) = [ 1 0 ]';
for ii=1:n-1
    zB(:,ii+1) = Bb*zB(:,ii);
end
% Midpoint
Bm = (eye(2) - h/2*A)\(eye(2) + h/2*A)
[ Qm Lambdam ] = eig(Bm)
DoubleCheck = Bm*Qm -Qm*Lambdam
zM(:,1) = [ 1 0 ]';
for ii=1:n-1
    zM(:,ii+1) = Bm*zM(:,ii);
end

% Plotting
subplot(2,2,1)
plot3(xtrue,ytrue,t,'k','LineWidth',1);
% legend('True','Midpoint')
axis square
grid on
set(gca,'fontsize',11,'fontweigh','bold')
title('\bf Analytic')
subplot(2,2,2)
plot3(zF(1,:),zF(2,:),t,'r','LineWidth',1);
axis square
grid on
set(gca,'fontsize',11,'fontweigh','bold')
title('\bf Forward')
subplot(2,2,3)
plot3(zB(1,:),zB(2,:),t,'g','LineWidth',1);
axis square
grid on
```

```
set(gca,'fontsize',11,'fontweigh','bold')
title('\bf Backward')
subplot(2,2,4)
plot3(zM(1,:),zM(2,:),t,'b','LineWidth',1);
axis square
grid on
set(gca,'fontsize',11,'fontweigh','bold')
title('\bf Midpoint')
saveas(gcf,'CircleEuler101DY.jpg')
%-------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 차분근사를 사용해서 나선구조 $\{[t, \cos t, \sin t\,] \mid t = 0 : 0.02 : 120\}$ 의 그래프들이 그림 8.10.1에 그려진다. 그림 8.10.1의 좌측상단에는 진짜 나선구조의 그래프, 우측상단은 전향차분근사에 의한 나선구조의 그래프, 좌측하단은 후향차분근사에 의한 나선구조의 그래프, 그리고 우측하단은 중심차분근사에 의한 나선구조의 그래프이다. 그림 8.10.1에서 알 수 있듯이, 전향차분근사를 사용하면 기울기벡터가 원 밖으로 향하고, 후향차분근사를 사용하면 기울기벡터가 원 안으로 향한다. ■

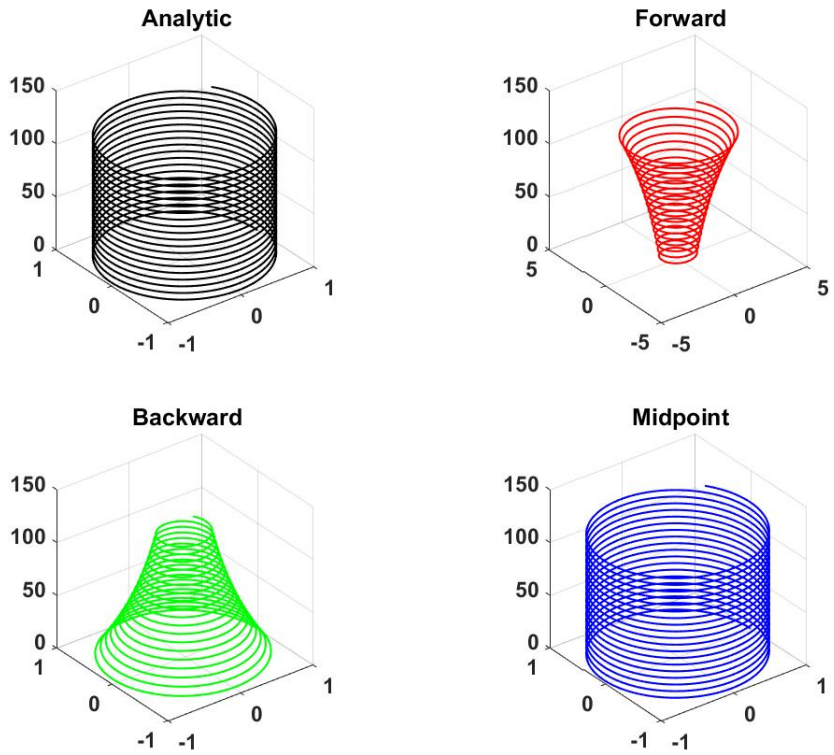


그림 8.10.1. Euler법

예제 8.10.2 Python을 사용해서 예제 8.10.1을 다시 다루기 위해서, 다음 Python프로그램 CircleEuler101DY.Py를 실행해 보자.

```
"""
%  Filename: CircleEuler101DY.m
%  Euler Methods
%  Programmed by CBS
"""

import numpy as np
from numpy import linalg as LA
import matplotlib.pyplot as plt
from matplotlib.ticker import LinearLocator, FormatStrFormatter

h = 0.02;
t = np.arange(0,120.01,h)
xtrue = np.cos(t);  ytrue = np.sin(t);
n = len(t)
A = np.array(( [ 0, -1], [1, 0 ] ));    print('A =', A)
eVal, Qa = LA.eig(A)
print('EigenValues =', eVal)
print('EigenVectors =', Qa)
Lambdaa = np.diag(eVal)
print('EigenValues Matrix =', Lambdaa)
DoubleCheck = np.matmul(A,Qa) - np.matmul(Qa,Lambdaa)
print('DoubleCheck =', DoubleCheck)

# Plotting
fig = plt.figure();        #figsize=plt.figaspect(0.5))
# True Solution
ax1 = fig.add_subplot(2,2,1, projection='3d')
ax1.plot(xtrue, ytrue, t, 'k-')
ax1.set_xlim(-1.0, 1.0)
ax1.xaxis.set_major_locator(LinearLocator(3))
ax1.xaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax1.set_ylim(-1.0, 1.0)
ax1.yaxis.set_major_locator(LinearLocator(3))
ax1.yaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax1.set_zlim(0.0, 150)
ax1.zaxis.set_major_locator(LinearLocator(4))
ax1.zaxis.set_major_formatter(FormatStrFormatter('%.0f'))

# Forward Euler Method
Bf = np.eye(2) + h*A;                 print('Bf =', Bf)
(Lambdaf0, Qf) = LA.eig(Bf)
print('F EigenValues =', Lambdaf0)
print('F EigenVectors =', Qf)
Lambdaf = np.diag(Lambdaf0)
print('F EigenValue Matrix =', Lambdaf)
DoubleCheck = np.matmul(Bf,Qf) - np.matmul(Qf,Lambdaf)
print('DoubleCheck =', DoubleCheck)
z = np.zeros((2,n))
z[0,0] = 1.0;  z[1,0] = 0.0;
for ii in range(0,n-1):
    z[:,ii+1] = np.matmul(Bf,z[:,ii])
ax2 = fig.add_subplot(2,2,2, projection='3d')
xF = z[0,:];  yF = z[1,:];
ax2.plot(xF, yF, t, 'r-')
ax2.set_xlim(-5.0, 5.0)
ax2.xaxis.set_major_locator(LinearLocator(3))
ax2.xaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax2.set_ylim(-5.0, 5.0)
ax2.yaxis.set_major_locator(LinearLocator(3))
ax2.yaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
```

```
ax2.set_zlim(0.0, 150)
ax2.zaxis.set_major_locator(LinearLocator(4))
ax2.zaxis.set_major_formatter(FormatStrFormatter('%.0f'))

# Backward Euler Method
Bb = LA.inv(np.eye(2) - h*A);         print('Bb =', Bb)
(Lambdab0, Qb) = LA.eig(Bb)
print('B EigenValues =', Lambdab0)
print('B EigenVectors =', Qb)
Lambdab = np.diag(Lambdab0)
print('B EigenValue Matrix =', Lambdab)
DoubleCheck = np.matmul(Bb,Qb) - np.matmul(Qb,Lambdab)
print('DoubleCheck =', DoubleCheck)
z = np.zeros((2,n))
z[0,0] = 1.0;  z[1,0] = 0.0;
for ii in range(0,n-1):
    z[:,ii+1] = np.matmul(LA.matrix_power(Bb,ii+1),z[:,0])
#    z[:,ii+1] = np.matmul(Bb,z[:,ii])
ax3 = fig.add_subplot(2,2,3, projection='3d')
xB = z[0,:];  yB = z[1,:];
ax3.plot(xB, yB, t, 'g-')
ax3.set_xlim(-1.0, 1.0)
ax3.xaxis.set_major_locator(LinearLocator(3))
ax3.xaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax3.set_ylim(-1.0, 1.0)
ax3.yaxis.set_major_locator(LinearLocator(3))
ax3.yaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax3.set_zlim(0.0, 150)
ax3.zaxis.set_major_locator(LinearLocator(4))
ax3.zaxis.set_major_formatter(FormatStrFormatter('%.0f'))

# Midpoint Euler Method
Bm = np.matmul( LA.inv(np.eye(2) - h/2*A),(np.eye(2) + h/2*A) );
print('Bm =', Bm)
(Lambdam0, Qm) = LA.eig(Bm)
print('M EigenValues =', Lambdam0)
print('M EigenVectors =', Qm)
Lambdam = np.diag(Lambdam0)
print('M EigenValue Matrix =', Lambdam)
DoubleCheck = np.matmul(Bm,Qm) - np.matmul(Qm,Lambdam)
print('DoubleCheck =', DoubleCheck)
z = np.zeros((2,n))
z[0,0] = 1.0;  z[1,0] = 0.0;
dumii1 = np.eye(2);
for ii in range(0,n-1):
    dumii1 = np.matmul(dumii1,Lambdam)
    dumii2 = np.matmul(np.matmul(Qm,dumii1),np.transpose(Qm))
    z[:,ii+1] = np.matmul(dumii2,z[:,0])
ax4 = fig.add_subplot(2,2,4, projection='3d')
xM = z[0,:];  yM = z[1,:];
ax4.plot(xM, yM, t, 'b-')
ax4.set_xlim(-1.0, 1.0)
ax4.xaxis.set_major_locator(LinearLocator(3))
ax4.xaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax4.set_ylim(-1.0, 1.0)
ax4.yaxis.set_major_locator(LinearLocator(3))
ax4.yaxis.set_major_formatter(FormatStrFormatter('%0.0f'))
ax4.set_zlim(0.0, 150)
ax4.zaxis.set_major_locator(LinearLocator(4))
ax4.zaxis.set_major_formatter(FormatStrFormatter('%.0f'))

```

```
plt.show()
fig.savefig('CircleEuler101DYpy.png')

# End of program
```

이 Python프로그램을 수행한 결과는 예제 8.10.1의 결과와 같다. ■

예제 8.10.3 이 예제에서는 Robertson [50, pp. 178-182]에서 제시한 다음과 같은 비선형연립방정식을 살펴보자.

$$\dot{x} = -0.05x + 9.3yz, \qquad x(0) = 1.0 \tag{1}$$

$$\dot{y} = 0.05x - 9.3yz - 2 \times 2000y^2, \qquad y(0) = 0.2 \tag{2}$$

$$\dot{z} = 2 \times 2000y^2, \qquad z(0) = 0 \tag{3}$$

Hairer & Wanner [27]는 이 문제를 ROBER라 명명했다. 식 (1)∼식 (3)을 다음과 같이 쓸 수 있다.

$$\dot{\boldsymbol{C}} = \boldsymbol{f}(\boldsymbol{C}) \tag{4}$$

여기서 $\boldsymbol{C}$와 $\boldsymbol{f}(\boldsymbol{C})$는 다음과 같다.

$$\boldsymbol{C} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} \doteq \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \quad \boldsymbol{f}(\boldsymbol{C}) = \begin{bmatrix} f_1(\boldsymbol{C}) \\ f_2(\boldsymbol{C}) \\ f_3(\boldsymbol{C}) \end{bmatrix} \doteq \begin{bmatrix} -0.05C_1 + 9.3C_2C_3 \\ 0.05C_1 - 9.3C_2C_3 - 4000C_2^2 \\ 4000C_2^2 \end{bmatrix} \tag{5}$$

Jacobian행렬은 다음과 같다.

$$J_{\boldsymbol{C}}^{\boldsymbol{f}}\Big|_{\boldsymbol{C}=\boldsymbol{C}(\boldsymbol{0})} = \frac{\partial(f_1, f_2, f_3)}{\partial(C_1, C_2, C_3)}\Big|_{\boldsymbol{C}=\boldsymbol{C}(\boldsymbol{0})} = \begin{bmatrix} -0.05 & 0 & 1.86 \\ 0.05 & -1600 & -1.86 \\ 0 & 1600 & 0 \end{bmatrix} \tag{6}$$

이 Jacobian행렬의 고유값들을 구하기 위해서, 다음 MATLAB프로그램 StiffJacobian101DY.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: StiffJacobian101DY.m
%  Stiff Ordinary Differential Equation
%  Programmed by CDY
%------------------------------------------------------------------------
clear all, close all, clc
format long g
syms c1 c2 c3
k1 = 0.05; k2 = 9.3; k3 = 2*10^3;
c10 = 0.8; c20 = 0.2; c30 = 0;
f1 = -k1*c1 + k2*c2*c3
f2 = k1*c1 - k2*c2*c3 - 2*k3*c2^2
f3 = 2*k3*c2^2
J = jacobian([f1 f2 f3], [c1 c2 c3])
J0 = subs(J,[c1 c2 c3 ],[c10 c20 c30])
J00 = double(J0)
[ eVector eValue ] = eig(J00)
% End of Program
% ------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, 이 Jacobian행렬의 고유값들이 -1598.14, -1.91 그리고 0 임을 알 수 있다. 이 고유값들에는 아주 큰 차이가 있다. 따라서, 미분방정식 (4)는 급격한 (stiff) 상미분방정식이다.

Runge-Kutta법 RK45와 RK23s를 사용해서 연립미분방정식 (4)를 풀기 위해, 다음 MATLAB프로그램 StiffODE101DY.m을 실행해 보자.

```
% ------------------------------------------------------------------------
%  Filename: StiffODE101DY.m
%  Stiff Ordinary Differential Equation
%  Programmed by CDY
%------------------------------------------------------------------------
function StiffODE101DY
clear all, close all, clc
format long g
% C(:,1) = x,   C(:,2) = y,  C(:,3) = z
% ode45
tic
[T C] = ode45(@StiffFunction101DY,[0 3000],[0.9 0.1 0]);
subplot(2,1,1)
plot(T,C(:,1),'-',T,C(:,2),'k-.',T,C(:,3),'r--','LineWidth',1.5);
legend('C1','C2','C3','location','NE')
set(gca,'fontsize',11,'fontweigh','bold')
title('\bf ode45 Solution')
elapsedTime45 = toc
% ode23s
tic
[T C] = ode23s(@StiffFunction101DY,[0 3000],[0.8 0.2 0]);
subplot(2,1,2)
plot(T,C(:,1),'-',T,C(:,2),'k-.',T,C(:,3),'r--','LineWidth',1.5);
legend('C1','C2','C3','location','NE')
set(gca,'fontsize',11,'fontweigh','bold')
title('\bf ode23s Solution')
elapsedTime23s = toc
% Saving Pictures
```

```
saveas(gcf,'StiffODE101DY.jpg')
end
% End of Program
% --------------------------------------------------------------------------
function dc = StiffFunction101DY(t,c)
dc = zeros(3,1);    % a columne vector
dc(1) = -0.05*c(1) + 9.3*c(2)*c(3);
dc(2) = 0.05*c(1) - 9.3*c(2)*c(3) - 2*2*10^3*c(2)^2;
dc(3) = 2*2*10^3*c(2)^2;
end
%--------------------------------------------------------------------------
```

이 MATLAB프로그램을 실행하면, Runge-Kutta법 RK45를 사용한 수치해를 출력한다. 앞에서 언급했듯이, MATLAB함수 ode45.m은 Runge-Kutta법을 적용하는 MATLAB함수들 중에서 가장 먼저 적용해봐야하는 ode함수이다. 이 ode함수는 경도가 높지 않은 미분방정식을 푸는 데 사용한다. 수치적 정확도는 중간 정도이다. ■

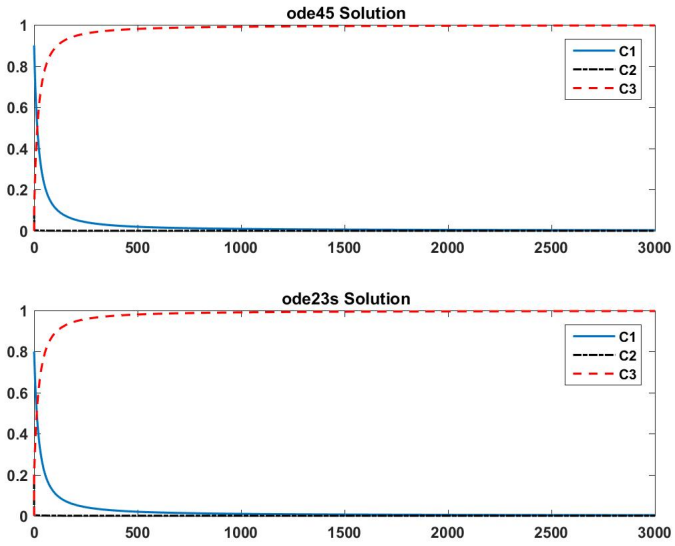


그림 8.10.2. 경도가 높은 미분방정식과 Runge-Kutta법

참고 문헌

[1] 최병선 (1995)『다변량시계열분석』, 세경사.
(본서에서는 **SAS4TSA4**로 표기함)

[2] 최병선 (1997a)『회귀분석(상)』, 세경사.
(본서에서는 **SAS4TSA3a**로 표기함)

[3] 최병선 (1997b)『회귀분석(하)』, 세경사.
(본서에서는 **SAS4TSA3b**로 표기함)

[4] 최병선 (2004)『이산형 재무모형의 수리적 배경』, 세경사.
(본서에서는 **IM&F9**으로 표기함)

[5] 최병선 (2007)『계산재무론: Computational Finance』, 세경사.
(본서에서는 **IM&F10**으로 표기함)

[6] 최병선 (2009) 금융공학 *I: Elements of Financial Engineering (IM&F*시리즈 *11)*, 세경사.
(본서에서는 **IM&F11**으로 표기함)

[7] Abramowitz, M. and Stegun, A. (1965) *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, Dover.

[8] Amman, H.M., Kendrick, D.A. and Rust, J. (Editors) (1996) *Handbook of Computational Economics, Volume 1*, North-Holland.

[9] Arfken, G.B., Weber, H.J., and Harris, F.E. (2012) *Mathematical Methods for Physicists (Seventh Edition)*, Academic Press.

[10] Armijo, L. (1966) Minimization of functions having Lipschitz continuous first partial derivatives, *Pacific J. Math.* **16**, 1–3.

[11] Ascher, U.M. and Grief, C. (2011) *A First Course in Numerical Methods*, SIAM.

[12] Boyd, S. and Vandenberghe, L. (2004) *Convex Optimization*, Cambridge University Press.

[13] Carr, P., and Madan, D.B. (2005) A note on sufficient conditions for no arbitrage. *Finance Research Letters* **2**, 125-130. doi:10.1016/j.frl.2005.04.005

[14] Cornuejols G. and Tütüncü, R. (2007) *Optimization Methods in Finance*, Cambridge University Press.

[15] Cuyt, A. and Wuytack, L. (1987) *Nonlinear Methods in Numerical Analysis*, Elsevier Science Pub. Co.

[16] de Boor, C. (1978) *A Practical Guide to Spline*, Springer-Verlag.

[17] de Boor, C. and Rice, J.R. (1968a) Least Squares Cubic Spline Approximation I — Fixed Knots, Computer Science Technical Reports. Paper 20. Available at http://docs.lib.purdue.edu/cstech/20.

[18] de Boor, C. and Rice, J.R. (1968b) Least Squares Cubic Spline Approximation, II - Variable Knots, Computer Science Technical Reports. Paper 149. Available at http://docs.lib.purdue.edu/cstech/149.

[19] Demmel, J.W. (1997) *Applied Numerical Linear Algebra*, SIAM.

[20] Eilers, P.H.C. and Marx, B.D. (1996) Flexible smoothing with B-splines and penalties, *Statistical Science* **11**, 89-102.

[21] Fengler, M.R. and Hin, L. (2015) Semi-nonparametric estimation of the call-option price surface under strike and time-to-expiry no-arbitrage constraints, *J. Econometrics* **184**, 242-261.

[22] Gander, W. and Gautschi, W. (2000) Adaptive Quadrature - Revisited, BIT, vol. 40, pp. 84-101.

[23] Golub, G.H. and Van Loan, C.F. (1996) *Matrix Computations (Third Edition)*, Johns Hopkins University Press.

[24] Goodwin, R.M. (1967) A Growth Cycle, *Socialism, Capitalism and Economic Growth*, Feinstein, C.H. (ed.), Cambridge University Press.

[25] Greenwald, B. and Stiglitz, J.E. (1986) Externalities in economies with imperfect information and incomplete markets, *Quarterly Journal of Economics* **101**, 229–264.

[26] Greville, T.N.E. (1968) Data fitting by spline functions, *MRC Technical Summary Report No 893*, Mathematics Research Center, University of Wisconsin.

[27] Hairer, E. and Wanner, G. (1996) *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems (Second Ed)*, Springer.

[28] Hanke, M. and Scherzer, O. (2001) Inverse problems light: Numerical differentiation, *The American Mathematical Monthly* **108**, pp. 512-521.

[29] Hastings, Jr., C. (1955) *Approximations for Digital Computers*, Princeton University Press.

[30] Inada, K. (1963). On a two-sector model of economic growth: Comments and a generalization, *The Review of Economic Studies* **30**, 119–127.

[31] Judd, K.L. (1998) *Numerical Methods in Economics*, The MIT Press.

[32] Karmarkar, N. (1984) A New Polynomial Time Algorithm for Linear Programming, *Combinatorica*, **4(4)**, 373–395.

[33] Karush, W. (1939) *Minima of Functions of Several Variables with Inequalities as Side Constraints*, M.Sc. Dissertation. Dept. of Mathematics, Univ. of Chicago.

[34] Körner, T.W. (1988) Fourier Analysis, Cambridge University Press.

[35] Kuhn, H.W. and Tucker, A.W. (1951) Nonlinear programming, *Proceedings of 2nd Berkeley Symposium*, University of California Press, 481–492.

[36] Lange, K. (1999) *Numerical Analysis for Statisticians*, Springer-Verlag.

[37] Lorenz, E.N. (1963) Deterministic nonperiodic flow, *Journal of the Atmospheric Sciences* **20(2)**, 130–141.

[38] Lotka, A.J. (1910) Contribution to the Theory of Periodic Reaction, *The Journal of Physical Chemistry* **14 (3)**, 271–274.

[39] Mas-Colell, A., Whinston, M. D., and Green, J. R. (1995) *Microeconomic Theory*, Oxford University Press.

[40] Mason, J.C., Rodriguez, G. and Seatzu, S. (1993) Orthogonal splines based on B-splines-with applications to least squares, smoothing and regularisation problems, *Numerical Algorithms* **5**, 25-40.

[41] Nelder, J.A. and Mead, R. (1965) A simplex method for function minimization, *Computer Journal*, **7**, pp. 308-313.

[42] Petrushev, R.V. and Popov, V.A. (1987) *Rational Approximation of Real Functions*, Cambridge University Press.

[43] Phillip, G.M. (2003) *Interpolation and Approximation by Polynomials*, Springer.

[44] Phillips, G.M. and Taylor, P.J. (1996) *Theory and Applications of Numerical Analysis (Second Edition)*, Academic Press.

[45] Powell, M.J.D. (1981) *Approximation Theory and Methods*, Cambridge University Press.

[46] Prenter, P.M. (2008) *Splines and Variational Methods*, Dover.

[47] Reinsch, C.H. (1967) Smoothing by spline functions, *Numerische Mathematik* **10**, 177-183.

[48] Richardson, L. F. (1911) The approximate arithmetical aolution by finite fifferences of physical problems involving fifferential equations, with an application to the atresses in a Masonry dam, *Philosophical Transactions of the Royal Society A* **210**, 307–357.

[49] Rivlin, T.J. (2010) *An Introduction to the Approximation of Functions*, Dover.

[50] Robertson, H.H. (1966) The solution of a set of reaction rate equations. *Numerical Analysis: An Introduction* (J. Walsh, Ed.), Academic Press.

[51] Sansone, G. (1991) *Orthogonal Functions*, Dover.

[52] Sansone, G. (2004) *Orthogonal Functions (Revised Edition)*, Dover.

[53] Schmedders, K. and Judd, K.L. (Editors) (2014) *Handbook of Computational Economics, Volume 3*, North-Holland.

[54] Schoenberg, I.J. (1973) *Cardinal Spline Interpolation*, SIAM.

[55] Schoenberg, I.J. (1981) *Spline Functions: Basic Theory (Third Edition)*, Wiley.

[56] Stoer, J. and Bulirsch, R. (1980) *Introduction to Numerical Analysis*, Springer-Verlag.

[57] Stoer, J. and Bulirch, R. (1987) *Introduction to Numerical Analysis (Second Edition)*, Springer-Verlag.

[58] Strang, G. (1987) Karmarkar's algorithm and its place in applied mathematics, *The Mathematical Intelligencer*, **9(2)**, 4–10.

[59] Sydsæter, K., Hammond, P., Seierstad, A., and Strøm, A. (2008) *Further Mathematics for Economic Analysis, (Second Edition)*, Financial Times Press.

[60] Tesfatsion, L. and Judd, K.L. (Editors) (2006) *Handbook of Computational Economics, Volume 2*, North-Holland.

[61] Todd, J. (1963) *Introduction to the Constructive Theory of Functions*, Academic Press.

[62] Uzawa, H. (1963) On a two-Sector model of economic growth II, *The Review of Economic Studies* **30**, 105–118.

[63] Venkataraman, P. (2009) *Applied Optimization with MATLAB Programming (Second Edition)*, Wiley.

[64] Volterra, V. (1926) Variazioni e fluttuazioni del numero d'individui in specie animali conviventi, *Mem. Acad. Lincei Roma* **2**, 31–113.

[65] Wegert, E. (2012) *Visual Complex Functions: An Introduction with Phase Portraits*, Birkhäuser

[66] Weierstrass, K. (1885) Über die analytische Darstellbarkeit sogenannter willkürlicher Functionen ¨ einer reellen Veränderlichen, *Verl. d. Kgl. Akad. d. Wiss. Berlin* **2**, 633–639. Also, refer to *http://people.math.sc.edu/schep/weierstrass.pdf*.

[67] Weisstein, E.W. (1998) *CRC Concise Encyclopedia of Mathematics (Second Edition)*, Chapman & Hall/CRC.

[68] Young, D.M. and Gregory, R.D. (1972) *A Survey of Numerical Mathematics, (Vol. I, II)*, Addison-Wesley.

찾아보기

저자 소개

최 병 선 (崔 秉 善)

서울대학교 수학과 졸업 (이학사)

미국 Stanford대학교 대학원 졸업 (경제학 석사, 통계학 석사,

Ph. D. in Statistics <major> and Economics <minor>)

연세대학교 상경대학 교수 역임

현재 서울대학교 경제학부 교수 (재무경제학 담당)

E-mail bschoi12@snu.ac.kr

금융공학 VII: Scientific Computing for Finance and Economics

2018년 3월 16일 제1판 1쇄 인쇄

2018년 3월 26일 제1판 1쇄 발행

저 자 **최 병 선**

발행처 **김 구 재 단**

서울특별시 서대문구 충정로9길 10-10 3F (충정로2가) 우편번호 03676

등 록 2014. 8. 29 제25100-2014-000058호

전 화 02-3146-6923

팩 스 02-312-2675

E-mail smny76@kimkoo.org

ISBN 979-11-963396-0-9 93320 비매품

이 도서의 국립중앙도서관 출판예정도서목록(CIP)은 서지정보유통지원시스템 홈페이지 (http://seoji.nl.go.kr)와 국가자료공동목록시스템(http://www.nl.go.kr/kolisnet)에서 이용하실 수 있습니다.(CIP제어번호: CIP2018008631)

최병선 교수 Creative Commons Book List
(http://s-space.snu.ac.kr/handle/10371/314)

1. (제2판) SAS를 이용한 현대통계학 (이성백교수와 공저)

   http://s-space.snu.ac.kr/handle/10371/94393
2. 행렬의 대각화와 Jordan표준형 (이성백교수와 공저)

   http://s-space.snu.ac.kr/handle/10371/94394
3. Fourier 해석 입문

   http://s-space.snu.ac.kr/handle/10371/94413
4. Lebesgue적분 입문

   http://s-space.snu.ac.kr/handle/10371/94414
5. Wavelet 해석

   http://s-space.snu.ac.kr/handle/10371/94415
6. 회귀분석(상)

   http://s-space.snu.ac.kr/handle/10371/94433
7. 다변량시계열분석

   http://s-space.snu.ac.kr/handle/10371/94434
8. 이산형 재무모형의 수리적 배경

   http://s-space.snu.ac.kr/handle/10371/94455
9. 회귀분석(하)

   http://s-space.snu.ac.kr/handle/10371/94456
10. 단변량시계열분석

    http://s-space.snu.ac.kr/handle/10371/94457
11. 금융공학 IV: Monte Carlo Methods for Finance and Economics

    http://s-space.snu.ac.kr/handle/10371/98466
12. 금융공학 V: Introduction to Financial Engineering with R

    http://s-space.snu.ac.kr/handle/10371/99003
13. SAS/IML 입문

    http://s-space.snu.ac.kr/handle/10371/116955