福井大学
工学部　研究報告
第48巻　第1号
2000年3月

数値計算のための簡便なインタフェースの構築

河野 正幸*　長谷川 武光**　佐藤 義雄**　細田 陽介**

Making Template Interfaces for Numerical Subroutines

Masayuki KONO, Takemitsu HASEGAWA,
Yoshio SATO, and Yosuke HOSODA

(Received Feb. 29, 2000)

An user interface for computing with routines in the mathematical subroutine package NUMPAC is constructed. The interface enables users to compute with NUMPAC by writing simple commands in a single line without taking trouble to write tedious main program calling the subroutines in NUMPAC. The interface is supplied by making corresponding template with awk program for each NUMPAC subroutine.

Key Words : Template, NUMPAC, Numerical Subroutines, Awk

1　序論

数学のためのサブルーチンパッケージは、ソフトウェア再利用のための有効なやり方の一つである。ライブラリについてよく知っている専門家はそれを役に立つソフトウェアにすばやく結びつける。しかしながら、専門家のために有効でも普通のユーザにとって大きいライブラリを使うことは難しい。プログラマが数値計算をする場合、適切なルーチンを探し、サンプルプログラムを修正し、適切なライブラリをリンクしてプログラムをコンパイルし、コードをテストし（あるいは、デバッギングをし）、答えを見つけるための出力を解釈する必要がある。このプロセスは多くの時間と知識が必要である。

精密なインタフェースを凝ったソフトウェアライブラリに結びつけるような環境は、専門家にとって優れているけれど、それを作るためには、多くの時間が必要である。そのため、個人のプログラマが、早急に、ソフトウェアライブラリを利用するための簡便なインターフェースが望まれる。また、プログ

* 工学研究科情報工学専攻 ** 工学部情報メディア工学科

ラムの知識がないユーザにも容易に数値計算が出来るようなインターフェースが必要となる。

そこで、数値計算を行うときに、1行程度の文を書くだけで自動的に、適切なサブルーチンを呼ぶフォートランプログラムを書いたり、コンパイルしたり、実行したりできるインターフェースを、NUMPAC(名古屋大学を中心に開発されてきた数学ソフトウェアのサブルーチンパッケージ)を使用して作成した。作成にあたって、以下のことに注意した。

- インターフェースは、初心者に近付きやすく、また、経験の積んだユーザにとっても十分便利でなければならない。
- 問題の指定は、簡単でなければならない。

本論文では、第3章で、一般に数値計算を行なう場合の一例をフォートラン言語で書く場合について述べ、第4章で、インターフェースのプログラミング例について述べたあと、第5章で、インターフェースの使用例を述べ、最後に結論を述べる。

2 NUMPAC

NUMPAC[4]は、名古屋大学を中心に開発されてきた数学ソフトウェアのサブルーチンパッケージで、約1000の数値計算のサブルーチンが用意されている。NUMPACのマニュアルはインターネット[6]上で参照することができる。NUMPACのライブラリー・プログラムは大別して、関数副プログラムとサブルーチン副プログラムに分けられる。実数型の関数名はフォートランの暗黙の型指定の規約に従っている。関数を使用する場合には、それを引用しているすべてのプログラム単位で、それらの関数名に適当な型宣言を行わねばならないことである。倍精度実数型の関数名には、対応する実数型の関数名の先頭にそれぞれ"D"を加える。サブルーチン名の頭字には型の意義はない。同じ目的を持ち、型だけが異なる一群のサブルーチンは名前の末尾の1字によって区別される。その原則は、S(単精度実数用)、D(倍精度実数用)、Q(4倍精度実数用)、C(単精度複素数用)、B(倍精度複素数用)、Z(4倍精度複素数用)、V(ベクトル計算機単精度用)、W(ベクトル計算機倍精度用)、X(ベクトル計算機単精度複素数用)、Y(ベクトル計算機倍精度複素数用)である。

引数が制限を破った場合には、エラーとしてメッセージを印刷し、関数値を一様に0として計算を続行する。メッセージには関数名、引数値、関数値(0)及びエラーの理由が印刷される。エラーが続出して計算が全く意味のなくなる場合を考慮して、エラー処理プログラムでエラーの回数をカウントし、それがあらかじめ定められたある制限を超えた場合に計算を停止するようにしている。　数値積分ルーチンや微分方程式解法ルーチンでは関数ルーチンやサブルーチンを利用者側で作成することが要求される。この場合にその引数の数、型及び順序は指定されている通りにしなければならない。規定の引数以外の副引数が必要な場合は、これらをcommon領域にとり、主プログラムとの連絡を図るようにする。

3 フォートラン言語による数値計算

NUMPACのルーチンを使って数値計算を行なう場合、フォートラン言語でNUMPACのルーチンを呼ぶ主プログラムを書くことになる。

例えば、区間$(3,4)$において、非線形方程式

$$f(x) = \sin x = 0$$

の解をNUMPACにある二分法の副プログラム"noleqs"を使って、要求精度10^{-6}で計算しよう。まず、数値計算ルーチン"noleqs"を使うための主プログラムを自分で作成したり、サンプルプログラムを修正したりする。フォートラン言語で書く場合、最低でも以下の量の主プログラムを書く必要がある。

```
% cat main.f
        real x,fx,fun
        external fun
        call noleqs(3.0,4.0,fun,1e-6,1000,
       *x,fx,n,ill)
        write(*,*) x,fx,n,ill
        stop
        end
 c
        function fun(x)
        fun=sin(x)
        return
        end
```

プログラムが書けたら、二分法のルーチンが書かれたファイル"noleqs.f"を探してきて、作成した上記のプログラムをコンパイル・実行してはじめて答えが表示される。しかし、この作業は、多くの時間と、フォートラン言語の知識が必要になる。そのため、NUMPACのルーチンを簡単に使用するために上記の作業を自動的に行ない、フォートラン言語の知識のいらないインターフェースが必要になる。

本研究では、上記の問題を行なう時、入力を下記のように1行程度の文を書くだけで、計算ができるようにする。

```
noleqs 'sin(x)' 3 4 1e-6 1000
```

まずは、命令を解釈し、自動的に、適切なサブルーチンを呼ぶためのフォートランのプログラムを書くために、その基板となるテンプレートを用意する。

4 インターフェースのプログラミング

この節では、§3の例の非線形方程式を、二分法で計算するルーチン"noleqs"を使って解く方法を見ることによって、インターフェースの簡単な作り方[1]を紹介する。

4.1 テンプレートプログラミングの例

数値計算ルーチンを使うためのフォートランのプログラムを自動的につくるために、基板となるファイル(ここでは、テンプレートと呼ぶ)を用意する。テンプレートはインターフェースを通してフォートランのプログラムに変換される。

下記のプログラムは、数値計算ルーチン"noleqs"を呼ぶためのフォートランのプログラムを作る基となるテンプレートの簡単な一例を示している。

```
% cat noleq=m.tplt
   real fun,x
   external fun
   call noleqs(@A@, @B@, fun, @EPS@, @NMAX@, x, fx, n, ill)
   write(*,*) x,fx,ill
   stop
   end
c
   function fun(x)
   fun=@EQN@
   return
   end
```

このテンプレートは、フォートランのプログラムを基に簡単に作成することが可能である。簡単に作るには、5つのパラメータ"EQN"、"A"、"B"、"EPS"、"NMAX"、を、それぞれ"@"マークで囲むだけでよい。

実際は、上記の5つのパラメータに加えて、型宣言文(上記の例で、1行目real文)と、関数呼出文(call)で呼び出されるサブルーチン名(上記の例で、3行目noleqs)を、"@"で囲まれたパラメータに置き換えている。その理由の1つは、数値計算ルーチンの中で同じ目的を持ち、型だけが異なる命令は1つのテンプレートを使用できるからである。上記の例の場合、単精度型と倍精度型で1つのテンプレートを使っている。もう一つの理由は、ルーチンの引数の数と順序が同じ場合も、1つのテンプレートを使用できる。例えば、三角関数の計算は、引数が1つなので、1つのテンプレートを使用できる。また、主プログラムのテンプレートと関数fのプログラムのテンプレートを別のファイルにし、関数fのプログラムのテンプレートは、共通に使っている。

4.2 インターフェースのプログラミング例

先の例を実行する"noleqs"命令はコマンドラインから5つの独立変数"EQN"、"A"、"B"、"EPS"、"NMAX"を読んで、テンプレートに代入して、フォートランのプログラムに変換し、そのフォートランのプログラムをコンパイル・実行して、結果を出力する。このインターフェースでは、フォートランのプログラムからNUMPACのルーチンを呼ぶ。そして、インタフェースは、awk[2]で実行する。下記は、簡単なawkプログラムの一例である。

```
BEGIN { subarr["EQN"]=ARGV[1]
        subarr["A"]=ARGV[2]
        subarr["B"]=ARGV[3]
        subarr["EPS"]=ARGV[4]
        subarr["NMAX"]=ARGV[5]
        dotemplate("noleq=m.tplt","main.f", subarr)
        system("fort77 main.f noleqs.f ; a.out")
}
```

最初の5行はコマンドラインを読み、配列"subarr"にそれぞれ保存する。ここで、注意することは、配列の名前をテンプレートのパラメータの"@"で囲まれた名前と同じにする。また、6行目は、テンプレートのファイル"noleq=m.tplt"をフォートランでの主プログラム"main.f"(§3参照)に変換するための関

数"dotemplate"を呼ぶ。7行目は、この主プログラムと数値計算ルーチンの書かれているファイルとをコンパイルして、オブジェクトコードを実行して、そして、答えを出力する。ここでは、コンパイラとして、"fort77"を使用している。また、NUMPACの各ルーチンについては、あらかじめオブジェクトファイルを作成しておき、それを結合編集して、それをリンクすることで、計算時間を短くしている。

4.3 関数"dotemplate"の例

関数"dotemplate"は、テンプレートをフォートランのプログラムに変換する関数で、テンプレートの書かれたファイルの名前(templatefile)、出力ファイルの名前(outfile)、入力の値を保存した配列(subarr)を引数に持つ。ここに簡単な関数"dotemplate"の例がある。

```
function dotemplate(templatefile, outfile, subarr) {
  while (getline<templatefile>0) {
    temp=$0
    for (i in subarr)
      gsub("@"i "@", subarr[i], temp)
        print temp > outfile
  }
  close(templatefile)
  close(outfile)
}
```

関数"dotemplate"は、テンプレートファイル(変数"templatefile")を1行ずつ読み込んで(2行目、"getline"関数)、その行(変数"temp")に"@"で囲まれたパラメータがあればその名前と同じ名前を持つ配列"subarr"の内容を代入する(5行目、"gsub"関数)。そして代入された行(変数"temp")を出力ファイル(変数"outfile")に書き込んでいく。これをテンプレートの終わりまで繰り返す。その結果、テンプレートのファイルが、フォートランのファイルとして出力ファイルに出力される。

この章で紹介したプログラムは、正しいプログラムを正しく処理する。それは、§4.1の11行のテンプレートと§4.2の18行のawk(関数"dotemplate"を含む)で実行される。完全な"noleqs"プログラムはエラーの調査により注意深い。すなわちコマンドラインは正確に5つの引数をもっているか？ フォートランは正確にコンパイルしているか？ そうでなければ、エラーメッセージを報告しなさい、など。テンプレート自身とエラー調査のためのコードとテンプレート解釈を含む完全なプログラムは約150行必要とする（テンプレートで20行程度必要）。インターフェース作成にあたって、awkを使用したのは、perlに比べて、プログラムの行数が半分くらいで、処理も軽く、実行時間が短いためである。NUMPACのための実行可能な命令は、442種類1098個である。ちなみに、テンプレートファイルの数は、303個である。

4.4 インターフェースの流れ

インターフェースの簡単な流れを図1に示し、この章の例"noleqs"を使って簡単に説明する。説明には、§4.1のテンプレートのファイル"noleq=m.tplt"、§4.3の関数"dotemplate"を含めた§4.2のインターフェースのプログラムファイル"noleqs"を使用している。

1. 命令と引数を1行に入力する
   "noleqs 'sin(x)' 3 4 1.0e-6 1000"

2. 命令"noleqs"を実行する
   インターフェースの開始

3. 2番目の引数が式なので配列"subarr"に格納する
(インターフェースの1行目subarr["EQN"]="sin(x)")

4. 3番目の引数が数字なので配列"subarr"に格納する
(インターフェースの2行目subarr["A"]="3")

5. 4番目の引数が数字なので配列"subarr"に格納する
(インターフェースの3行目subarr["B"]="4")

6. 5番目の引数が数字なので配列"subarr"に格納する
(インターフェースの4行目subarr["EPS"]="1.0e-6")

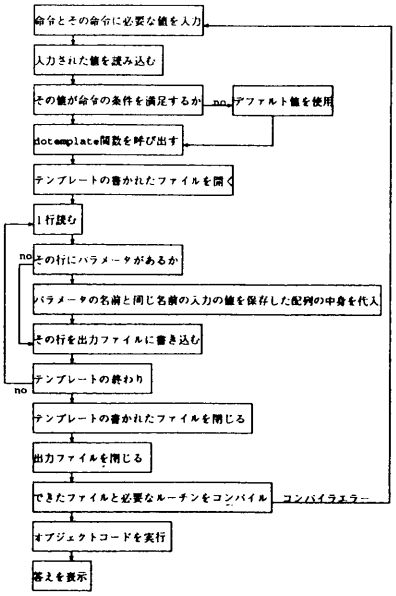

図1: インターフェースの流れ

7. 6番目の引数が数字なので配列"subarr"に格納する
(インターフェースの5行目subarr["NMAX"]="1000")

8. 関数dotemplateを呼び出す(インターフェースの6行目)
引数はテンプレートファイル templatefile="noleq=m.tplt"と出力ファイル outfile="main.f"と配列 subarr

9. テンプレートファイル"noleq=m.tplt"から1行ずつ読んで(関数"dotemplate"の2行目 while(getline < templatefile>0))、変数tempに代入する

   (1) テンプレートファイル1行目(関数"dotemplate"の3行目 temp="real fun,x")
   - "@"で囲まれたパラメータが見付からない
   - 出力ファイル"main.f"に書き込む
     (関数"dotemplate"の6行目 print temp > outfile)

   (2) テンプレートファイル2行目
   (関数"dotemplate"の3行目 temp="external fun")
   - "@"で囲まれたパラメータが見付からない
   - 出力ファイル"main.f"に書き込む
     (関数"dotemplate"の6行目 print temp > outfile)

   (3) テンプレートファイル3行目
   (関数"dotemplate"の3行目
   temp="call noleq(@A@,@B@,fun,@EPS@,@NMAX@,x,fx,n,ill)")
   - "@"で囲まれたパラメータが見付かる
     "@"で囲まれたパラメータが存在する間変換する
     (関数"dotemplate"の4行目 for(i in subarr))

i. 文字列 "@A@"を配列 subarr["A"] の要素に変換
(関数 "dotemplate"の 5 行目 gsub("@"A"@",subarr["A"],temp))
temp="call noleq(3,@B@,fun,@EPS@,@NMAX@,x,fx,n,ill)"

ii. 文字列 "@B@"を配列 subarr["B"] の要素に変換
(関数 "dotemplate"の 5 行目 gsub("@"B"@",subarr["B"],temp))
temp="call noleq(3,4,fun,@EPS@,@NMAX@,x,fx,n,ill)"

iii. 文字列 "@EPS@"を配列 subarr["EPS"] の要素に変換
(関数 "dotemplate"の 5 行目 gsub("@"EPS"@",subarr["EPS"],temp))
temp="call noleq(3,4,fun,1.0e-6,@NMAX@,x,fx,n,ill)"

iv. 文字列 "@NMAX@"を配列 subarr["NMAX"] の要素に変換
(関数 "dotemplate"の 5 行目 gsub("@"NMAX"@",subarr["NMAX"],temp))
temp="call noleq(3,4,fun,1.0e-6,1000,x,fx,n,ill)"

- 出力ファイル "main.f"に書き込む
(関数 "dotemplate"の 6 行目 print temp > outfile)

(4) テンプレートファイル 4 行目 (temp="write(*,*) x,fx,ill")

- "@"で囲まれたパラメータが見付からない
- 出力ファイル "main.f"に書き込む
(関数 "dotemplate"の 6 行目 print temp > outfile)

(5) テンプレートファイル 5 行目 (temp="stop")

- "@"で囲まれたパラメータが見付からない
- 出力ファイル "main.f"に書き込む
(関数 "dotemplate"の 6 行目 print temp > outfile)

(6) テンプレートファイル 6 行目 (temp="end")

- "@"で囲まれたパラメータが見付からない
- 出力ファイル "main.f"に書き込む
(関数 "dotemplate"の 6 行目 print temp > outfile)

(7) テンプレートファイル 7 行目 (temp="c")

- "@"で囲まれたパラメータが見付からない
- 出力ファイル "main.f"に書き込む
(関数 "dotemplate"の 6 行目 print temp > outfile)

(8) テンプレートファイル 8 行目 (temp="function fun(x)")

- "@"で囲まれたパラメータが見付からない
- 出力ファイル "main.f"に書き込む
(関数 "dotemplate"の 6 行目 print temp > outfile)

(9) テンプレートファイル 9 行目 (temp="fun=@EQN@")

- "@"で囲まれたパラメータが見付かる
"@"で囲まれたパラメータが存在する間変換する
(関数 "dotemplate"の 4 行目 for(i in subarr))

i. 文字列 "@EQN@"を配列 subarr["EQN"] の要素に変換
(関数 "dotemplate"の 5 行目 gsub("@"EQN"@",subarr["EQN"],temp))
temp="fun=sin(x)"

- 出力ファイル "main.f" に書き込む
  (関数 "dotemplate" の 6 行目 print temp > outfile)

(10) テンプレートファイル 10 行目 (temp="return")

- "@" で囲まれたパラメータが見付からない
- 出力ファイル "main.f" に書き込む
  (関数 "dotemplate" の 6 行目 print temp > outfile)

(11) テンプレートファイル 11 行目 (temp="end")

- "@" で囲まれたパラメータが見付からない
- 出力ファイル "main.f" に書き込む
  (関数 "dotemplate" の 6 行目 print temp > outfile)

10. テンプレートファイルを閉じる (関数 "dotemplate" の 7 行目 close(templatefile))
11. 出力ファイルを閉じる (関数 "dotemplate" の 7 行目 close(outfile))
12. できたファイル "main.f" と必要なルーチン "noleqs.f" をコンパイルする
    (インターフェースの 7 行目 system("fort77 main.f noleqs.f")
13. 実行、結果の表示 (インターフェースの 7 行目 a.out)、命令 "noleqs" の終了

5 インタフェースの使用法

本章では、インタフェースの使い方について例を示して紹介する。このインタフェースは、WWW を使用した NUMPAC の案内システムのページである NetNUMPAC Guide Page[6] から使用可能である。

5.1 非線形方程式の解法の例

区間 $(3,4)$ において方程式

$$f(x) = \sin(x) = 0$$

の解を 10^{-6} の精度で二分法を利用して求めるプログラム "noleqs" を使って解く場合、Top 画面 (図 2) に下記のように計算に必要な引数をスペースで区切って 1 行に書く。

```
noleqs 'sin(x)' 3 4 1e-6 1000
```

ここで、"noleqs" 命令は "noleqs" のプログラムを使うということを示し、続いて、式 $f(x)$ をシングルコーテーションマーク (') で囲み、順に区間の左端、右端、解の精度、関数 $f(x)$ の評価回数の上限を入力する。

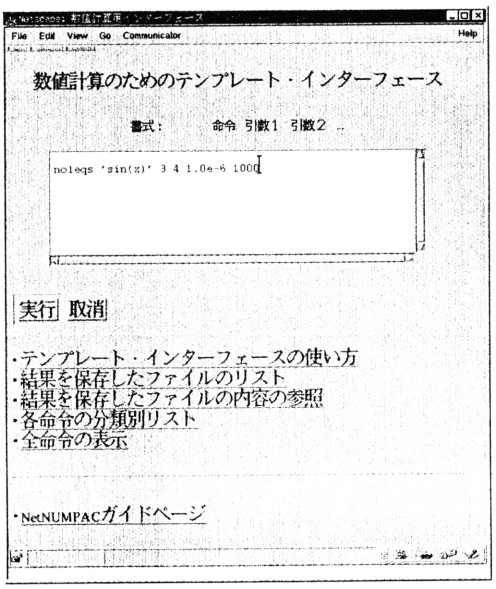

図 2:top の画面

引数の順序は、命令によって決まっており、§5.5で紹介するヘルプで確かめることができる。入力した命令、引数が正しければ、入力ボックスの下の"実行"ボタンを押すことで、計算が開始される。"実行"ボタンの隣の"取消"ボタンを押すと入力した文字列がすべて消される。正しく実行されると、自動的にフォートランのプログラムを作成して、そのプログラムと計算に必要なルーチンとをコンパイルして、オブジェクトを実行して、答えを出力する(§4)。画面上(図3)に解の近似値(x)、解の近似値での関数 $f(x)$ の値(y)、関数 $f(x)$ の評価回数(n)、エラーコード(ill)が表示される。エラーコード(ill)は0が表示されれば正常に終了したことを示す。図3の1行目は、その計算をするにあたって入力した値を括弧で囲んで示している。

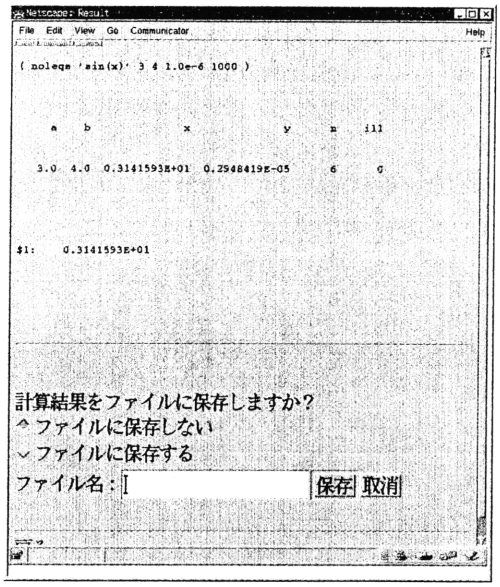

図3: 結果の出力

5.2 計算結果の保存

このインターフェースでは、計算結果は指示に従って、ファイルに保存する。もし、計算結果を保存する場合、図4のように、"ファイルに保存する"を選び、ファイル名(この場合、例えばtest.txt)を入力する。ファイル名を入力したら、"保存"ボタンを押す。ここで、"取消"ボタンは、入力した文字列(ファイル名)を消去する。また、"ファイルに保存しない"を選ぶとファイルに計算結果が保存されないことをメッセージで表示する。指定したファイル(この場合、test.txt)が存在しなければ、指定したファイル名で計算結果が保存され、その旨をメッセージで表示する。その後で、ファイルの内容を確認することもできる。もし、指定したファイルが存在すれば、図5のように、メッセージでその旨を表示し、画面の下部に存在するファイルの内容を表示し、もう一度ファイル名の入力を促す。

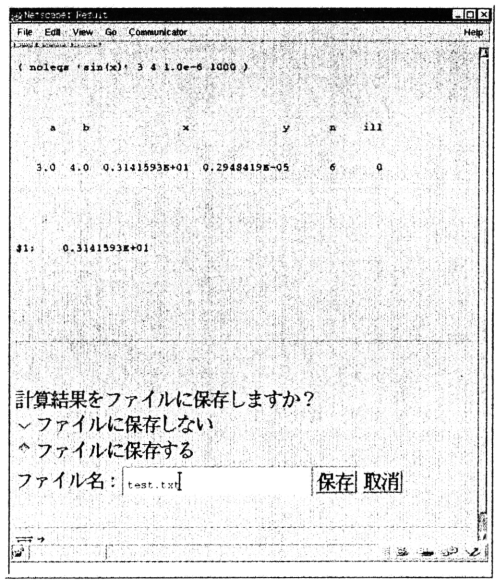

図4: ファイル名の入力

そこで、”ファイル名を変更する”を選び、別のファイル名を入力し、そのファイルに保存するか、あるいはすでに存在するファイル(test.txt)に上書きして保存する場合、”ファイルを上書きする”を選び、”保存”ボタンを押す。”ファイルに保存しない”を選ぶとファイルに計算結果が保存されないことをメッセージで表示する。

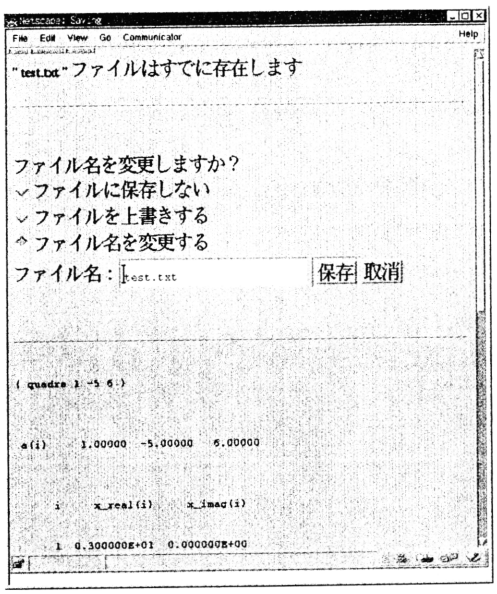

図 5: ファイル名の再入力

5.3 さまざまな入力

正しく計算を行う場合、それぞれの命令によって決められた引数を正しく入力することが大切である。しかし、引数の中には、だいたい毎回決まった値をとるようなものもある。したがって、このインターフェースには入力をより簡単にするために、それぞれの命令によってデフォルト値が用意されている。例えば、noleqs 命令では、命令を入力する時に下記のように、”-”マークを数値の代わりに入力するとデフォルト値を使って計算される。この場合、解の精度がデフォルト値(この場合、1.0e-6)で計算される。

```
noleqs 'sin(x)' 3 4 - 1000
```

また、下記のように、後ろの方の値を、省略するとデフォルト値で計算される。この場合、解の精度と関数 $f(x)$ の評価回数の上限がデフォルト値(前者が、1.0e-6、後者が、1000)で計算される。

```
noleqs 'sin(x)' 3 4 _ _
```

このインターフェースでのデフォルト値は、命令によってそれぞれ適切な値になっているが、他の引数の値によってはふさわしくない値になる可能性がある。また、引数の順序は、それぞれの計算に重要な値の順番になっていて、特に、命令のすぐあとの入力の引数は、”noleqs”命令の場合”式”になるが、計算するのになくてはならない値がくる場合が多いため、必ず値を書く必要がある。”noleqs”命令のように式をシングルコーテーションマーク(')で囲むことはこのインターフェースの原則で、もし囲まれていない場合は、その文字列がファイル名と解釈される。したがって、エラー扱いになり、正しい入力をするようにメッセージを表示する。

”noleqs”命令は、計算は全て単精度実数型で行なわれている。もし、同じ計算を倍精度実数型で行いたい場合、下記のように、命令の語尾を”s”から”d”に換えると倍精度実数型で計算される。

```
noleqd 'sin(x)' 3 4 1e-6 1000
```

命令によっては、s(単精度実数型)、d(倍精度実数型)のほかに、i(整数型)、a(文字型)、c(単精度複素数型)、b(倍精度複素数型)、v(ベクトル計算機単精度型)、w(ベクトル計算機倍精度型)、x(ベクトル計算機単精度複素数型)、y(ベクトル計算機倍精度複素数型)の型が指定できる。どの型での計算が可能

かは、§5.5で紹介するヘルプを使って、確かめられる。命令の型を指定しない時は、単精度実数型で計算される。つまり、noleqsを下記のように書くこともできる。

```
noleq_ 'sin(x)' 3 4 1e-6 1000
```

計算結果を保存することができるので(§5.2)、保存した結果を使って別の計算をすることもできる。例えば、ファイル"test.txt"に下記のように保存されているとする。括弧で囲まれた文字列は、その計算をしたときに、入力した値を表している。

```
% cat test.txt

(  noleqs 'sin(x)' 3 4 1e-6 1000  )

            x                 f(x)   n   ill
3.141592741    2.948419251e-6    6    0

$1 : 3.141592741
```

そこで、$\sin((\pi\ /2) \times x)$の計算(命令は、sinhps)を、このファイルの中の値(x=3.141592741)を使う場合、命令に続いてファイル名"test.txt"を入力する。

```
sinhps test.txt
```

もし、指定したファイル(例えば、"test2.txt")に下記のように、複数の結果が保存されている場合、どの結果を使用するかを"$"記号と数字で指定する。ここで、"$"記号の後の数字は、ファイルに保存された順番を表している。

```
% cat test2.txt

(  sinhps 0.2  )

      x              sinhp(x)
   0.200    0.309017003E+00

$1 : 0.309017003E+00


(  noleqs 'sin(x)' 3 4 1e-6 1000  )

            x                 f(x)   n   ill
3.141592741    2.948419251e-6    6    0

$2 : 3.141592741
```

例えば、ファイルの中の2番目の結果の値を使って命令 sinhps を実行する場合、ファイル名に続いて、"$2"を入力する。

```
sinhps test2.txt $2
```

これは、下記の入力と同じ意味である。

```
sinhps 3.141592741
```

どの結果を使うか指定しない場合、ファイルの中の1番目の結果を使う (ここでは、"test2.txt"の $1=0.309017003E+00)。

図2の入力フォーム内で複数の命令を改行で区切れば入力した順番に計算する。その時、その中の計算した結果を使用する場合、下記のように"#"記号と数字で指定することで計算できる (ここでは、"#2"は、2番目の計算の結果を使う事を示している、つまり、#2=3.141592741 で、"noleqs"の計算結果である)。"#"の後の数字は、計算される順番 (入力した順番) を示している。

```
coshps 0.5
noleqs 'sin(x)' 3 4 1e-6 1000
sinhps #2
```

5.4 結果を保存したファイルの参照

計算結果を保存したファイルの内容を見るには、コマンドラインから、"cats"という命令を使う。入力の方法として、命令に続いてファイル名を入力する。ファイル名は複数入力可能である。

```
cats test.txt ...
```

また、別の方法として、Top画面 (図2) で"結果を保存したファイルの内容の参照"を選ぶと図6が表示される。そこで、計算結果を保存したファイル名 (例えば、test.txt) を入力して、"表示"のボタンを押すと、指定されたファイルが存在すれば、そのファイルの内容が表示される (図7)。ただし、この場合、ファイルは、1つづつしか参照できない。ファイルの1行目は、計算をする時に入力した値が括弧で囲まれて表示されている。

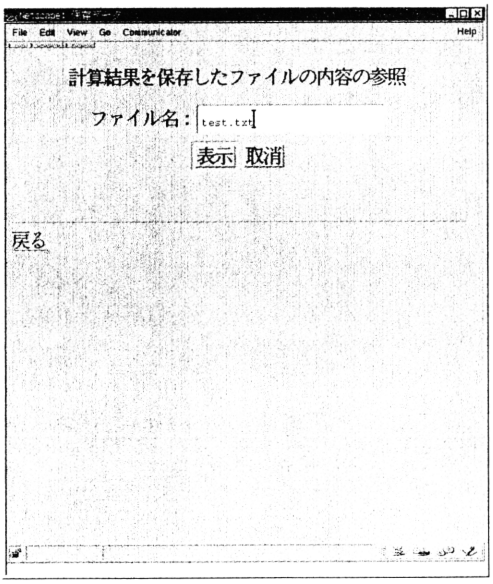

図 6: ファイル名の入力

もし、保存されたファイル名が分からない場合は、コマンドライン上で、"dirs"という命令を入力すれば、今までに記録されたファイルのリストが表示される。そこで、ファイル名を確認して、その後で、見たいファイルを先程説明した"cats"命令で見ることができる。

```
dirs
cats test.txt
```

また、Top画面(図2)で"結果を保存したファイルのリスト"を選ぶ。すると保存されたファイルの一覧(図8)がリンク付きで表示され、希望するファイル名を選ぶと、そのファイルの内容が表示される。

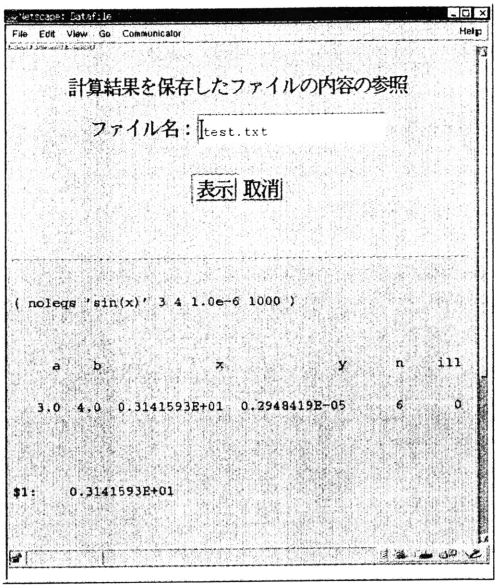

図 7: ファイルの参照

5.5 各命令のヘルプ

NUMPACには、約1000のルーチンがあり、このインターフェースで、それらのルーチンを使って数値計算をする命令に、どのようなものがあり、1つの命令に対して、入力の仕方や引数の順序、制限を示すためにヘルプ機能がある。入力の方法として"help"命令を引数なしで入力すると、全命令の一覧が表示される。そこで、使用可能な命令の名前を確認する事ができる。あるいは、Top画面(図2)で"全命令の表示"を選ぶと使用可能な命令のリストが表示される(図9)。

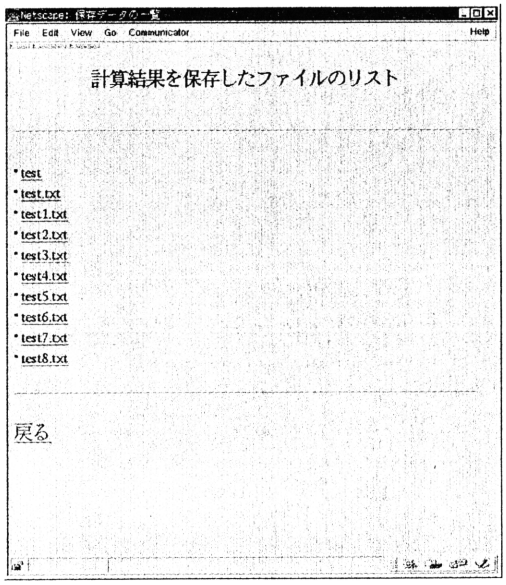

図 8: 保存ファイル名のリスト

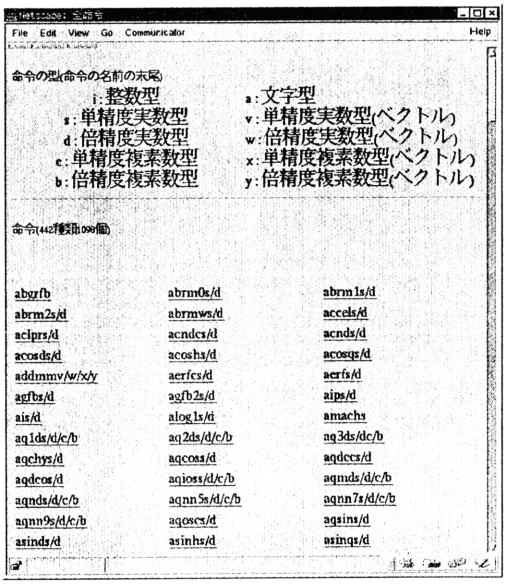

図 9: 全命令の表示

そこで、見たい命令を選ぶとその命令のヘルプを見る事ができる。図9の画面上部には、命令の語尾1文字がどういう計算の型を表しているかを表示している。"help"命令の後、ある命令のコマンドを続けて入力するとその命令のヘルプが表示される。例えば、非線形方程式を二分法で解く命令"noleq"の説明が見たいときは、下記のように入力する。

help noleq

"help"命令に続いて、命令名を複数入力すれば、その分のヘルプを見る事ができる。あるいは、図9で、命令の部分を選ぶと、その命令のヘルプを見る事ができる。また、別の表示方法として、一部の命令に関して、Top画面で(図2)"各命令の分類別リスト"を選ぶと分類別が簡単な内容説明と共にリストで表示される(図10)。そこで、命令の名前を選ぶとヘルプを見る事ができる。ここでの分類は、NetNUMPAC Guide Page [6]の分類に基づいている。

例として、"noleq"命令のヘルプを図11に示す。図11のnoleq命令のヘルプ画面は、上から、

- 入力方法
- 入力ボックス、実行ボタン
- 命令の計算内容の簡単な説明
- 引数の意味と制限などの説明

になっている。

入力ボックスに、ヘルプを見ながら、引数を入力して、実行ボタンを押すと計算する事ができる。ただし、ここでは、1つの命令のみ実行される。

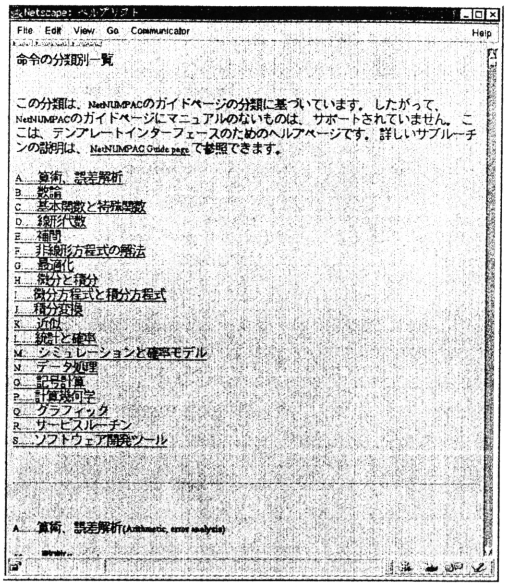

図10: 各命令の分類別リスト

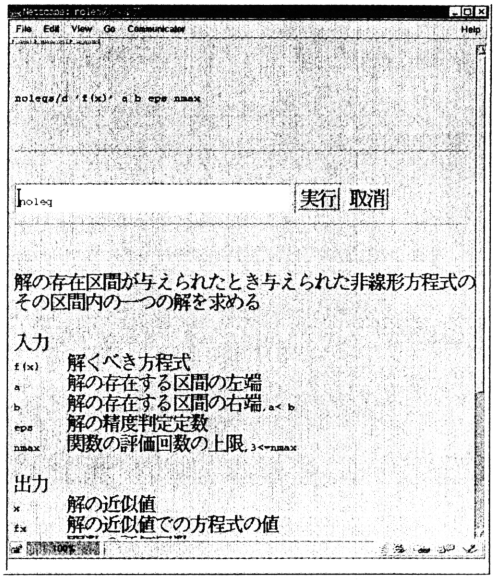

図11: ヘルプの例；"noleq"

5.6 使い方マニュアル

Top 画面 (図 2) で”テンプレート・インターフェースの使い方”を選ぶとこのインターフェースの使い方が表示される。このマニュアルは、§5をまとめたものを表示している。

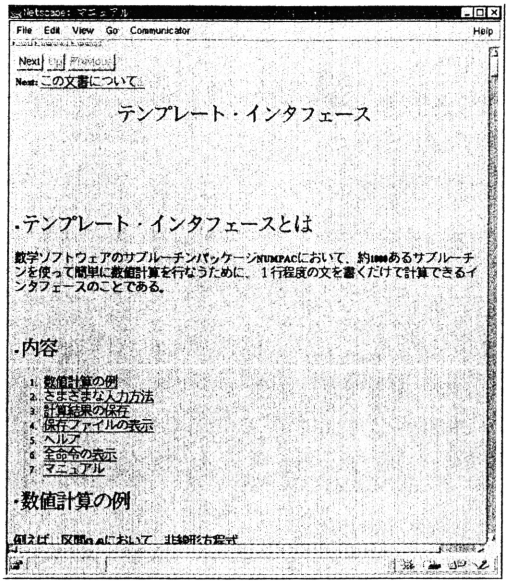

図 12: マニュアルの表示

6 結論

NUMPAC の約 1000 個のルーチンがテンプレート形式で簡単に計算が実行できるようになった。多数の数値計算を 1 行程度の文を書くだけで実行できる。したがって、プログラムの知識がないユーザにも使えるようになった。そして、複数の命令を実行するには、順に 1 行に 1 命令を書き連ねる事で可能になった。また、計算結果をファイルに保存することで、その値を使って、別の計算をする事が可能になった。デフォルト値をそれぞれの命令に定義しているので、一層入力が簡単になった。NUMPAC のルーチンについて、あらかじめ、オブジェクトファイルを作成しておき、それを使う事で、計算時間の短縮を図った。今後の課題として、式が場合分けを必要とする場合、それに図形を書くためのルーチン、に対応させることが必要になる。また、命令が多数あるため、ヘルプ機能をより使いやすくする必要がある。

参考文献

[1] J.L.Bentley, M.F.Fernandez, B.W.Kernighan, and N.L.Schryer, Template-Driven Interfaces for Numerical Subroutines. ACM Trans. Math. Softw., 19, (1993) 265-287.

[2] D.Dougherty 著, 福崎俊博 訳, sed&awk プログラミング. アスキー, (1997).

[3] 原田賢一著, Fortran77 プログラミング. サイエンス, (1993).

[4] 二宮市三, 名古屋大学数学ライブラリー NUMPAC について, 名古屋大学計算機センターニュース Vol.15, No.2, (1984) p.222.

[5] Fujitsu, NUMPAC 使用手引書 Vol.1-3.

[6] NetNUMPAC ガイドページ, http://numpac.fuis.fukui-u.ac.jp/numpac/.

[7] 河野 正幸著, 平成 9 年度卒業論文「数値計算ルーチンのためのテンプレートインターフェースの作成」(1998).

[8] 角谷 秀治著, 平成 10 年度卒業論文「数値計算パッケージ NUMPAC の使いやすいインターフェースの開発」(1999).