Универзитет у Београду
Машински факултет

Радиша Ж. Јовановић

Matlab и Simulink

у аутоматском управљању

Универзитет у Београду
Машински факултет

Matlab и Simulink
у аутоматском управљању

Радиша Ж. Јовановић

Београд, 2021.

Др Радиша Ж. Јовановић, редовни професор
Универзитет у Београду, Машински факултет

Matlab и Simulink у аутоматском управљању

II издање

Рецензенти:
Проф. др Зоран Рибар, редовни професор у пензији,
Универзитет у Београду, Машински факултет
Проф. др Зоран Бучевац, редовни професор у пензији,
Универзитет у Београду, Машински факултет

Издавач:
Машински факултет Универзитета у Београду
ул. Краљице Марије 16, 11120, Београд 35
тел. (011) 3302-384
факс (011) 3370-364

За издавача:
Проф. др Радивоје Митровић, декан

Главни и одговорни уредник:
др Милан Лечић, редовни професор
Председник комисије за издавачку делатност
Машинског факултета у Београду

Штампање II издања одобрили:
Комисија за издавачку делатност Машинског факултета у Београду
и
Декан Машинског факултета одлуком бр. 26/2021 од 09.09.2021. године

Дизајн корица:
Мина Михаиловић

Тираж:
300 примерака

Штампа:
PLANETA PRINT, Београд

ISBN 978-86-6060-091-4

Садржај

ПРЕДГОВОР

Књига *Matlab и Simulink у аутоматском управљању* је конципирана тако да обухвата области предвиђене наставним планом и програмом предмета *Програмирање у аутоматском управљању*, који се слуша на Основним академским студијама на Машинском факултету у Београду. Међутим, узимајући у обзир актуелност и значај упознавања са неким од софтверских алата за различите врсте инжењерских прорачуна и анализа (а Matlab је један од најзаступљенијих, готово незаобилазан), књига је написана тако да могу да је користе и студенти са других факултета у чијим наставним плановима и програмима је заступљења ова проблематика. У том смислу, програмски језик Matlab (у даљем тексту Матлаб) и његов пакет Simulink (у даљем тексту Симулинк) представљени су у прва три дела књиге. Четврти и пети део књиге се односе на примену Матлаба и Симулинка у области система аутоматског управљања, и добрим делом представљају подршку настави из предмета *Основе аутоматског управљања* који је обавезан предмет на основним академским студијама на Машинском факултету у Београду. Коначно, књига може да послужи као основа за будуће напредно коришћење Матлаба и Симулинка.

Композиција књиге је организована у пет делова. Основа програмирања у Матлабу је изложена у првом делу. Прво поглавље доноси увод о Матлабу као моћном програмском језику, опис његовог развојног окружења и основних операција. Променљиве и основни типови података тема су другог поглавља. У трећем поглављу уводи се концепт низа, који представља фундаментални елемент у Матлабу, и детаљно се обрађују нумерички низови (вектори и матрице), знаковни низови и низови ћелија, као и основне операције над њима. Први део се наставља четвртим поглављем које се бави писањем програма у Матлабу, почев од једноставних скриптова и логичких инструкција до различитих типова функција и функцијских датотека. У петом поглављу се разматрају симболички пакет Матлаба и методе за симболичко решавање алгебарских и диференцијалних једначина. Шесто поглавље је посвећено релационим и логичким операторима и контроли тока програма. Пажња је у седмом поглављу посвећена и делу

графичких могућности Матлаба а које се односе на дводимензионалну графику.

У другом делу књиге се излажу основне нумеричке методе које су садржане и имплементиране у уграђеним функцијама Матлаба. Системи линеарних једначина и проблеми сопствених вредности матрица су тема осмог поглавља. Девето поглавље је посвећено проблему апроксимације функције једне променљиве, при чему значајно место заузима и интерполација алгебарским полиномима и сплајновима, као и метода најмањих квадрата као једна од метода за апроксимацију функција. У десетом поглављу се илуструју методе за решавање нелинеарних алгебарских једначина. Садржај једанестог поглавља чине основе метода нумеричког диференцирања и интеграљења, при чему је посебна пажња посвећена методама на којима се заснивају уграђене функције Матлаба за нумеричко диференцирање и интеграљење. У дванаестом поглављу се разматрају методе за решавање обичних диференцијалних једначина и њихова имплементација у Матлабу.

Трећи део књиге се односи на Симулинк, где се у тринаестом и четрнаестом поглављу излажу основни елементи Симулинк модела и основни блокови из библиотеке блокова. Петнаесто поглавље је посвећено примени Симулинка у симулацији динамичких система, линеарних и нелинеарних. Четврти део књиге кроз шеснаесто поглавље доноси основне појмове из моделовања динамичких система, са посебним освртом на особине и облике модела линеарних стационарних динамичких система. Примена Лапласове трансформације у одређивању одзива линеарних стационарних система третира се у петом делу књиге, у седамнаестом поглављу. Осамнаесто поглавље се детаљно бави представљањем линеарних система у Матлабу и њиховим различитим моделима. Анализа линеарних система, аналитичка и нумеричка, теме су деветнаестог поглавља.

Изложени материјал је илустрован бројним примерима. Тамо где је неопходно, дати су и резултати извршавања одређених наредби, скриптова, а одређени број примера, посебно у петом делу, урађен је и аналитичким путем.

Рецензентима на корисним сугестијама, као и свима онима који су на неки начин помогли у току писања ове књиге, аутор се најтоплије захваљује.

Београд, март 2016. године Аутор

Предговор другом издању

У другом издању исправљено је неколико грешака уочених у првом издању. Аутор позива све читаоце ове књиге да својим сугестијама, примедбама и указивањима на евентуалне грешке, помогну у побољшању следећег издања.

Београд, септембар 2021. године Аутор

ПОГЛАВЉЕ 2

Промењљиве и типови података

2.1 Промењљиве

Концепт промењљивих, чије се вредности мењају у току извршавања програма, представља саставни део свих програмских језика. Појам промењљиве у Матлабу (и у програмском језику уопште) означава апстрактан појам меморијске ћелије рачунара или скупа меморијских ћелија. Другим речима, промењљиве представљају у неку руку имена (симболичка) меморијских локација, и свакако је програмерима лакше и природније да раде са таквим, симболичким именима, него са њиховим апсолутним адресама. Дефинисање промењљиве се у Матлабу врши наредбом доделе.

2.1.1 Оператор доделе

У Матлабу се знак = назива *оператором доделе*. Он овде има другачије значење од знака једнакости познатог из математике. Када се откуца на пример $x = 5$, то говори Матлабу да додели вредност 5 промењљивој x. Према томе, овај оператор додељује вредност промењљивој:

$$ImePromenljive = \text{нумеричка вредност или израз}$$

Претходна употреба се не разликује од оне познате из математике. Међутим, у Матлабу је могуће имати и овакав облик: $x = x + 2$. Ова наредба говори Матлабу да дода број 2 тренутној вредности промењљиве x и замењује тренутну вредност од x са том новом вредношћу. Уколико оригинално x има вредност 5, његова нова вредност ће бити 7. Ова примена оператора = је различита од његове примене у математици. На пример, у математици једначина $x = x + 2$ нема смисла, јер она имплицира да је $0 = 2$.

На левој страни оператора доделе може бити једна и само једна промењљива. Стога се у Матлабу не може користити, на пример, овако нешто: $6 = x$, јер би то значило да нумеричкој вредности треба да се додели садржај

могу садржати слова, цифре и карактер доња црта (подвлака), при чему морају почињати словом. Матлаб прави разлику између великих и малих слова, тако да су АУ, Ау, ау и аУ четири различите променљиве. При избору имена треба избегавати имена уграђених функција, као што су `cos`, `sin`, `exp`, `sqrt`, `min`, итд. Функција чије се име искористи за дефинисање променљиве не може се више користити. На пример, ако се дефинишу променљиве:

```
>> sqrt = 10, pi = 6
sqrt =
    10
pi =
     6
```

тада се позивом `sqrt(4)` добија порука о грешци, а константа π има вредност која није она која се очекује. Претходним наредбама редефинисани су и константа и функција, а објашњење је у начину рада интерпретатора. Наиме, када интерпретатор препозна ниску у некој наредби, он прво покушава да нађе њену интерпретацију у радном простору. Ако интерпретација у радном простору не постоји, онда интерпретатор прелази на претраживање садржаја директоријума који је приказан у прозору радног директоријума, затим претражује основни радни директоријум, и на крају системске делове Матлаба у којима се налазе уграђене функције Матлаба. Према томе, с обзиром да су вредности променљивих `pi` и `sqrt` дефинисане у радном простору Матлаба, оне се у њему и налазе, па их и интерпретатор одмах налази. Да ли се може поправити ово што је учињено? Наравно да може. Неопходно је само обрисати променљиве које су креиране у радном простору Матлаба.

```
>> clear sqrt pi
>> pi
ans =
    3.1416
```

Адреса променљиве

Адреса променљиве је адреса меморијске локације коју променљива представља. Адресе променљивих неће бити предмет даљих излагања.

Тип променљиве

Тип променљиве одређује подручје вредности које се променљивој могу доделити, као и скуп операција које се могу обављати над променљивама тог типа.

Вредност променљиве

Вредност променљиве је садржај меморијске локације (или више њих) коју променљива представља. Меморијска локација садржи одговарајући број бајтова за запис вредности одређеног типа променљиве. О величини меморијског простора за смештање одређених типова променљивих биће речи у наставку.

Видљивост променљиве

Видљивост неке променљиве односи се на подручје наредби програма у којима је променљива ,,видљива", тј. може бити референцирана. Према

ПОГЛАВЉЕ 3

Низови и операције над низовима

3.1 Врсте низова

Низови представљају фундаменталну структуру података у Матлабу. У науци и инжењерству, једнодимензионални низови представљају векторе, а дводимензионални низови представљају матрице.

Слика 3.1: Класификација низова у Матлабу

постоји нека веза. Једна од тих веза је да је разлика између суседних елемената иста. То је случај у разним мерењима, аквизицији података, управљању, када се унапред зна периода одабирања, и подаци се прикупљају и шаљу у унапред дефинисаним временским тренуцима, који су еквидистантни. Тада је могуће имати велики број елемената и применом оператора : (две тачке) вектор се може дефинисати првим и последњим елементом, и кораком између њих:

$$\mathbf{x} = [\text{први\_елемент} : \text{корак}: \text{последњи\_елемент}]$$

Угласта заграда није обавезна у претходној наредби. Пример креирања вектора са кораком 2 између елемената на поменути начин би био:

```
>> x = 1:2:7
x =
     1     3     5     7
```

Уколико је корак између елемената једнак 1, онда се он може изоставити из наредбе. Тако, пример са почетка би био:

```
>> vekt_vrsta = 1:4
vekt_vrsta =
     1     2     3     4
```

Треба уочити да ће први операнд из наредбе увек бити елемент вектора који се дефинише, док десни операнд не мора. То зависи од величине корака. У примеру

```
>> y = 1:2:8
y =
     1     3     5     7
```

је очито да је први елемент дефинисаног вектора једнак вредности левог операнда (1), док десни операнд 8 није елемент. Дакле, последњи елемент вектора ће бити последњи елемент из низа који није већи од десног операнда.

Такође, сви операнди у наредби не морају бити цели бројеви, а корак може бити и негативна вредност:

```
>> x = 10:-3:0
x =
    10     7     4     1
```

Оператор : се може користити само за дефинисање вектора врста, тј. матрице типа $1 \times n$. То значи да се на овај начин не може креирати вектор колона. Међутим, спас постоји у коришћењу особина матрица да се вектор колона може добити транспоновањем вектора врсте:

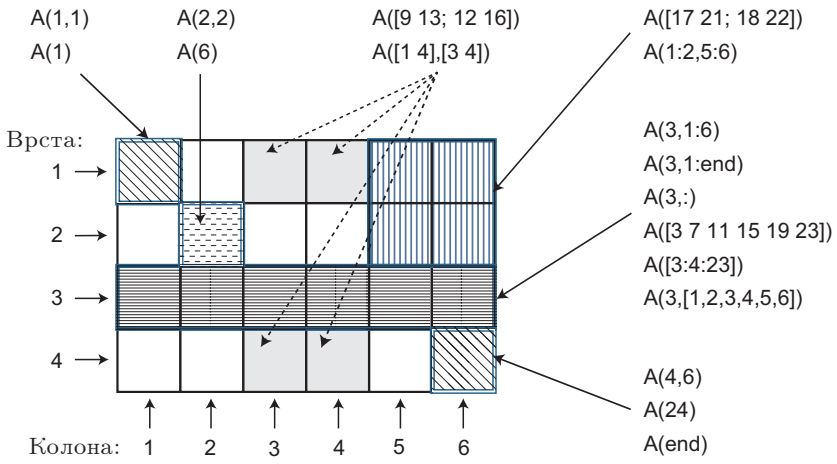
```
>> vekt_vrsta = 1:2:6;
>> vekt_kolona = vekt_vrsta'
vekt_kolona =
     1
     3
     5
```

```
>> x(end)
ans =
     8
>> x(end-4:end-1)
ans =
     4     5     6     7
```

Претходна техника важи и у случају матрица:

```
>> A = [1 2 3; 4 5 6; 7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9
>> A(end,end)
ans =
     9
<< A(end-1,:)
ans =
     4     5     6
```

На примеру једне матрице типа 4×6 на слици 3.2 су приказани различити начини за приступ елементима и групама елемената матрице.



Слика 3.2: Пример индексирања елемената матрице

3.2.12 Елементне операције над векторима и матрицама

Често је потребно да се уместо матричних аритметичких операција множења, дељења, степеновања, изврше те операције елемент по елемент, тј. над одговорајућим елементима матрица. Наравно, то је једино могуће код *матрица исте величине*, и такве операције се називају *елементне операције*. Оператори за рад елемент по елемент се добијају додавањем тачке (.) (тзв. dot оператор) испред одговарајућег матричног оператора. Сваки од

3.2.15 Вектори и матрице као аргументи функција

У Матлабу се вектор или матрица могу проследити као аргументи функције, при чему ће се функција извршити на сваком елементу. То значи да ће резултат бити промењива исте величине као улазни аргумент. На пример, за дати вектор **x** применом функције `sin()` могу се добити вредности $y = sin\, x$ за сваки елемент x вектора **x**. Резултат ће бити вектор исте дужине као улазни вектор:

```
>> x = -2:2
x =
    -2    -1     0     1     2
>> y = sin(x)
y =
   -0.9093   -0.8415         0    0.8415    0.9093
```

За матрицу, резултујућа матрица ће бити истог типа као улазна матрица. На пример, функција `sign()` ће одредити знак сваког елемента матрице:

```
>> mat_A = [0 2 -1; -3 0 2]
mat_A =
     0     2    -1
    -3     0     2
>> sign(mat_A)
ans =
     0     1    -1
    -1     0     1
```

Функцијама као што су `sin()` и `sign()` могу се проследити као аргументи и скалари и низови (вектори или матрице). Међутим, постоји известан број функција које су написане специјално за рад са векторима или са колонама матрица. У те функције спадају, на пример, функције `min()`, `max()`, `prod()`, `cumsum()` и `cumprod()`.

На пример, нека су дефинисане две векторске промењиве, $\mathbf{x}_1$ и $\mathbf{x}_2$:

```
>> x_1 = 1:5;
>> x_2 = [3 5 8 2];
```

Функције `min()` и `max()` као резултат враћају минималну и максималну вредност садржану у вектору, следствено.

```
>> min(x_1)
ans =
     1
>> max(x_2)
ans =
     8
```

Функција `sum()` израчунава суму свих елемената вектора. На пример, за вектор $\mathbf{x}_1$ се добија $1 + 2 + 3 + 4 + 5 = 15$:

```
>> sum(x_1)
ans =
    15
```

ПОГЛАВЉЕ 4

Програмирање у Матлабу

До сада је употреба Матлаба била интерактивна, у смислу да су све наредбе уношене у командни прозор, и да су се оне извршавале након притиска на тастер **Enter**. То је задовољавајући начин рада уколико су циљ једноставна израчунавања. Међутим, у многим случајевима неопходан је низ корака пре него што се добије крајњи резултат. Уколико треба извршити низ наредби међу којима постоји одређена зависност, онда је непрактичност примене само командног прозора очита. Када се притисне тастер **Enter**, извршава се само последња унета наредба, а све што је извршено раније остаје непромењено. Уколико је потребна корекција у наредби која је претходно извршена и резултат те наредбе се користи у наредбама које следе, све наредбе се морају унети и извршити поново. Тада је ефикасније да се неопходне наредбе групишу, чиме се добија један *рачунарски програм*. У ту сврху може се креирати једна датотека са листом наредби, снимити и потом покренути њено извршавање. Када се датотека покрене, наредбе садржане у њој се извршавају по редоследу у коме су написане. Уколико је потребно, наредбе у датотеци се могу поправљати или мењати, и датотека може бити снимљена и покренута поново. Датотеке које се користе у ове сврхе, и на овај начин, називају се *скриптови*. Овај концепт биће илустрован кроз низ примера у наставку излагања.

4.1 Алгоритми

Пре писања било каквог програма корисно је размотрити кораке који су неопходни да би се дошло до решења. У модуларном прилазу програмирању решење проблема се дели у посебне кораке, а потом се сваки корак даље анализира, рашчлањује, све док резултујући кораци не буду довољно мали да би представљали изводљив задатак. Овај низ корака и скуп правила дефинисаних у циљу решавања одређеног проблема назива се *алгоритам*. Као једноставан пример, биће размотрен проблем израчунавања површине

круга. Најпре треба одредити које информације су неопходне да би се тај проблем решио, а у овом случају то је податак о полупречнику круга. Даље, за дати полупречник круга неопходно је израчунати његову површину. И на крају, када се површина израчуна, треба је приказати на неки начин. Основни алгоритам се, према томе, може представити са три корака:

1. добијање улазног податка – полупречник круга,
2. израчунавање резултата – површина,
3. приказивање резултата.

Чак и код овог једноставног алгоритма, могуће је даље редефинисати сваки од наведених корака. Када се пише програм у циљу имплементације овог алгоритма, кораци би требало да буду следећи:

- Одакле долази улазни податак? Два су могућа извора: из неке спољашње датотеке, или од стране корисника (особе која покреће програм) који уноси број куцајући са тастатуре. За сваки систем један од ова два могућа избора биће стандардни улазни уређај (дакле, он је подразумеван, ако није специфицирано другачије). Уколико се кориснику постави задатак да унесе полупречник, он такође мора да зна у којим јединицима то да уради. Према томе, програм треба да каже кориснику ту информацију. Може се закључити да се улазни корак састоји од два међукорака: упит и информација кориснику да унесе полупречник, и када он то учини, потребно је да се тај податак учита у програм.
- Да би се израчунала површина, неопходно је знати формулу. У овом једноставном примеру, користи се позната формула: површина круга P једнака је $P = r^2\pi$, где је r полупречник круга. Може се уочити, такође, да је неопходна и константа π у програму.
- Где треба да се проследи излаз (резултат) из програма? Две су могућности: у спољашњу датотеку или на екран. Када се приказује излаз из програма, он треба да буде информативан што је више могуће. Другим речима, уместо да се само штампа вредност површине (број), пожељно је да се одштампа нека лепо форматирана реченица. Такође, да би одговор био још јаснији, пожељно је одштампати и вредност улаза у програм. На пример, излаз би могао бити реченица: ,,Површина круга чији је полупречик $1\,\mathrm{cm}$ износи $3.1415\,\mathrm{cm}^2$".

4.2 Матлаб скрипт

Пошто се проблем проанализира и дефинише алгоритам за његово решавање, решење проблема се пише у одређеном програмском језику. Рачунарски програм је низ инструкција у датом језику, које извршавају неки задатак. Програмски језици високог нивоа имају наредбе и функције, као што су на пример ,,штампај ово" или ,,ако је $x < 5$ уради то и то". Рачунар, међутим, може да извршава само наредбе писане у његовом машинском

Ово резултује да датотека ,,testdat.dat" сада садржи следеће податке:

```
0.8147    0.1270    0.6324
0.9058    0.9134    0.0975
0.2785    0.9649    0.9572
0.5469    0.1576    0.4854
0.9575    0.9706    0.8003
```

4.4.3 Учитавање података из датотеке

Учитавање података из датотеке се постиже коришћењем наредбе `load`. Када се датотека једном креира (као у претходно наведеном случају), њен садржај се може учитати у неку матричну променљиву. Уколико је то датотека са нумеричким подацима, функција `load()` ће учитати податке из датотеке ,,ImeDatoteke.ext" (на пример, екстензија може бити .dat) и креирати матрицу са истим именом као што је име датотеке. На пример, ако се искористи датотека раније креирана, ,,testdat.dat", следећим наредбама ће се учитати подаци из ње, и сместити резултат у матричну променљиву која се назива *testdat*:

```
>> clear
>> load testdat.dat
>> who
Your variables are:
testdat
>> testdat
testdat =
    0.8147    0.1270    0.6324
    0.9058    0.9134    0.0975
    0.2785    0.9649    0.9572
    0.5469    0.1576    0.4854
    0.9575    0.9706    0.8003
```

Функција `load()` ради једино ако постоји исти број вредности у свакој линији тако да подаци могу бити смештени у матрицу, и наредба `save` једино уписује податке из матрице у датотеку. Уколико то није случај, морају се користити улазно/излазне наредбе нижег нивоа, које нису предмет ових излагања.

4.5 Функције

Матлаб поседује богату библиотеку уграђених функција, и оне се могу користити у математичким изразима једноставно – куцањем њихових имена и вредности њихових аргумената, као што су примери `sin(x)`, `sqrt(x)`, итд. Најчешће у рачунарским програмима постоји потреба да се израчунавају вредности функција које нису унапред дефинисане у Матлабу. Када је израз функције једноставан, и ако је потребно израчунати је само једном, та функција може бити део програма. Међутим, уколико је неку функцију потребно израчунавати много пута, и при томе и са различитим вредности-

Улазни и излазни аргументи функције

Улазни и излазни аргументи се користе за пренос података *у функцију* и *из функције*, следствено. Обично, функција има најмање један улазни аргумент, мада је могуће дефинисати и функцију која нема улазне аргументе. Уколико их има више, улазни аргументи су раздвојени запетом. Програмски кôд који извршава израчунавања у функцији се пише уз коришћење управо тих улазних аргумената, уз претпоставку да су им додељене нумеричке вредности. То значи да се математички изрази у функцијама морају писати у складу са димензијама улазних аргумената, који могу бити скалари, вектори или матрице. Тренутне вредности аргумената се додељују приликом позива функције.

Функција може имати један или више излазних аргумената, или ниједан. У случају једног излазног аргумента он се може користити без заграде.

Да би се функција могла користити, у телу функције мора постојати наредба која додељује вредности излазним променљивим, тј. излазним аргументима. Ако функција нема излазни аргумент, оператор доделе у дефиниционој линији може се изоставити. На пример, функција без излазног аргумента може бити нека функција која црта дијаграме, штампа или уписује податке у датотеку итд.

Уобичајено је да се сви улази у функцијску датотеку, као и излази из ње, преносе преко улазних и излазних аргумената. Поред тога, међутим, све улазно-излазне карактеристике скрипт датотеке се могу користити и у функцијској датотеци. То значи да ће се свака променљива којој је додељена вредност у телу функције приказати на екрану командног прозора уколико се на крају наредбе доделе не налази знак `;`. Такође, функција `input()` се може користи за интерактивни унос података, а функције `disp()`, `fprintf()` и `plot()` се могу користити за приказ информација на екрану, снимање у датотеку или цртање дијаграма, баш као и у скрипт датотекама.

На пример, нека је потребно да се напише функција која рачуна обим и површину круга за задати полупречник. То се може урадити на следећи начин:

```
function [Obim, Povrsina] = krug_op(radijus)
% Funkcija vraca obim i povrshinu kruga.
% Ulazni argument: radijus (skalar)
% Izlazni argumenti: Obim i Povrsina (skalari)

Obim = 2*radijus*pi;
Povrsina = radijus^2*pi;
end
```

Дакле, функција има један улазни и два излазна параметра, и сви они представљају скаларе. Пример позива функције би био:

```
>> [O,P] = krug_op(3)
O =
   18.8496
P =
   28.2743
```

Важно је водити рачуна о томе да се функција позове са одговарајућим бројем аргумената, и да су они одговарајућег типа. На пример, позив

```
>> [O,P] = krug_op
Error using krug_op (line 6)
Not enough input arguments.
```

резултује грешком, због изостављања аргумента. Са друге стране, уколико се функција позове на следећи начин:

```
>> krug_op(3)
ans =
   18.8496
```

очито је да је информација о другом излазном аргументу изгубљена. У позиву функције не постоји променљива у коју би се резултат сместио и сачувао.

Претходни пример се може модификовати тако да функција рачуна обим и површину за низ улазних вредности, тј. за вектор улазних података, тако што се у изразу за израчунавање површине употреби елементни оператор множења.

```
function [Obim, Povrsina] = krug_opv(radijus)
% Funkcija vraca obim i povrshinu kruga.
% Ulazni argument: radijus (vektor)
% Izlazni argumenti: Obim i Povrsina (vektori)

Obim = 2*radijus*pi;
Povrsina = radijus.^2*pi;
end
```

Извршавањем наредби:

```
>> x = 1:5;
>> [O,P] = krug_opv(x)
O =
    6.2832   12.5664   18.8496   25.1327   31.4159
P =
    3.1416   12.5664   28.2743   50.2655   78.5398
```

резултат израчунавања функције су два вектора, истих димензија као и улазни вектор.

Запажање 4.1. *Када се позива нека функција, важно је знати какав облик ћеног излаза се очекује, тј. да ли је у питању скалар, вектор или матрица; наравно, важно је знати и број излазних величина.*

H1 линија и коментари

H1 линија и текст помоћи су линије коментара (линије које почињу са знаком процента), а које следе након дефиниционе линије функције. H1 линија је прва линија коментара и обично садржи име и кратак опис функције. Када корисник у командном прозору откуца наредбу `lookfor` *neka_rech*, Матлаб тражи реч *neka_rech* у H1 линијама свих функција, и ако је нађе,

3. из функције на било ком нижем нивоу (C може позивати B или D, али не може позивати и E).

Такође, подфункција се може позивати из било које угнеждене функције унутар исте m-датотеке.

4.5.8 Анонимне функције

Анонимне функције омогућавају креирање једноставних функција без потребе за креирањем функцијске датотеке за њих. Оне се могу креирати или у командној линији Матлаба, или унутар друге функције или скрипта. Синтакса за дефинисање анонимне функције је следећа:

$$fpok = \texttt{@(}ListaArgumenata\texttt{)}\ TeloFunkcije$$

где је *ListaArgumenata* запетама раздвојена листа улазних аргумената која се прослеђује функцији, а *TeloFunkcije* је било који важећи израз у Матлабу, и он представља тело функције. Овом синтаксом креира се показивач на функцију *fpok*, који представља тип података `function_handle`, који омогућава позивање функције. У наставку ће бити разматроне неколико једноставних примера.

Матлаб има уграђену функцију за рачунање природног логаритма `log()`, али се може креирати анонимна функција која то израчунава, и за њен показивач користити име `ln`, што је честа математичка ознака:

```
>> ln = @(x) log(x)
ln =
    @(x)log(x)
```

Сада се та анонимна функција може користити као било која друга функција, на пример:

```
>> ln(10)
ans =
    2.3026
```

У овом случају, `ln` је име промењиве која се налази у радном простору Матлаба:

```
>> whos ln
  Name      Size            Bytes  Class              Attributes
  ln        1x1                32  function_handle
```

Појам класе `function_handle` је веома сличан појму показивача у програмском језику C, мада је он овде уопштенији.

За израчунавање површине круга својевремено је коришћен скрипт, али се то може урадити и помоћу анонимне функције на следећи начин.

```
>> KrugPovrsina = @(radijus) pi*radijus.^2;
```

У телу функције је коришћен елементни оператор степеновања, што омогућава да се као улазни параметар проследи вектор. Примери позива су:

```
>> KrugPovrsina(4)
ans =
   50.2655
```

ПОГЛАВЉЕ 5

Симболичка математика

Симболичка математика подразумева бављење математиком над симболима, а не над бројевима, као на пример, $a+a=2a$. Симболичке математичке функције део су Матлабовог Симболичког пакета, који користи посебан и нов производ MuPAD. Пре његове појаве и његовог развијања Матлаб је кроз стотине развијених функција користио Maple за подршку симболичким израчунавањима. Данас, функционалност MuPAD језика, заједно са укљученим библиотекама је значајно испред првобитног симболичког математичког пакета. Имена и функционалност многих функција доступних у овом пакету су идентични функцијама које постоје у Матлабу, али је разлика управо у томе што функције у симболичком пакету раде са симболичким објектима, за разлику од оних у Матлабу, које раде са нумеричким вредностима.

Основни елемент за симболичке операције је симболички објект. Симболички објекти се састоје од променљивих и бројева који када се користе у математичким изразима говоре Матлабу да изврши операције симболички. Уколико је потребно симболички изрази се могу користити и у нумеричким операцијама.

5.1 Симболичке променљиве и изрази

Симболички објект може бити променљива (без претходно додељене нумеричке вредности), број, или израз састављен од нумеричких променљивих и бројева. За креирање симболичког објекта користи се функција `sym()` и/или наредба `syms`. Ако је улазни аргумент функције `sym()` стринг, резултат је симболички број или променљива. За случај да је улазни аргумент нумерички скалар или матрица, резултат је симболичка презентација датих нумеричких вредности. На пример, наредба `x=sym('x')` креира симболичку променљиву под именом x и смешта резултат у x. Наредба `y=sym('1/7')` или `y=sym(1/7)` креира један симболички број, чиме се избегава апрокси-

```
>> [X Y Z] = solve(4*x-2*y+z==7,'x+y+5*z==10','-2*x+3*y-z==2')
X =
124/41

Y =
121/41

Z =
33/41
```

5.1.4 Симболичка математичка анализа

Диференцирање

Функција `diff()` се користи за симболичко одређивање извода функција. Иако ова функција има исто име као функција која се користи за нумеричко израчунавање коначних разлика, Матлаб препознаје коју функцију ће користити у конкретном случају. Основна синтакса функције је `diff(`F`)`, при чему се враћа извод израза функције F у односу на подразумевану независну променљиву.

```
>> syms x f
>> f = x^3 + 2*x^2 - 4*x +3 ;
>> diff(f) % ili direktno diff(x^3+2*x^2-4*x+3)
ans =
3*x^2 + 4*x - 4
```

Уколико функција зависи од више променљивих, онда функција рачуна парцијални извод реда n по изабраној променљивој var, и тада је синтакса позива облика `diff(`F`,`var`,`n`)`.

```
>> syms a x, f = sin(a*x^2);
>> dfdx = diff(f)
dfdx =
2*a*x*cos(a*x^2)
>> dfda = diff(f,a)
dfda =
x^2*cos(a*x^2)
>> d2fdx2 = diff(f,2)
d2fdx2 =
2*a*cos(a*x^2) - 4*a^2*x^2*sin(a*x^2)
```

Интеграљење

Функција `int()` се позивом облика `int(`$expr$`,`var`)` користи за одређивање неодређеног интеграла симболичког израза $expr$ у односу на симболичку променљиву var. Функција покушава да нађе симболички израз I тако да је `diff(`I`,`var`)`$=expr$. Параметар var је опциони, и ако се не наведе, функција враћа интеграл израза $expr$ у односу на подразумевану независну променљиву. Уколико интеграл не постоји у затвореном облику, или Мат-

лаб не може да га израчуна иако он постоји, тада се наредба враћа неизвршена (дакле, онако како је унета).

На пример, неодређени интеграл функције $f(x) = 3x^2 - 1$ може се наћи на следећи начин:

```
>> syms x
>> int(3*x^2-1)
ans =
x^3 - x
```

Са друге стране, одређени интеграл ове функције у границама $x = 2$ до $x = 4$ може се добити са:

```
>> int(3*x^2-1,2,4)
ans =
54
```

Граничне вредности

Граничне вредности функције могу се наћи функцијом `limit()`, чија је општа синтакса `limit(`*expr*`,`*x*`,`*a*`)`, и којом се рачуна гранична вредности израза *expr* када променљива x тежи променљивој a, тј. $\lim\limits_{x \to a} expr$. Параметар x се може изоставити, и онда се користи подразумевана променљива. На пример, граничне вредности:

$$\lim_{x \to 2} \frac{x-2}{x^2-4} = \frac{1}{4}, \quad \lim_{h \to 0} \frac{cos(x+h) - cos(x)}{h} = -\sin(x)$$

се добијају на следећи начин:

```
>> syms h x
>> limit((x-2)/(x^2-4),2)
ans =
1/4
>> limit((cos(x+h)-cos(x))/h,h,0)
ans =
-sin(x)
```

Уколико се параметар a изостави, гранична вредност се одређује када променљива тежи нули.

5.2 Решавање диференцијалних једначина

Функција `dsolve()` се користи за решавање обичних диференцијалних једначина. Различити облици позива функције одређени су димензионалношћу проблема који се решава, тј. у зависности да ли је у питању скаларна диференцијална једначина или систем једначина, као и постојањем или непостојањем почетних и граничних услова. Функција `dsolve()` подразумева да је унапред дефинисана независна променљива t а не x, као код других симболичких функција.

ПОГЛАВЉЕ 6

Оператори и контрола тока програма

Програми разматрани у претходним поглављима били су структурно једноставни, јер су се наредбе извршавале једна за другом, онако како су писане, без икаквог гранања и понављања. Међутим, већина програмерских проблема није тако једноставна. Чињеница је у ствари, да велика моћ програмских језика долази из њихове могућности да дају инструкције рачунару да изврши исти задатак више пута, са понављањем, или да изврши различите задатке ако се одређени параметри мењају или су задовољени одређени услови. У вишим програмским језицима ово се постиже уз помоћ наредби за контролу тока програма. Наредбе за контролу тока програма се могу поделити у две основне групе: наредбе за условно гранање, и наредбе петљи. Условно гранање представља могућност одлуке да ли се одређени кôд извршава или не, а у зависности од вредности неког израза. Петље, или другим речима итерације, представљају способност да се изврши скуп истих операција више пута, док се специјални захтев не задовољи. У наредбама одлучивања, гранања, и уопште у контроли тока програма, једну од важних улога играју логички и релациони оператори, и о њима ће такође бити речи у даљем тексту.

6.1 Релациони оператори

Релациони оператори (или *оператори поређења*) су оператори односа између двеју променљивих или израза, а као резултат дају логички тип података. Они дозвољавају било коју варијанту операнада, што значи да се могу примењивати између скалара, вектора и матрица, али и између скалара и матрице (вектора). Такође, релациони оператори се могу примењивати и на комплексне бројеве, и у том случају се упоређују само њихови реални делови.

Дакле, резултат поређења скалара применом релационих оператора је или 0 (ако поређење није тачно) или 1 (ако је поређење тачно), и резултат

порука о грешкама код израза код којих би се у неким околностима то могло појавити (на пример, дељење нулом).

Логички оператори се могу користити са свим нумеричким типовима података, узимајући у обзир да се све нумеричке вредности различите од нуле третирају као логичка вредност `true`, а вредност нула се третира као логичка вредност `false`.

6.3 Условно гранање

Условно гранање је најосновнији елемент за контролу тока у било ком програмском језику. Он омогућава програму да доноси одлуке и д одлучује да ли ће се извршити или неће низ наредби, а на основу вредности неког израза. Како се вредност овог израза може разликовати са различитим извршавањима програма, поменута карактеристика омогућава да програм реагује динамички на различите податке. У Матлабу се условно извршавање програма постиже кључним речима `if`, `else` и `elseif`.

6.3.1 If услов

Облик једне `if` наредбе је веома једноставан. Након кључне речи `if` следи један логички израз, који се другачије и назива *услов* if исказа. Уколико *uslov* има вредност `true` (1), извршава се низ наредби које следе иза исказа `if`, све до исказа `end` (*grupa naredbi* из синтаксе). У супротном, ако је логичка вредност израза *uslov* `false`, програм прескаче групу наредби између `if` и `end`, и извршавање програма се наставља првом наредбом иза кључне речи `end`.

```
if uslov
   grupa naredbi
end
```

Израз *uslov* након кључне речи `if` може бити сложен израз који се састоји од релационих и/или логичких оператора. Што се тиче димензија, може бити скалар, вектор или матрица.

Структура `if-else` омогућава извршавање једне од две могуће групе наредби, а у зависности од истинитосне вредности услова `if` исказа и њен општи облик је (слика 6.1):

```
if uslov
   grupa naredbi 1
else
   grupa naredbi 2
end
```

Ако израз *uslov* након кључне речи `if` има вредност `true`, извршава се низ наредби означен са *grupa naredbi 1*, који се налази између кључних речи `if` и `else`, а затим се прескачу остале наредбе до исказа `end`. Уколико је вредност израза *uslov* `false`, програм прескаче наредбе до исказа `else`, а затим извршава низ наредби означен са *grupa naredbi 2* између `else` и `end`.

дефинише када петља почиње. Насупрот томе, код `while` петље број пролаза није познат унапред и процес понављања петље се наставља све док се специфицирани услов не задовољи. Обе врсте петље могу се прекинути у било ком тренутку наредбом `break`.

6.4.1 For петља

`For` петља извршава наредбу или низ наредби унапред одређени број пута. Структура `for` петље је једноставна и њен општи облик је:

```
for indeks_petlje = vrednosti
    grupa naredbi
end
```

Прва линија идентификује петљу и дефинише променљиву *indeks_petlje*, која се назива *променљива петље* или *индекс петље*, а представља број чија се вредност мења током проласка кроз петљу. Израз *vrednosti*, који може бити вектор или матрица, представља *скуп вредности* које променљива *indeks_petlje* узима док се петља извршава. Суштински, број елемената у вектору вредности (или број колона, уколико је у питању матрица) одређује број пролазака кроз петљу. Наредба или група наредби која следи, означена са *grupa naredbi* , назива се *тело петље*, и оно се извршава при сваком пролазу. Кључна реч `end` идентификује крај `for` петље. Петља се извршава једанпут за сваку вредност променљиве *indeks_petlje*. У примеру:

```
for k = [1 3 7 2]
    x = k^2;
    fprintf(' %d', x)
end
fprintf('\n')
```

променљива k представља индекс петље, а вектор $\begin{bmatrix}1 & 3 & 7 & 2\end{bmatrix}$ представља скуп вредности. Током првог пролаза кроз петљу променљива k има вредност 1, што је прва вредност из вектора вредности. У телу петље се рачуна квадрат променљиве k и додељује променљивој x, а потом се вредност променљиве x штампа на екрану. Током следећег пролаза вредност k се мења на вредност 3, што је друга вредност у вектору вредности, и извршава се поново тело петље. При сваком проласку кроз петљу вредност променљиве k се мења и додељују јој се следећа вредност из променљиве вредности. Извршавањем скрипта добија се:

```
 1 9 49 4
```

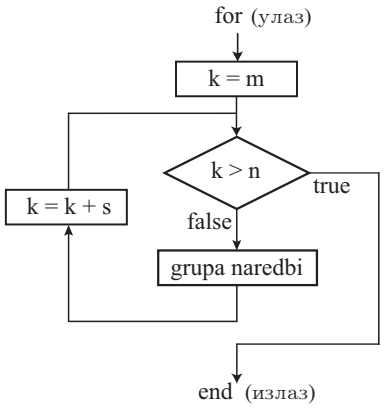
У претходном примеру индекс петље је узимао вредности из вектора, између чијих елемената не постоји никаква уочљива зависност. Међутим, најчешће је примена `for` петље у следећем облику (слика 6.3):

```
for k = m:s:n
    grupa naredbi
end
```

Дакле, вектор вредности овде представља вектор са особином да је разлика између два суседна елемента константна и једнака s. Израз $m{:}s{:}n$ додељује

индексу петље k почетну вредност m која се током сваког пролаза повећава за s, па се величина s назива *корак петље*. Петља се завршава када вредност индекса k достигне вредност n. У том случају, програм се не враћа више на наредбу `for`, већ наставља са извршавањем прве следеће наредбе иза кључне речи `end`. На пример, ако је $k = 1 : 3 : 10$, биће 4 пролаза кроз петљу, а вредности индекса k у пролазима кроз петљу су 1, 4, 7 и 10, следствено.

Корак петље k може бити и негативан. Тако, на пример, $k = 20 : -5 : 10$ резултује са 3 пролаза у којима је $k = 20, 15, 10$. Уколико није задат корак петље, подразумева се да је он једнак 1. Ако је $m = n$, петља ће се извршити само једном. У случају да је $m > n$ и $s > 0$, или пак $m < n$ и $s < 0$, петља се неће извршити ниједном. Специјално, ако су вредности k, s и n такве да k не може бити једнако n, уколико је корак s позитиван, последњи пролаз ће се извршити када k има највећу вредност која је мања од n. На пример, $k = 1 : 2 : 6$ даје 3 пролаза у којима је $k = 1, 3, 5$. Ако s има негативну вредност, последњи пролаз ће се извршити када k има најмању вредност која је већа од n.



Слика 6.3: Дијаграм тока `for` петље

И треба напоменути – свакој наредби `for` у програму мора бити додељена одговарајућа кључна реч `end`. Када се петља заврши, индекс петље има последњу додељену му вредност.

Матрица као индекс петље

Индекс петље може бити и матрица, за разлику од претходних примера где је то био вектор. На пример, нека је дата матрица A типа $m \times n$. Наредбе

```
for k = A
    grupa naredbi
end
```

додељује индексу k вектор $A(:, i)$, где i представља текући број итерације петље, односно пролаза петље. У првом пролазу, k је једнако $A(:, 1)$, у другом $k = A(:, 2)$, у i-том $k = A(:, i)$ и све до $k = A(:, n)$. Дакле, број пролаза је једнак броју колона матрице A, а у сваком пролазу индекс k представља одговарајућу колону матрице A. Један такав пример је:

```
>> A = [1 2 3; 4 5 6]
A =
     1     2     3
     4     5     6
>> for k = A, y = 2*k, end
y =
     2
     8
```

ПОГЛАВЉЕ 8

Системи линеарних једначина

На потребу изучавања система линеарних једначина, као најједноставнијег типа алгебарских једначина, упућују многи проблеми теорије и праксе. У наставку се у сажетом приказу дају теоријске основе, као и основа њиховог решавања у Матлабу.

8.1 Основни појмови и број решења

Дефиниција 8.1. *Израз облика* $a_1x_1 + a_2x_2 \ldots + a_nx_n$, *за* $n \geq 1$ *и где су* a_1, a_2, ..., a_n *реални бројеви а* x_1, x_2, ..., x_n *променљиве, назива се* **линеарна комбинација променљивих** x_1, x_2, ..., x_n.

Дефиниција 8.2. *Једначина облика:*

$$a_1x_1 + a_2x_2 \ldots + a_nx_n = b, \tag{8.1}$$

где $a_1, a_2, \ldots, a_n \in \mathcal{R}$ *и* $b \in \mathcal{R}$ *назива се* **линеарна једначина** *по променљивама* $x_1, x_2, \ldots, x_n$. *Уколико је* $b = 0$, *линеарна једначина је* **хомогена**, *у супротном, за* $b \neq 0$, *она је* **нехомогена**. *Скалари* $a_1, a_2, \ldots, a_n$ *се називају* **коефицијенти једначине**, *а коефицијент* b *је* **слободни члан** *једначине* (8.1).

Дефиниција 8.3. *Вектор* $\mathbf{r} = \begin{bmatrix} r_1 & r_2 & \ldots & r_n \end{bmatrix}^T \in \mathcal{R}^n$ *је* **партикуларно решење** *(или краће:* **решење***) линеарне једначине* (8.1) *ако важи*

$$a_1r_1 + a_2r_2 \ldots + a_nr_n = b. \tag{8.2}$$

Опште решење *линеарне једначине* (8.1) *је скуп свих њених партикуларних решења.*

Дефиниција 8.4. *Скуп од* m $(m > 1)$ *линеарних једначина по променљивим*

у други, *еквивалентан* систем једначина, који је једноставнији за решавање. Два система су еквивалентна ако поседују исти скуп решења. Три операције над једначинама које не мењају њихово решење, тзв. елементарне операције, су:

- замена места било којим двема једначинама
- множење било које једначине скаларом различитим од нуле
- множење једне једначине скаларом различитим од нуле и потом њено додавање било којој другој једначини.

Недостатак директних метода је што се код система високог реда може јавити проблем тачности услед акумулације грешака заокруживања. Итеративне, или индиректне методе, почињу од неке претпоставке решења, и потом континуирано редефинишу решење све док се не задовољи одређени критеријум конвергенције. Ове методе су погодне за системе са великим бројем једначина, али се јавља проблем конвергенције тих метода.

У Матлабу се за решавање система линеарних једначина (8.4) користи метода левог дељења. Ова метода се заснива на методи Гаусове елиминације за решавање система линеарних алгебарских једначина.

Код решавања система линеарних једначина (8.4) могу се разликовати три случаја:

- број једначина m је једнак броју непознатих, тако да матрица коефицијената представља квадратну матрицу
- број једначина m је мањи од броја непознатих n $(m < n)$
- број једначина m је већи од броја непознатих n $(m > n)$.

Све три варијанте се разматрају у наставку.

8.2.1 Систем једначина са $m = n$

Ако је $m = n$, тада је матрица A квадратна матрица, тако да се решење може наћи као

$$\mathbf{x} = A^{-1}b, \tag{8.7}$$

уколико матрица A није сингуларна. У Матлабу, то се може израчунати применом функције `inv()`, тј. `inv(`A`)` или A`^(-1)`. На пример:

```
>> A = [1 2; 3 4]; b = [-1; -1];
>> x = A^(-1)*b
x =
    1.0000
   -1.0000
>> x = inv(A)*b
x =
    1.0000
   -1.0000
```

ПОГЛАВЉЕ 9

Апроксимација и интерполација

9.1 Општи проблем апроксимације

Нека је функција $f : \mathcal{X} \longrightarrow \mathcal{R}$, $\mathcal{X} \subseteq \mathcal{R}$ функција реалне промењиве. Проблем апроксимације функције $f(\cdot)$ своди се на одређивање неке друге функције

$$\phi(x) = \phi(x; a_0, a_1, \ldots, a_m)$$

која зависи од $m+1$ параметара $a_0, a_1, \ldots, a_m$, тако да важи $\phi(x) \approx f(x)$. Функција $\phi(\cdot)$ се назива *апроксимациона функција*. У пракси се проблем апроксимације може јавити из два разлога. Први је што аналитички облик функције $f(\cdot)$, иако је познат, може бити веома компликован, па је израчунавање њених вредности сложено. Са друге стране, аналитички облик функције није познат, већ су познате само вредности функције на неком скупу тачака. То је чест случај у многим научним и инжењерским применама, када се подаци прикупљају у експериментима при разним мерењима различитих физичких величина, на пример код експерименталних одређивања статичких карактеристика, као што је случај код криве сила-издужење у поступку одређивања крутости опруге, и слично.

Један од основних проблема апроксимације је како одредити, односно изабрати функцију апроксимације. У принципу, све апроксимационе функције се могу поделити на линеарне и нелинеарне. Општи облик линеарне апроксимационе функције је

$$\phi(x) = a_0\phi_0(x) + a_1\phi_1(x) + \ldots + a_m\phi_m(x), \tag{9.1}$$

при чему систем функција $\phi_k(\cdot)$ задовољава одређене особине. Овде се појам линеарне функције односи на линеарност по параметрима a_i, $i = 0, 1, \ldots, m$, а добра страна овог облика апроксимације је што се одређивање параметара своди на системе линеарних једначина.

9.4.3 Функције за интерполацију у Матлабу

Интерполациони кубни сплајн се може одредити коришћењем уграђене функције Матлаба, `spline()`, чија је општа синтакса облика:

$$\mathbf{y}_i = \texttt{spline}(\mathbf{x},\mathbf{y},\mathbf{x}_i)$$

где су $\mathbf{x}$ и $\mathbf{y}$ вектори који садрже тачке интерполације, а $\mathbf{y}_i$ вектор који садржи резултате интерполације израчунате у тачкама вектора $\mathbf{x}_i$. Подразумевани гранични услов је тзв. *not-a-knot* услов. У случају да вектор $\mathbf{y}$ садржи две вредности више него што има вектор $\mathbf{x}$, тада се прва и последња вредност у вектору $\mathbf{y}$ користе као изводи у крајњим тачкама. На тај начин ова опција омогућава да се имплементира и *clamped-end* услов, тј. да се добије *потпуни* сплајн.

Уграђена функција `interp1()` омогућава да се имплементира неколико типова у деловима једнодимензијских интерполација. Њена општа синтакса је:

$$\mathbf{y}_i = \texttt{interp1}(\mathbf{x},\mathbf{y},\mathbf{x}_i,method)$$

где су $\mathbf{x}$ и $\mathbf{y}$ вектори који садрже вредности које се интерполирају, $\mathbf{y}_i$ је вектор који садржи резултате интерполације добијене у тачкама вектора $\mathbf{x}_i$ применом методе дефинисане у параметру *method*. Доступне методе су:

- `'nearest'`: интерполираној тачки се додељује вредност њој најближе тачке. Према томе, представља интерполацију полиномом нултог реда.
- `'linear'`: линеарна интерполација. Ова метода користи праве линије за повезивање тачака.
- `'spline'`: интерполација у деловима кубним сплајном. Ова метода је идентична функцији `spline()`.
- `'pchip'` и `'cubic'`: у деловима кубна Ермитова интерполација.

Уколико се аргумент *method* у позиву функције изостави, подразумевана метода је метода линеарне интерполације.

Пример 9.1. *На основу произвољно одабраних тачака интерполирати функцију* $f(x) = (x^2 - 3x + 5)e^{-5x}sinx$ *на интервалу* $[0, 1]$.

Решење: Пример решења дат је следећом функцијом, са краћим објашњењима и коментарима о појединим наредбама. Графички приказ резултата дат је на слици 9.3.

```
function inter_primer
% Primer interpolacije funkcije
% y = f(x) = (x.^2 - 3*x + 5).*exp(-5*x).*sin(x);
% razlichitim metodama interpolacije

% generisanje chvorova interpolacije
x = 0:0.1:1;
y = (x.^2 - 3*x + 5).*exp(-5*x).*sin(x);
```

ПОГЛАВЉЕ 10

Нелинеарне алгебарске једначине

10.1 Нелинеарна једначина једне променљиве

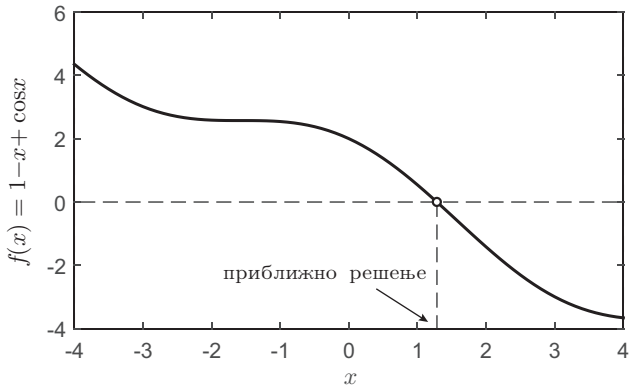
У многим проблемима у науци и инжењерству неопходно је одредити решење неке нелинеарне једначине облика:

$$f(x) = 0. \tag{10.1}$$

Графички, решење или корен једначине представља тачку пресека функције $f(\cdot)$ и $x-$осе. Према томе, у зависности од функције $f(\cdot)$, једначина (10.1) може имати јединствено решење, више решења или неће имати решење. Корен једначине се некад може одредити аналитички и онда је то тачно решење. Међутим, у већини ситуација то није могуће, па се корени једначине морају наћи применом неке нумеричке процедуре. Један такав пример је једначина облика $1 - x + cosx = 0$. Слика 10.1 приказује да ова једначина има само једно решење, које се може апроксимирати са жељеном тачношћу уз помоћ неке нумеричке методе.

Запажање 10.1. *Треба обратити пажњу на терминологију и употребу појмова* **решење**, **корен** *и* **нула**. *Корен представља решење једначине $f(x) = 0$, а то решење у ствари представља нулу функције $f(\cdot)$.*

Нумеричке методе за решавање нелинеарних једначина се могу поделити у две главне категорије: *методе на затвореном интервалу* и *методе на отвореном интервалу*, или краће – *отворене методе*. Методе на затвореном интервалу захтевају да се идентификује, тј. одреди, почетни интервал који садржи корен једначине. Према слици 10.2а, која приказује основни принцип метода на затвореном интервалу, то значи да треба одредити интервал $[a, b]$ тако да је $f(a)f(b) < 0$. Почетни интервал се тада систематски смањује на подинтервале у којима се такође налази тај корен једначине, све док се не задовољи жељена тачност. Како се тачно мења дужина интервала у сваком кораку зависи од методе која се користи. Очито је да методе



Слика 10.1: Приближно решење једначине $1 - x + cosx = 0$

на затвореном интервалу увек конвергирају решењу, тј. корену. Примери таквих метода су *метода бисекције* (метода половљења интервала) и *regula fasi* (метода погрешног положаја).

Код отворених метода у току итеративног поступка не постоји ограничење да се апроксимација мора налазити у унапред утврђеном затвореном интервалу. Насупрот претходно описаним методама, отворене методе захтевају само једну вредност промењљиве x за почетак итерације, тј. захтевају почетну процену решења, блиску стварном корену. Додуше, неке отворене методе могу такође да захтевају две или више тачака, али не захтевају да се корен изолује у затвореном интервалу. Након почетне процене решења, тачније апроксимације, решења се генеришу сукцесивно, посебним методама (слика 10.2б). Ове методе, уопштено, не гарантују конвергенцију све док почетна претпоставка x_0 није довољно блиска корену једначине. Међутим, чак и тада, нумерички алгоритам може да дивергира. Упркос томе, предност отворених метода је у томе што ако конвергирају, оне демонстрирају много бољу брзину конвергенције у поређењу са методама на затвореном интервалу. Неке од отворених метода су *метода фиксне тачке*, *Њутнова метода*, *метода сечице* (она захтева две тачке за почетак итерације) и *инверзна квадратна интерполација* (захтева три почетне тачке).

10.1.1 Решавање нелинеарних једначина у Матлабу

Матлаб поседује само две функције за одређивање корена једначине. То су функција `fzero()`, која се може применити за било коју функцију једне промењљиве, и функција `roots()` за одређивање нула полинома. Функција `fzero()` користи комбинацију метода бисекције, сечице и инверзне квадратне интерполације за налажење решења. Општи облик позива функције је:

$$x = \texttt{fzero}(fun,\, x_0)$$

Функција покушава да нађе нулу функције fun узимајући x_0 као почетну апроксимацију решења. Уколико је x_0 скалар, процедура за тражење нуле

Пример 10.1. *Одредити нуле функције* $f(x) = 1 - x + \cos x$.

Решење: За пример функције $f(x) = 1 - x + \cos x$ са слике 10.1, може се уочити да је њена нула у околини вредности 1, што се може искористити као почетна претпоставка и као улазни аргумент функције `fzero()`:

```
>> fun = @(x) 1 - x + cos(x)
>> r = fzero(fun,1)
r =
    1.2834
```

Претходно се може добити и ако се функција `fzero()` позове са почетном проценом интервала на коме се налази нула функције, на пример са двоелементним вектором $\mathbf{x}_0 = \begin{bmatrix} 1 & 2 \end{bmatrix}$:

```
>> r = fzero(fun,[1 2])
r =
    1.2834
```

Слично, функција $f(x) = 1 - x + cos(x)$ се могла директно проследити функцији `fzero()` као стринг:

```
>> r = fzero('1 - x + cos(x)', 1)
r =
    1.2834
```

10.1.2 Функција fzero() и једначина са више коренова

Функција `fzero()` се може искористити и у случају да нека једначина на датом интервалу има више коренова, и то илуструје следећи пример.

Пример 10.2. *Наћи коренове једначине* $xsinx = 0$ *на интервалу* $[-10, 10]$.

Решење: На датом интервалу постоји неколико коренова једначине, те се сваки корен мора одредити посебно, користећи одговарајуће почетне процене. Ове почетне процене се могу генерисати израчунавањем вредности функције у одређеном броју тачака из датог интервала, и идентификујући промену знака функције.

```
fun = @(x) x.*sin(x);
x = linspace(-10,10,20);
f = fun(x);
plot(x,f)
```

Свака промена знака функције $f(x)$ може се идентификовати на следећи начин:

```
I = find(sign(f(2:end)) ~= sign(f(1:end-1)));
```

Сада се у петљи за сваку промену знака применом функције `fzero()` са почетном проценом у облику двоелементног вектора:

```
for n = 1:length(I)
    r(n) = fzero(fun, x([I(n) I(n)+1]));    % Priblizhna reshenja
end
```

ПОГЛАВЉЕ 11

Нумеричко диференцирање и интеграљење

11.1 Нумеричко диференцирање

У многим инжењерским проблемима захтева се израчунавање извода, као на пример одређивање брзине на основу података о позицији, односно пређеном путу. Некада је познат аналитички опис функције која се диференцира, али је аналитичко диференцирање веома компликовано. У том случају, функција се дискретизује и генерише одређен број тачака које се потом користе у некој нумеричкој методи за апроксимацију извода функције. Са друге стране, често су доступни подаци управо у облику дискретног скупа тачака, као резултат експерименталних мерења. У том случају, извод се може нумерички апроксимирати на један од следећа два начина. Један начин је диференцирањем интерполационих формула, када се подаци апроксимирају погодном кривом, лаком за диференцирање, а потом диференцира аналитички израз добијене функције. На пример, за случај интерполације непознате функције $f(\cdot)$ полиномом, апроксимациона функција је облика

$$\phi(x) = P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0, \tag{11.1}$$

тако да је поступак диференцирања једноставан, тј. $f^{(k)}(x) \approx \phi^{(k)}(x)$. Међутим, треба напоменути да овакав поступак диференцирања може да буде нестабилан и нетачан, јер мала одступања апроксимационе функције од стварне, $|f(x) - P_n(x)|$ не значе обавезно и мала одступања њихових извода $|f^{'}(x) - P_n^{'}(x)|$. Са друге стране, добра особина је што познат, дискретан скуп тачака не мора да буде еквидистантан.

Други прилаз је примена коначних разлика у којима се користе подаци у околини тачке од интереса за апроксимацију извода, и о овим методама ће бити више речи у наставку.

Пример 11.1. *Показати примену функције* `diff()` *у приближном одређивању извода функције*

$$f(x) = 0,2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5.$$

и упоредити добијене резултате са тачним решењем на интервалу $x \in [0,\ 0,8]$.

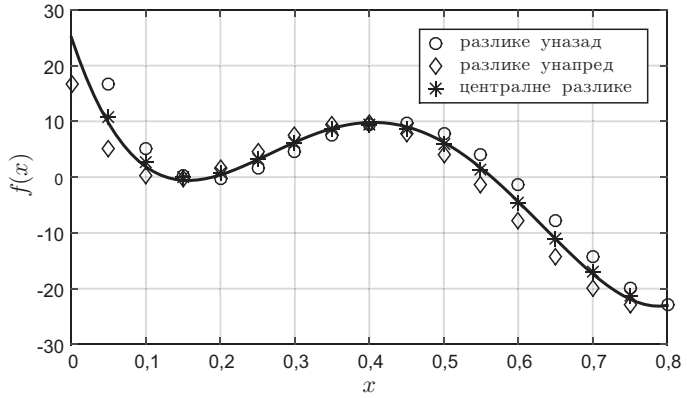
Решење: Предлог решења дат је у следећем скрипту. Поређење аналитичког и приближних решења дато је на слици 11.2, при чему је приказан дијаграм који одговара графику $figure(2)$ у скрипту. Из приказаног резултата може се уочити да се применом коначних разлика уназад и унапред не може добити вредност извода у првој, односној последњој тачки интервала, следствено, а применом централних разлика се не могу добити вредности извода у крајњим тачкама посматраног интервала.

```
f = @(x) 0.2 + 25*x - 200*x.^2 + 675*x.^3 - 900*x.^4 + 400*x.^5;
x = 0:0.05:0.8; n = length(x);
y = f(x);         % izrachunavanje funkcije u izabranim tachkama
xa = 0:0.01:0.8;  % za crtanje analitichkog reshenja
% analitichko reshenje:
yda = 25 - 400*xa + 3*675*xa.^2 - 4*900*xa.^3 + 5*400*xa.^4;
d = diff(y)./diff(x);
% Aproksimacija izvoda: razlike unazad
subplot(3,1,1)
plot(x(2:n),d,'bo'), hold on, plot(xa,yda), grid
% Aproksimacija izvoda: razlike unapred
subplot(3,1,2)
plot(x(1:n-1),d,'rd'), hold on, plot(xa,yda), grid
% Aproksimacija izvoda: centralne razlike.
dc = (y(3:n) -y(1:n -2))./(x(3:n)-x(1:n-2));
subplot(3,1,3)
plot(x(2:n-1),dc,'k*'), hold on, plot(xa,yda), grid
% Na jednom dijagramu
figure(2)
plot(x(2:n),d,'bo', x(1:n-1),d,'rd', x(2:n-1),dc, 'k*')
hold on, grid on, plot(xa,yda)
legend('razlike unazad', 'razlike unapred', 'centralne razlike')
```

Функција polyder()

Функција `polyder()` омогућава да се аналитички одреди извод полиномске функције, одређене њеним коефицијентима. При томе, функција дозвољава три облика позива:

- `b=polyder(p)` – враћа вектор **b** који садржи коефицијенте извода полинома дефинисаног вектором **p** (по опадајућем степену),
- `b=polyder(p1,p2)` – враћа вектор **b** који садржи коефицијенте извода производа полинома $\mathbf{p}_1$ и $\mathbf{p}_2$,
- `[n,d]=polyder(p1,p2)` – враћа векторе **n** и **d** који садрже коефицијенте бројиоца и имениоца извода количника полинома $\mathbf{p}_1$ и $\mathbf{p}_2$.



Слика 11.2: Поређење приближних и тачних извода функције из примера 11.1.3

На пример, извод полинома $p(x) = 2x^3 - 5x + 3$ је $p'(x) = 6x^2 - 5$ је:

```
>> p = [2 0 -5 3]
>> polyder(p)
ans =
     6     0    -5
```

У Матлабу се може одредити и градијент скупа података који представљају дводимензионалну функцију помоћу функције `gradient()`. Један од облика позива функције је

$$\texttt{[}\mathbf{F}_x\texttt{,}\mathbf{F}_y\texttt{]} = \texttt{gradient(}F\texttt{,}dx\texttt{,}dy\texttt{)}$$

где је F матрица, а $\mathbf{F}_x$ и $\mathbf{F}_y$ су вектори парцијалних извода $\partial F/\partial x$ и $\partial F/\partial y$, следствено. Више о функцији се може наћи наредбом `help gradient`.

11.2 Нумеричко интеграљење

Веома важно место у многим инжењерским проблемима заузима и нумеричко интеграљење (на пример, израчунавање пређеног пута на основу зависности брзине кретања од времена). Основна идеја нумеричке интеграције је израчунавање интеграла

$$I = \int_a^b f(x)dx \tag{11.19}$$

коришћењем вредности функције $f(\cdot)$ на неком коначном скупу тачака. Формуле за приближно интеграљење функција једне променљиве зову се и *квадратурне формуле*, због интерпретације интеграла као површине испод криве, тј. испод функције $f(\cdot)$.

Од многих метода, у наставку ће се укратко изложити две, трапезно и Симпсоново правило, а на којима се заснивају уграђене функције Матлаба. Обе методе се добијају из Њутн-Котесових (Newton-Cotes) формула

где интегранд мало варира, а краћи интервали у областима где интегранд показује нагле промене. У анализи која следи, међутим, претпоставља се да су тачке еквидистантне. На сваком од подинтервала $[x_i, x_{i+1}]$ интеграл се апроксимира површином трапеза. Сабирањем површина трапеза долази се до приближне вредности одређеног интеграла:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} \frac{f(x_i) + f(x_{i+1})}{2}(x_{i+1} - x_i). \tag{11.22}$$

Грешка општег трапезног правила је реда h^2, тј. $\mathcal{O}(h^2)$.

11.2.2 Матлаб функција trapz()

У Матлабу је имплементирана трапезна метода приближне интеграције у облику функције `trapz()`. Њена синтакса је

`I=trapz(x,y)`

где низ **y** садржи вредности функције израчунате за тачке садржане у низу **x**, а I је вредност интеграла. Уколико се жели интеграл једне функције, тада је **y** вектор, а за случај више функција њихове вредности се морају дефинисати у облику матрице Y. Тада ће функција израчунати интеграл сваке од колона матрице Y. Важно је да се не може аналитички дефинисати функција као интегранд, већ интегранд мора бити дискретан скуп података.

Пример 11.2. *Одредити интеграл* $\int_{-1}^{1} \frac{1}{x+3}dx$ *применом трапезног правила.*

Решење: Да би се применило трапезно правило, потребно је генерисати дискретни скуп тачака и потом позвати функцију `trapz()`. Следећи низ наредби илуструје решење:

```
>> x = -1:0.1:1;
>> y = 1./(x+3);
>> I = trapz(x,y)
I =
    0.6933
```

Аналитичко решење је:

$$\int_{-1}^{1} \frac{1}{x+3}dx = \ln|x+3|\Big|_{-1}^{1} = \ln 4 - \ln 2 = \ln 2 = 0.6931.$$

То се може проверити и применом симболичког рачуна:

```
>> syms x
>> int(1/(x+3),-1,1)
ans =
log(2)
```

Поделом интервала интеграције $[-1, 1]$ на већи број тачака може се добити још тачнији резултат.

ПОГЛАВЉЕ 12

Нумеричко решавање обичних диференцијалних једначина

12.1 Појам диференцијалне једначине

За разлику од алгебарских једначина, у *диференцијалним једначинама* непозната која се одређује је реална (или комплексна) функција. У диференцијалној једначини фигуришу независна променљива t, непозната функција $y(\cdot)$ и неки њени изводи. Општи облик скаларне диференцијалне једначине је:

$$\phi(t, y, \dot{y}, \ddot{y}, \ldots, y^{(n)}) = 0 \tag{12.1}$$

Уколико је реч о функцији једне реалне променљиве, као у једначини (12.1), тада је у питању *обична диференцијална једначина*. Осим обичних, постоје и *парцијалне диференцијалне једначине* код којих непозната функција зависи од више независно променљивих, па у диференцијалној једначини фигуришу парцијални изводи. Надаље, предмет разматрања ће бити само обичне диференцијалне једначине. Слично алгебарским једначинама, постоје и системи диференцијалних једначина. Диференцијалне једначине се јављају као математички модели различитих појава, процеса и система у природи и техници, и о томе ће бити речи нешто касније.

Највиши извод тражене функције који се појављује у диференцијалној једначини представља *ред* диференцијалне једначине. Решити диференцијалну једначину значи одредити непознату функцију $y(\cdot)$ која се у њој јавља. *Решење* диференцијалне једначине на неком интервалу J је пресликавање $y(\cdot) : J \longrightarrow \mathcal{R}$ диференцијално n пута, тако да је једначина (12.1) задовољена за свако $t \in J$. Ако се претпостави да се једначина (12.1) може решити по највећем изводу непознате фунције $y(\cdot)$, добија се једначина

$$y^{(n)} = \varphi(t, y, \dot{y}, \ddot{y}, \ldots, y^{(n-1)}) = 0, \tag{12.2}$$

Задавањем n услова које решење једначине (12.5) треба да задовољава, из општег решења се одређује тзв. *партикуларно* решење. Једначина (12.5) заједно са условима задатим у једној тачки, означеној са $\mathbf{x}_0$, представља проблем који се назива *проблем почетних вредности* или *Кошијев проблем*.

У циљу разумевања метода за нумеричко решавање Кошијевог проблема, и због погодности графичке интерпретације а без губитка у општости, разматра се скаларна диференцијална једначина првог реда са датим почетним условом:

$$\dot{x}(t) = f(t, x), \quad x(t_0) = x_0. \tag{12.6}$$

Све методе које се илуструју у наставку важе и у општем случају векторског облика претходне једначине.

Према томе, Кошијев проблем се састоји из одређивања оног решења једначине (12.6) које пролази кроз дату тачку, тј. решења које задовољава услов $x(t_0) = x_0$. У наставку, усваја се да важи следећа претпоставка:

Претпоставка 12.1. *Систем* (12.6) *има јединствено решење на интервалу* $[a, b]$.

Нумеричко решење овог проблема генерише низ вредности независне променљиве $t_0, t_1, \ldots, t_n$ и одговарајући низ вредности зависне променљиве $x_0, x_1, \ldots, x_n$, тако да свако x_i апроксимира решење у тачки t_i:

$$x_i \approx x(t_i),\ \forall i = 0, 1, \ldots, n.$$

Разлика h_k, $h_k = t_k - t_{k-1}$ представља *корак дискретизације*, или *корак интеграције* на k-том сегменту $[t_{k-1}, t_k]$ и он у општем случају не мора бити константан, што је назначено индексом k. Наравно, у случају константног корака интеграције и поделе интервала на коме се Кошијев проблем решава на n сегмената (подинтервала), важи следеће:

$$t_1 = t_0 + h,\ t_2 = t_0 + 2h,\ \ldots,\ t_n = t_0 + nh, \quad \text{где је} \quad h = \frac{t_n - t_0}{n}.$$

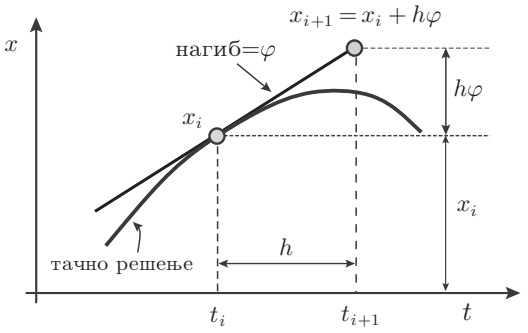
Решење у тачки t_0 је познато из датог почетног услова. Циљ је наћи приближна решења у следећим тачкама $t_1, t_2, \ldots, t_n$.

12.3 Једнокорачне методе

Једнокорачним методама се приближно решење у тачки x_{i+1} које одговара вредности t_{i+1} добија екстраполацијом, а на основу вредности x_i у претходној тачки t_i. На који начин се ова нова вредност добија екстраполацијом зависи од врсте нумеричке методе која се користи. На слици 12.1 је приказана једноставна једнокорачна метода, где се вредност нагиба $\phi(t_i, x_i)$ праве кроз тачку (t_i, x_i) користи за одређивање вредности x_{i+1}, тј.:

$$x_{i+1} = x_i + h\phi(t_i, x_i),\ i = 0, 1, \ldots, n. \tag{12.7}$$

Вредност $\phi(t_i, x_i)$ се може тумачити као приближни прираштај по јединичном кораку и геометријски представља нагиб постављене праве у тачки (t_i, x_i), а назива се и *функција прираштаја*. Почевши од унапред познатог почетног услова x_0, претходна једначина се примењује на сваки подинтервал $[t_i, t_{i+1}]$ за налажење приближног решења у тачкама $t_1, t_2, \ldots, t_n$. Општи облик једначине (12.7) описује све једнокорачне методе, при чему се методе разликују у свом прилазу за одређивање $\phi(t_i, x_i)$. Методе су добиле назив по томе што се за апроксимацију x_{i+1} користи само вредност x_i у претходној тачки, односно, у једном кораку се из x_i добија x_{i+1}. Најједноставнија једнокорачна метода је Ојлерова метода, којој је посвећен наредни одељак.



Слика 12.1: Једнокорачна метода.

12.3.1 Ојлерова метода

Ојлерова метода обично даје најмање тачне резултате, али ипак пружа добру основу за разумевање софистицираних метода, те се стога разматра у наставку. Код ове методе се за вредност функције $\phi(t_i, x_i)$ узима вредност функције $f(t_i, x_i)$, тј. узима се нагиб на почетку интервала као апроксимација средњег нагиба на целом интервалу.

$$x_{i+1} = x_i + hf(t_i, x_i),\ i = 0, 1, \ldots, n. \tag{12.8}$$

Претходна једначина се може добити на различите начине, па се у наставку кроз поменуту методу илуструје суштина нумеричког решавања диференцијалне једначине.

Геометријски аспект

Посматра се график тачног решења Кошијевог проблема $x(t)$. Познати почетни услов представља тачку која се налази на тачном решењу, тако да је (t_0, x_0) природан и логичан избор за почетну тачку поступка апроксимације. Да би се добило приближно решење у тачки t_1, израчунава се нагиб тангенте у тачки (t_0, x_0) као $\dot{x}_0 = f(t_0, x_0)$, и тангента представља праву линију чија је једначина

$$T_0(t) = x_0 + (t - t_0)\dot{x}_0. \tag{12.9}$$

За приближно решење x_1 у тачки t_1 може се искористити вредност тангенте T_0 у тој тачки, тј.:

$$x_1 = T_0(t_1) = x_0 + (t_1 - t_0)\dot{x}_0 = x_0 + hf(t_0, x_0). \tag{12.10}$$

ode23s. Овај једнокорачни нумерички поступак је заснован на модификованој Росенброковој формули другог реда и намењен је за симулацију крутих динамичких система. У случају грубљих толеранција овај поступак је ефикаснији од ode15s поступка.

ode23tb. Представља имплицитну методу Рунге Кута која користи комбинацију трапезног правила и диференцијалне формуле другог реда. Као и ode23s, овај поступак може бити ефикаснији у односу на ode15s поступак при грубљим толеранцијама.

12.7 Функција ode..() у Матлабу

Синтакса позива је иста за све нумеричке методе, то јест решаваче. Нумеричка метода се може позвати из командне линије на следећи начин:

$$\texttt{[t,x]= odesolver(}odefun\texttt{,tspan,x}_0\texttt{)}$$

$$\texttt{[t,x]=odesolver(}odefun\texttt{,tspan,x}_0\texttt{,}options\texttt{)}$$

Параметар *odefun* представља име функцијске датотеке или анонимне функције која се користи за израчунавање десне стране диференцијалне једначине (12.6) за дате вредности независне и зависне променљиве. Уколико се користи m-датотека, дефиниционa линија функције обично има облик

```
function xdot = odefun(t,x)
```

и датотека се снима под именом odefun.m. Излазна величина **xdot** мора бити вектор исте димензије као **x**. Може се уочити да независна променљива **t** мора бити део листе улазних аргумената, чак и ако се она не појављује експлицитно у изразу који се користи за израчунавање **xdot**. Променљива *odefun* може да садржи име m-датотеке или може да буде показивач генерисан анонимном функцијом. Двоелементни вектор **tspan** дефинише интервал интеграције $[t_0\ t_f]$, где је t_0 почетни а t_f крајњи тренутак симулације. Осим тога, то може бити вектор вредности за које се жели решење. Вектор $\mathbf{x}_0$ представља вектор почетних услова за зависну променљиву. Наравно, потребно је онолико почетних услова колико има зависних променљивих. Параметар *options* представља једну Матлаб структуру (коју креира функција `odeset()`), и она омогућава да се утиче на параметре израчунавања. Овај аргумент је опционалан, и уколико се не користи, решавач ће користи подразумеване вредности, о којима ће бити речи нешто касније. Вектор **t** садржи вредности независне променљиве за које се низ решења **x** израчунава. Треба имати у виду да, у подразумеваном случају, ове вредности не представљају еквидистантан скуп вредности. Свака колона од **x** представља различиту зависну променљиву. Величина низа је `length(t)` $\times$ `length(`$\mathbf{x}_0$`)`.

Примена функције биће илустроваана на примеру. Посматра се најпре скаларна диференцијална једначина

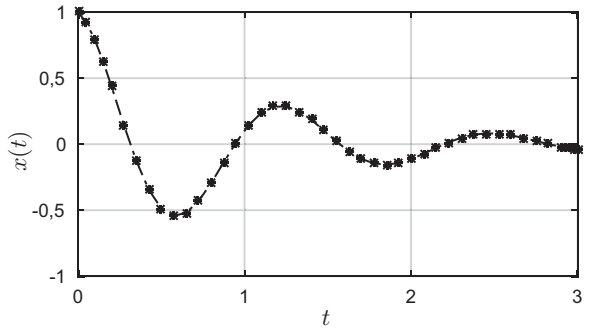
$$\dot{x}(t) = -x(t) - 5e^{-t}\sin 5t, \quad x(0) = 1$$

коју треба решити на интервалу $0 \leq t \leq 3$. Креира се анонимна функција

```
xdot = @(t,x) -x - 5*exp(-t)*sin(5*t);
```

и низом следећих наредби добија се решење, приказано на слици 12.6:

```
tspan=[0 3]; x0=1;
[t,x] = ode45(xdot,tspan,x0);
plot(t,x,'*-.')
grid on
xlabel('$t$','Interpreter','Latex')
ylabel('$x(t)$','Interpreter','Latex')
```



Слика 12.6: Решење скаларне диференцијалне једначине

Дакле, улазни аргументи при позиву функције су показивач на функцију *xdot*, двоелементни вектор **tspan** који дефинише интервал на коме се тражи решење, и почетни услов x_0. Излазни аргументи су вектори **t** и **x**. Вредности вектора **t** су у опсегу $[0, 3]$, и $\mathbf{x}(i)$ апроксимира решење у тренутку $\mathbf{t}(i)$. То значи да је $\mathbf{t}(1) = 0$ и $\mathbf{t}(\text{end}) = 3$, док су тачке $\mathbf{t}(2 : \text{end} - 1)$ аутоматски изабране од стране решавача у процесу адаптивног решавања једначине.

Ако се жели решење једначине у одређеним временским тренуцима (а не у онима у којима сама функција `ode45()` враћа), тада вектор **tspan** треба да садржи жељене вредности. Следећи низ наредби приказује један такав пример.

```
>> tspan_2 = 0:4; x0 = 1;
>> [t2,x2] = ode45(xdot,tspan_2,x0);
>> disp([t2,x2])
         0    1.0000
    1.0000    0.1043
    2.0000   -0.1136
    3.0000   -0.0378
    4.0000    0.0075
```

Овакав захтев за решењима у специфичним тренуцима има мали утицај на брзину нумеричког решавања. Важно је да специфицирање ових вредности не значи да решавач решава диференцијалну једначину само у тим тренуцима. Он и даље сам адаптивно одређује кораке и временске тренутке у процесу добијања решења које ће да задовољи тражену тачност, а ови специфирани тренуци су само повратне вредности функције – функција не

ПОГЛАВЉЕ 13

Увод у Симулинк

Понашање динамичких система описано је диференцијалним једначинама или скупом диференцијалних једначина које су уопштено нелинеарне. Основни проблем код анализе таквих система је непостојање опште методологије за решавање нелинеарних диференцијалних једначина. Као логично решење проблема анализе понашања оваквих система намеће се симулација система на дигиталном рачунару. Симулинк представља графички алат који користи математичку основу Матлаба и обезбеђује графички кориснички интерфејс који користи мноштво елемената који се називају *блокови* за симулацију динамичких система. Графички интерфејс омогућава да се блокови позиционирају, да се мења њихова величина, дају називи, специфицирају параметри блокова, међусобно повезују и комбинују у групе блокова или подсистеме, а у циљу описивања компликованих система за симулацију.

Израда симулационог модела унутар Симулинка обавља се на једноставан начин коришћењем библиотеке готових графичких блокова. Осим постојећих блокова корисник може написати и властите блокове користећи било Матлабове m-функције или функције написане у програмском језику $C/C++$ (такозване S-функције). Симулациони модел у Симулинку се састоји од блокова и линија које представљају сигнале а којима се повезују поједини блокови, и тиме реализују једначине које описују систем. Симулација унутар Симулинка се може поделити у три фазе и то:

(1) превођење модела,

(2) повезивање модела и

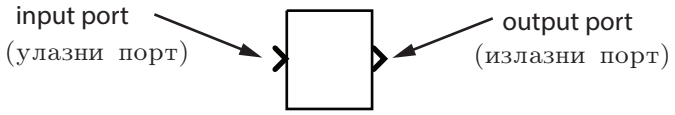
(3) симулација.

Превођење модела. У овој фази модел се преводи у извршни облик, при чему се обављају следеће операције:

- израчунавају се параметри блокова,

13.3.1 Блокови

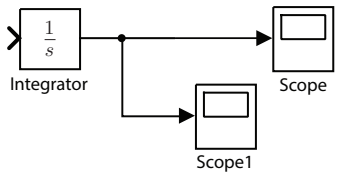
Блокови могу имати неколико улазних портова или ниједан, што се односи и на њихове излазне портове. Индикатор да је неки улазни порт неискоришћен је мали отворени троугао (стрелица), док је индикатор за неискоришћени излазни порт затворени троугао. Блок приказан на слици 13.1 има један неискоришћен улаз (на левој страни) и излаз (на десној страни).



Слика 13.1: Пример блока са улазним и излазним портом

13.3.2 Линије

Линије врше пренос сигнала у смеру одређеном стрелицом. Линије увек преносе сигнале са излаза једног блока на улаз другог блока. Изузетак једино представља линија која полази из друге линије, када се један исти сигнал шаље на улазе више блокова (гранање сигнала), слика 13.2.



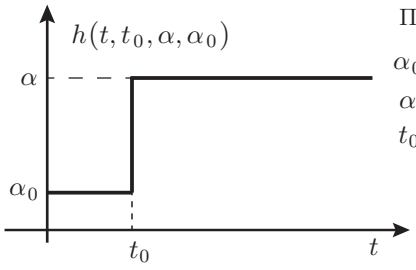
Слика 13.2: Пример гранања сигнала

Линија никада не може довести неки сигнал директно у другу линију: линије се морају комбиновати применом блокова, као што је на пример блок за сабирање. Неки сигнал може бити или *скаларни* или *векторски* сигнал. За једноструко преносне системе (системи који имају један улаз и један излаз), уопштено се користе скаларни сигнали. За вишеструко преносне системе (системи са више улаза и/или више излаза) често се користе векторски сигнали који се састоје од два или више скаларних сигнала. Линије које се користе за пренос скаларних и векторских сигнала су идентичне. Тип сигнала који се преноси линијом одређен је блоковима на било ком крају линије.

13.3.3 Креирање модела система

У циљу демонстрације како се један систем може представити у Симулинку, креираће се симулациони модел за систем у коме се синусни улазни сигнал множи константом и прослеђује на осцилоскоп (слика 13.3).

Дакле, овај модел ће се састојати од три блока: **Sine Wave**, **Gain** и **Scope**. Сигнал из **Sine Wave** блока се преноси линијом у смеру дефинисаном стрелицом до блока **Gain**, који га модификује (множи својом константном вредношћу) и генерише на свом излазу нови сигнал који се помоћу следеће линије преноси до **Scope** блока. Креирање модела система почиње



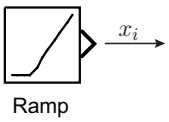
Параметри:

α_0 — `Initial value` (почетна вредност)
α — `Final value` (крајња вредност)
t_0 — `Step time` (почетни тренутак)

Слика 14.6: Функција генерисана **Step** блоком

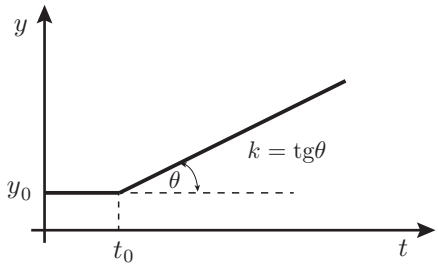
Step *блок. Све поменуте функције представљају специјалан случај функције са слике 14.6 за вредност* $\alpha_0 = 0$*, и о њима ће бити речи у одељку 19.1.*

14.1.5 Ramp блок



Слика 14.7: **Ramp** блок

Ramp блок (слика 14.7) генерише сигнал који почиње у дефинисаном тренутку и дефинисаном вредношћу, и чија се вредност мења током временом унапред дефинисаном брзином. Другим речима, то је у деловима линеарна функција са константним нагибом (слика 14.8). Карактеристике генерисаног сигнала су одређене параметрима `Slope`, `Start time` и `Initial output`.



Параметри:

y_0 — `Initial output` (почетна вредност излаза)
t_0 — `Start time` (почетни тренутак)
k — `Slope` (нагиб)

Слика 14.8: Функција генерисана **Ramp** блоком

Запажање 14.2. *У одељку 19.1 биће речи о* **нагибној** *функцији, чији облик је посебан случај функције са слике 14.8, за* $y_0 = 0$*.*

14.1.6 Sine wave блок

Sine Wave блок (слика 14.9) се користи за генерисање синусног сигнала. Синусни сигнал може бити двојаког типа: дефинисан у временском моду (`Time based`) или моду заснованом на узорковању (одабирању),

14.4.3 Trigonometric Function блок

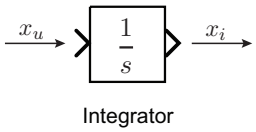
Trigonometric Function блок (слика 14.27) на свој улазни сигнал примењује тригонометријске функције, затим инверзне тригонометријске функције, хиперболичке функције и инверзне хиперболичке функције. Након избора жељене функције њено име се појављује на икони блока.

14.5 Библиотека блокова континуалних система

14.5.1 Integrator блок

Integrator блок (слика 14.28) се користи за временски непрекидне сигнале и он на свом излазу даје сигнал који представља интеграл улазне величине. Уколико је $x_u(t)$ улазна величина у **Integrator** блок, а $x_i(t)$ излазна величина из њега, онда је једначина његовог понашања облика:

$$x_i(t) = x_i(0) + \int_{t_0}^{t} x_u(t)\mathrm{d}t. \tag{14.1}$$



Слика 14.28: **Integrator** блок

Симулинк третира **Integrator** блок као један динамички систем са једним стањем, и то је његов излаз. Према томе, улазни сигнал у блок у ствари представља први извод по времену његовог стања. Изабрани решавач израчунава вредност излаза из блока у текућем временском тренутку на основу тренутне вредности улаза и стања блока у претходном временском тренутку. Да би решавач знао вредност почетног стања блока на самом почетку симулације, неопходно ју је дефинисати, и то се чини помоћу параметра `Initial condition`, чија је подразумевана вредност 0.

Прозор за подешавање параметара блока, `Function Block Parameters`, омогућава да се дефинише нека друга вредност за почетни услов, или пак, да се креира улазни порт блока за дефинисање почетне вредности. Такође, омогућава да се дефинише горња и доња граница интеграције, да се креира један улазни порт који ће ресетовати излаз (стање) блока на његову почетну вредност у зависности од промене улаза, и да се креира један опциони излаз који ће омогућити да се користи вредност излаза блока за његово ресетовање или побуђивање. У зависности од могућих избора, изглед **Integrator** блока се мења, са интуитивним приказом изабраних варијанти.

Пример 14.1. *Креирати Симулинк модел за нумеричко решавање диференцијалне једначине:*

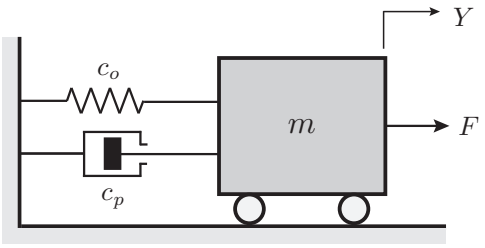
$$\dot{X}_i(t) = -10X_i(t) + X_u(t),$$

за $X_u(t) = 2\sin(4t)$ *и* $X_i(0) = 1$.

лације је 100 s. Резултати симулације су приказани на слици 15.7б.

15.2 Симулација система другог реда

Посматра се један механички систем са масом m, опругом и пригушивачем (слика 15.9), на који делује спољашња сила F.



Слика 15.9: Механички маса-опруга-пригушивач систем

На основу једначине динамичке равнотеже сила, и усвајајући да је позиција тела Y његова излазна величина, $X_i = Y$, а сила F његова улазна величина, $X_u = F$, може се добити математички модел система у облику:

$$m\ddot{X}_i(t) + c_p\dot{X}_i(t) + c_oX_i(t) = X_u(t), \tag{15.5}$$

где c_o представља крутост опруге, а c_p коефицијент вискозног трења. Ова једначина представља обичну линеарну диференцијалну једначину са константним коефицијентима другог реда.

Запажање 15.1. *Математички модел механичког система са слике 15.9 детаљније се разматра у шеснаестом поглављу.*

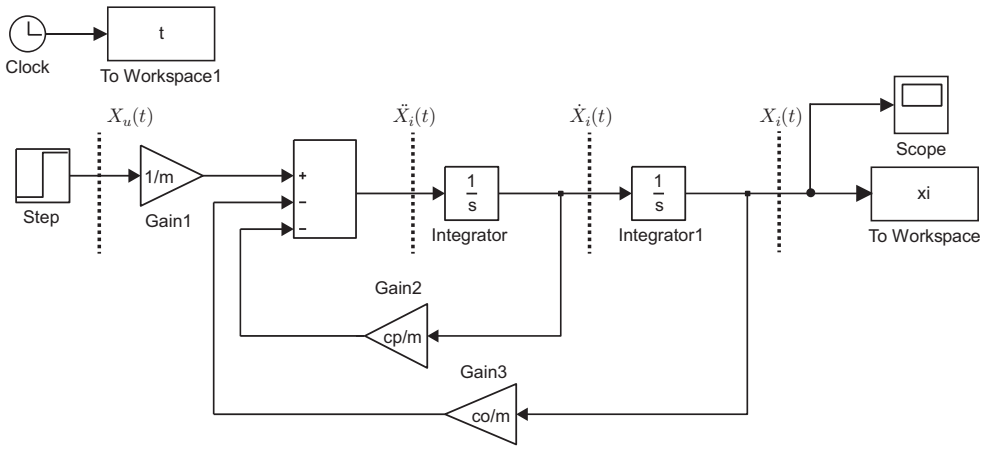
15.2.1 Креирање Симулинк модела

Пре него што се било шта уради у Симулинку, диференцијална једначина понашања (15.5) ће се написати у облику који је неопходан – решиће се по највећем изводу излазне величине. Како је највећи извод излазне величине X_i њен други извод, добија се:

$$\ddot{X}_i(t) = -\frac{c_p}{m}\dot{X}_i(t) - \frac{c_o}{m}X_i(t) + \frac{1}{m}X_u(t). \tag{15.6}$$

Дакле, други извод излазне величине је једнак алгебарском збиру неких функција, што значи да ће у симулационом моделу $\ddot{X}_i(t)$ управо бити излаз из сабирача, у који улазе све функције са десне стране претходне једначине. Треба приметити да се једначина решава по $\ddot{X}_i(t)$, а на десној страни су такође непознате функције $\dot{X}_i(t)$ и $X_i(t)$). Поставља се питање, како добити те функције са десне стране? Одговор је: на основу ,,познавања" $\ddot{X}_i(t)$ и чињенице да се $\dot{X}_i(t)$ може добити интеграцијом величине $\ddot{X}_i(t)$, а $X_i(t)$

интеграцијом $\dot{X}_i(t)$. Дакле, претпоставља се да је позната величина $\ddot{X}_i(t)$, што се представља вертикалном цртом у симулационом моделу, и додају се два интегратора на ред. Излаз из првог интегратора представља сигнал $\dot{X}_i(t)$, а излаз из другог интегратора величину $X_i(t)$. Сада је позната комплетна десна страна једначине (15.6), уз претпоставку да је промена улазне величине $X_u(t)$ увек позната. Следећи претходно описани поступак, добија се Симулинк модел приказан на слици 15.10. С обзиром на зајед-



Слика 15.10: Симулинк модел механичког система маса-опруга-пригушивач

нички члан $1/m$ у свим сабирцима на десној страни једначине (15.6), она се може написати и у облику:

$$\ddot{X}_i(t) = \frac{1}{m}\left(-c_p\dot{X}_i(t) - c_oX_i(t) + X_u(t)\right). \tag{15.7}$$

за коју се добија Симулинк модел као на слици 15.11. Да би се покренула симулација, неопходно је дефинисати све константе. То се може урадити на два начина:

- уношењем конкретних, бројчаних вредности у модел, или
- писањем посебног скрипта, у коме ће се дефинисати све константе, и извршавањем тог скрипта у радном окружењу Матлаба.

Дакле, величине из радног окружења Матлаба су доступне Симулинку, и обратно. Претходна чињеница указује да избор да се ради у општим бројевима има за последицу да се добија у неку руку универзални модел, који се може покренути са различитим скуповима параметара, а без икакве модификације самог Симулинк модела.

Пример 15.1. *Направити симулациони модел и извршити симулације рада механичког система за следеће вредности параметара:*

$$m = 1, \quad c_0 = 10, \quad c_p = \{0, 2, 8\}.$$

17.5.2 Функције са вишеструким половима

У општем случају функција $X(s)$ има једноструке и вишеструке полове. Претпоставља се да $X(s)$ има један пол $s = s_i^*$ реда $\nu_i > 1$ (вишеструкости $\nu_i > 1$), и да су сви остали полови од $X(s)$ једноструки ($\nu_1 = \nu_2 = \ldots = \nu_{i-1} = \nu_{i+1} = \ldots = \nu_n = 1$). Тада Хевисајдов развој од $X(s)$ има следећи облик:

$$X(s) = R_0 + \frac{R_1}{s - s_1^*} + \ldots + \frac{R_{i-1}}{s - s_{i-1}^*} + \frac{R_{i+1}}{s - s_{i+1}^*} + \ldots + \frac{R_n}{s - s_n^*} + \frac{R_{i1}}{(s - s_i)^*} + \frac{R_{i2}}{(s - s_i^*)^2} + \ldots + \frac{R_{i\nu_i}}{(s - s_i^*)^{\nu_i}}. \tag{17.7}$$

И овде важи да у случају $R_0 \neq 0$ треба $X(s)$ заменити са $X^-(s)$.

Коефицијенти R_{ij} функције $X(s)$ који одговарају полу s_i^* вишеструкости ν_i одређени су са:

$$R_{ij} = \frac{1}{(\nu_i - j)!} \frac{d^{\nu_i - j}}{ds^{\nu_i - j}} \left[(s - s_i^*)^{\nu_i} \frac{\varphi(s)}{f(s)} \right] \Bigg|_{s = s_i^*}. \tag{17.8}$$

Запажање 17.1. *У случају дегенеративне функције $X(s)$ пожељно је прво ту функцију довести на њен недегенеративни облик скраћивањем свих једнаких нула и полова, па тек онда приступити Хевисајдовом развоју недегенеративног облика. На овај начин се значајно поједностављују израчунавања.*

17.6 Хевисајдов развој у Матлабу

Матлаб поседује функцију за Хевисајдов развој функције $X(s)$, то јест за њено растављање на парцијалне чиниоце. Посматра се функција $X(s)$, која се може приказати на следећи начин:

$$X(s) = \frac{\varphi(s)}{f(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \cdots + a_1 s + a_0}, \tag{17.9}$$

где неки од коефицијената b_i и a_i могу бити једнаки нули. У Матлабу вектори врсте **num** и **den** одређују коефицијенте бројиоца и имениоца функције $X(s)$, то јест:

$$\mathbf{num} = [b_m \quad b_{m-1} \quad \ldots \quad b_1 \quad b_0], \quad \mathbf{den} = [a_n \quad a_{n-1} \quad \ldots \quad a_1 \quad a_0].$$

Функција

$$[\mathbf{r},\mathbf{p},k] = \texttt{residue}(\mathbf{num},\mathbf{den})$$

одређује вектор коефицијената **r**, вектор полова **p** и директни члан развоја k у парцијалне разломке количника два полинома $\varphi(s)/f(s)$. Функција $X(s)$ тада постаје

$$X(s) = \frac{\varphi(s)}{f(s)} = \frac{\mathbf{r}(1)}{s - \mathbf{p}(1)} + \frac{\mathbf{r}(2)}{s - \mathbf{p}(2)} + \ldots + \frac{\mathbf{r}(n)}{s - \mathbf{p}(n)} + k. \tag{17.10}$$

Упоређујући једначине (17.5) и (17.10) очигледна је следећа веза:

$$\begin{aligned}&\mathbf{r}(1)=R_1, \quad \mathbf{r}(2)=R_2, \quad \ldots, \quad \mathbf{r}(n)=R_n,\\ &\mathbf{p}(1)=s_1^*, \quad \mathbf{p}(2)=s_2^*, \quad \ldots, \quad \mathbf{p}(n)=s_n^*,\\ &k=R_0.\end{aligned}$$

Важно је уочити да ако постоји пол $\mathbf{p}(j)$ који је реда ν (тј. вишеструкости ν), тада развој у парцијалне разломке укључује чланове

$$\frac{\mathbf{r}(j)}{s-\mathbf{p}(j)}+\frac{\mathbf{r}(j+1)}{(s-\mathbf{p}(j))^2}+\ldots+\frac{\mathbf{r}(j+\nu-1)}{(s-\mathbf{p}(j))^\nu}.$$

Запажање 17.2. *У општем случају, у Матлабу је могуће раставити у парцијалне разломке и рационалну функцију која је количник два полинома, где је полином у бројиоцу вишег реда од полинома у имениоцу. У том случају директни члан k је функција променљиве s, то јест $k=k(s)$. С обзиром на то да у системима аутоматског управљања диференцијалне једначине представљају математичке моделе реалних, физичких система, и није могуће да степен полинома бројиоца буде већи од степена полинома имениоца, у наставку се неће третирати такви случајеви.*

Пример 17.3. *Раставити на парцијалне разломке функцију комплексне променљиве s:*

$$X(s)=\frac{s^3+s^2-2s-9}{s^3-3s+2}.$$

Решење: Извршавањем следећих наредби:

```
>> num = [1 1 -2 -9]; den = [1 0 -3 2];
>> [r,p,k] = residue(num,den)
r =
   -1.0000
    2.0000
   -3.0000

p =
   -2.0000
    1.0000
    1.0000

k =
     1
```

добија се

$$X(s)=1-\frac{1}{s+2}+\frac{2}{s-1}-\frac{3}{(s-1)^2}.$$

Пример 17.4. *Раставити на парцијалне разломке функцију комплексне променљиве s:*

$$X(s)=\frac{s^2+4s+13}{(2s^2+12s^2+20s+16)(s+3)}.$$

функције комплексне променљиве s:

$$W(s) = \frac{p(s)}{q(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0}, \quad m \leq n. \tag{18.1}$$

18.1.1 TF облик преносне функције

Да би се формирао модел у Матлабу, неопходно је бројилац и именилац преносне функције, тј. полиноме, представити векторима врсте чији су елементи коефицијенти одговарајућег степена од s у полиномима, по опадајућем степену. Нека вектор $\mathbf{num} = \begin{bmatrix} b_m & b_{m-1} & \ldots & b_1 & b_0 \end{bmatrix}$ представља полином бројиоца, а вектор $\mathbf{den} = \begin{bmatrix} a_n & a_{n-1} & \ldots & a_1 & a_0 \end{bmatrix}$ полином имениоца преносне функције $W(s)$. Наредбом

W=`tf(`**num**`,`**den**`)`

се генерише преносна функција система (18.1).

Пример 18.1. *Представити у Матлабу систем дат преносном функцијом:*

$$W(s) = \frac{p(s)}{q(s)} = \frac{4s^2 + 8s + 5}{s^3 + 2s^2 + 4s + 8}.$$

Решење: Бројилац и именилац преносне функције се могу представити векторима $\mathbf{num} = \begin{bmatrix} 4 & 8 & 5 \end{bmatrix}$ и $\mathbf{den} = \begin{bmatrix} 1 & 2 & 4 & 8 \end{bmatrix}$, следствено. Тада се преносна функција може добити са:

```
>> num = [4 8 5]; den = [1 2 4 8];
>> W = tf(num,den)
W =
     4 s^2 + 8 s + 5
  ---------------------
  s^3 + 2 s^2 + 4 s + 8

Continuous-time transfer function.
>> whos W
  Name      Size            Bytes  Class    Attributes

  W         1x1              1081  tf
```

Дакле, систем постоји у радном простору Матлаба као LTI објект у TF облику преносне функције.

Запажање 18.1. *Функција* `tf()` *се може користити и тако што се вектори који дефинишу полиноме бројиоца и имениоца директно уносе као аргументи функције, без дефинисања посебних векторских променљивих. Према томе, претходна преносна функција се може добити и наредбом:*

```
>> W = tf([4 8 5], [1 2 4 8])
```

Преносна функција се може дефинисати и директно као функција комплексне променљиве s тако што се s дефинише као TF модел, а потом и преносна функција као рационалан израз у функцији од s. На пример:

18.2 Модел система у простору стања

За систем описан у простору стања (УСИ систем) функцијом `ss()` генеришу се једначина стања и једначина излаза као LTI објект који се назива SS објект:

$$sys = \texttt{ss}(A, B, C, D)$$

Матрице A, B, C и D су матрице из математичког описа УСИ система.

Пример 18.5. *УСИ систем је дат својим матрицама:*

$$A = \begin{bmatrix} -5 & 1 \\ -6 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad D = 0.$$

Представити дати систем у Матлабу моделом у простору стања.

Решење: Дати систем представља једноструко преносни систем и његов модел у простору стања у Матлабу се добија наредбама:

```
>> A = [-5 1; -6 0]; B = [1; 1];
>> C = [1 0]; D = 0;
>> sistem = ss(A,B,C,D)
sistem =
  a =
       x1  x2
   x1  -5   1
   x2  -6   0

  b =
       u1
   x1   1
   x2   1

  c =
       x1  x2
   y1   1   0

  d =
       u1
   y1   0

Continuous-time state-space model.
```

Запажање 18.4. *Треба приметити да Матлаб користи мала слова a, b, c, d у приказу модела, а не велика слова којима је позвана функција* `ss()`*. За анализу модела који постоји у* SS *облику у радном простору Матлаба довољно је користити додељено му име (у претходном примеру то је sistem) као аргумент одговарајућих функција.*

У наставку ће бити приказана примена већ постојећих функција за претварање у TF облик.

Нека је дефинисан LTI објект неког једноструко преносног система у простору стања, тј. познат је његов SS облик, и нека је то променљива *SSmodel*. Наредбом:

`W=tf(SSmodel)`

добија се LTI објект у облику преносне функције. Други начин је применом функције `ss2tf()`:

`[num,den]=ss2tf(A,B,C,D)`

где се као резултат добијају вектори врсте **num** и **den** који дефинишу бројилац и именилац преносне функције. Дакле, у овом случају улазни параметар функције није LTI објект већ матрице система, и као резултат се не добија сама преносна функција, већ вектори који је дефинишу. Наравно, применом наредбе `tf(num,den)` може се добити и сама преносна функција. У случају вишеструко преносног система позив функције `ss2tf()` је облика:

`[NUM,den]=ss2tf(A,B,C,D,k)`

и као резултат се у овом случају добија матрица NUM, која садржи коефицијенте бројилаца преносних функција за k-ту улазну величину са онолико врста колико има излазних величина, и вектор **den** који садржи коефицијенте имениоца преносних функција.

Пример 18.7. *Нека је систем описан у простору стања:*

$$\dot{\mathbf{x}} = \begin{bmatrix} -4 & 1 & 0 \\ -6 & 0 & 1 \\ -4 & 0 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix} \mathbf{x}_u, \quad \mathbf{x}_i = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \mathbf{x}.$$

Одредити преносну функцију система.

Решење: У питању је једноструко преносни систем, тј. $N = M = 1$. Применом следећих наредби:

```
>> A = [-4 1 0; -6 0 1; -4 0 0];
>> B = [0; 1; 3]; C = [1 0 0]; D = 0;
>> SSmodel = ss(A,B,C,D);
>> W = tf(SSmodel);
```

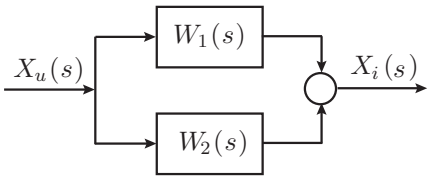
добија се преносна функција система:

$$W(s) = \frac{s+3}{s^3 + 4s^2 + 6s + 4}.$$

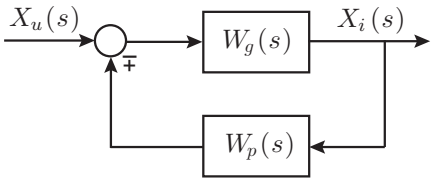
Наредбом:

```
>> [num,den] = ss2tf(A,B,C,D);
```

добију се вектори **num** $= \begin{bmatrix} 1 & 3 \end{bmatrix}$ и **den** $= \begin{bmatrix} 1 & 4 & 6 & 4 \end{bmatrix}$ који одређују бројилац и именилац преносне функције.



Слика 18.7: Паралелна спрега



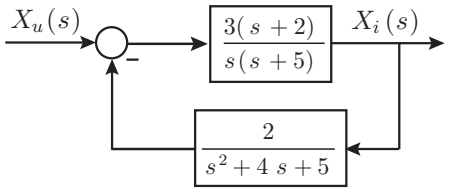
Слика 18.8: Повратна спрега

Ако су у Матлабу дефинисане преносне функције $W_g(s)$ и $W_p(s)$ својим LTI објектима у форми преносне функције W_g и W_p, следствено, тада се еквивалентна преносна функција повратне спреге са слике 18.8 може добити наредбом

$$W = \texttt{feedback}(W_g, W_p, \; znak)$$

где параметар *znak* има вредност знака повратне спреге (+1 или −1). Уколико је у питању негативна повратна спрега (што је подразумевана опција), претходна наредба се може користити у облику без параметра *znak*, $W = \texttt{feedback}(W_g, W_p)$.

Пример 18.14. *Наћи преносну функцију затвореног САУ чији је блок дијаграм приказан на слици 18.9. Одредити* LTI *објект у Матлабу у форми преносне функције и одредити њене нуле и полове.*



Слика 18.9: Блок дијаграм система из примера 18.14

Решење: Најпре се дефинишу подсистеми посматраног система, тј. преносне функције $W_1(s)$ и $W_2(s)$. Преносна функција $W_1(s)$ је дата у факторизованом облику: има једну нулу, $s^0 = -2$, два пола $s_1^* = 0$ и $s_2^* = -5$, а појачање је $k = 3$, па се може дефинисати функцијом `zpk()`. Преносна функција $W_2(s)$ је дата у развијеном облику полинома у имениоцу, па се може дефинисати функцијом `tf()`. Ова два система су повратно спрегнута, те се наредбама:

```
>> W1 = zpk(-2, [0 -5], 3); W2 = tf([2], [1 4 5]);
>> W = feedback(W1,W2)
W =
            3 (s+2) (s^2 + 4s + 5)
  ----------------------------------------
  (s+5.312) (s+0.6638) (s^2 + 3.024s + 3.403)
```

ПОГЛАВЉЕ 19

Анализа линеарних система

19.1 Типичне промене улазних величина

19.1.1 Значај и врсте типичних промена улазних величина

У већини система управљања време је једина независна променљива, и обично је циљ да се одреди промена излаза током времена, односно, да се одреди *одзив* система. У проблемима анализе система, на улаз у систем се доводи задати (жељени) улазни сигнал, а потом се одређују карактеристике система разматрањем одзива система у временском домену. На пример, ако је циљ система управљања да излазна величина система прати улазни, задати сигнал, од неког почетног тренутка и за неке почетне услове, неопходно је упоређивати задату величину и одзив система као функције времена. Стога, у већини система управљања крајња оцена карактеристика система је у временском домену и заснива се на *временском одзиву*.

Први корак у анализи система управљања је одређивање математичког модела система. Када је модел познат, на располагању су различите методе за анализу динамичких карактеристика система. У пракси, улазни сигнали у систем управљања нису познати унапред, већ су по својој природи случајни, и тренутни вектор улаза се не може изразити аналитички.

На пример, у радарском систему противваздушних ракетних система, позиција и брзина циља који се прати може да се мења на непредвидив начин, тако да се не може унапред одредити. Ово поставља проблем пројектанту управљачког система, јер је тешко пројектовати систем управљања тако да се он понаша на задовољавајући начин на све могуће облике улазних сигнала.

У анализи система управљања неопходно је да постоји основа за поређење перформанси различитих система управљања. Ова основа може да се састоји у специфицирању одређених *типичних*, *тест* улазних сигнала и поређењу одзива различитих система на ове улазне сигнале. Употреба тест

услова, је *временски одзив система*, или краће *одзив система.* У математичком смислу, одзив система представља решење диференцијалне једначине понашања система. С обзиром на то да су УИ системи описани линеарним диференцијалним једначинама са константним коефицијентима, одзив система се може одредити применом методе Лапласове трансформације. Поступак би се могао приказати на следећи начин:

1. диференцијална једначина се трансформише у комплексни домен променљиве s применом Лапласове трансформације,

2. алгебарска једначина у s домену се реши по излазној променљивој, то јест одреди се њен комплексни лик,

3. уколико је потребно, комплексни лик излазне променљиве се развија у суму парцијалних разломака (Хевисајдов развој),

4. одзив се добија применом инверзне Лапласове трансформације помоћу таблица Лапласових трансформација.

Пример 19.1. *Одредити одзив система описаног диференцијалном једначином*

$$\ddot{x}_i(t) + 3\dot{x}_i(t) + 2x_i(t) = 5x_{iz}(t), \quad \textit{за} \quad x_{iz}(t) = h(t),\, x_i(0^-) = -1,\, \dot{x}_i(0^-) = 2.$$

Решење: Применом Лапласове трансформације на диференцијалну једначину понашања система добија се

$$s^2X_i^-(s) - sx_i(0) - \dot{x}_i(0) + 3sX_i^-(s) - 3x_i(0) + 2X_i^-(s) = \frac{5}{s},$$

при чему је искоришћена чињеница да је $\mathcal{L}\{h(t)\} = \dfrac{1}{s}$.

Заменом почетних услова у последњу једначину и њеним решавањем по излазној променљивој $X_i^-(s)$ добија се

$$X_i^-(s) = \frac{-s^2 - s + 5}{s(s^2 + 3s + 2)} = \frac{-s^2 - s + 5}{s(s+1)(s+2)}.$$

Рационална функција $X_i(s)$ представљена је количником два полинома

$$X_i^-(s) = \frac{\varphi(s)}{f(s)}, \quad \text{где су} \quad \varphi(s) = -s^2 - s + 5, \quad f(s) = s^3 + 3s^2 + 2s.$$

Полови функције $X_i^-(s)$ су $s_1^* = 0$, $s_2^* = -1$ и $s_3^* = -2$ и очито је да су они једноструки, па је њен Хевисајдов развој на парцијалне разломке

$$X_i^-(s) = R_0 + \frac{R_1}{s} + \frac{R_2}{s+1} + \frac{R_3}{s+2}.$$

С обзиром да је $f'(s) = \dfrac{df(s)}{ds} = 3s^2 + 6s + 2$, резидијуми у половима су:

$$R_1 = \frac{\varphi(s)}{f'(s)}\Big|_{s=s_1^*} = \frac{-s^2 - s + 5}{3s^2 + 6s + 2}\Big|_{s=0} = \frac{5}{2},$$
$$R_2 = \frac{\varphi(s)}{f'(s)}\Big|_{s=s_2^*} = \frac{-s^2 - s + 5}{3s^2 + 6s + 2}\Big|_{s=-1} = -5,$$
$$R_3 = \frac{\varphi(s)}{f'(s)}\Big|_{s=s_3^*} = \frac{-s^2 - s + 5}{3s^2 + 6s + 2}\Big|_{s=-2} = \frac{3}{2}.$$

Како је степен полинома $\varphi(s)$ мањи од степена полинома $f(s)$, то је $R_0 = 0$. Коначно,

$$X_i^-(s) = \frac{5}{2s} - \frac{5}{s+1} + \frac{3}{2(s+2)}.$$

Применом инверзне Лапласове трансформације на последњу једначину добија се

$$x_i(t) = \mathcal{L}^{-1}\left\{X_i^-(s)\right\} = \mathcal{L}^{-1}\left\{\frac{5}{2s} - \frac{5}{s+1} + \frac{3}{2(s+2)}\right\}.$$

Коришћењем особине Лапласове трансформације да је линеарни оператор, добија се одзив посматраног система:

$$x_i(t) = \mathcal{L}^{-1}\left\{\frac{5}{2s}\right\} - \mathcal{L}^{-1}\left\{\frac{5}{s+1}\right\} + \mathcal{L}^{-1}\left\{\frac{3}{2(s+2)}\right\} =$$
$$\left[\frac{5}{2} - 5e^{-t} + \frac{3}{2}e^{-2t}\right] h(t), \quad t \geq 0^+.$$

Скрипт којим се у Матлабу врши растављање на парцијалне разломке дат је у наставку.

```
syms s
num = [-1 -1 5]; den = [1 3 2 0];
[r,p,k] = residue(num,den)
Xis = r(1)/(s-p(1)) + r(2)/(s-p(2)) + r(3)/(s-p(3))
xi = ilaplace(Xis)
```

Добијени резултати из командног прозора Матлаба су:

```
R =
    1.5000
   -5.0000
    2.5000
P =
    -2
    -1
     0
K =
     []
```

Облици позива као и значење параметара у њима су идентични онима у позиву функције `step()`, те стога овде неће бити понављани.

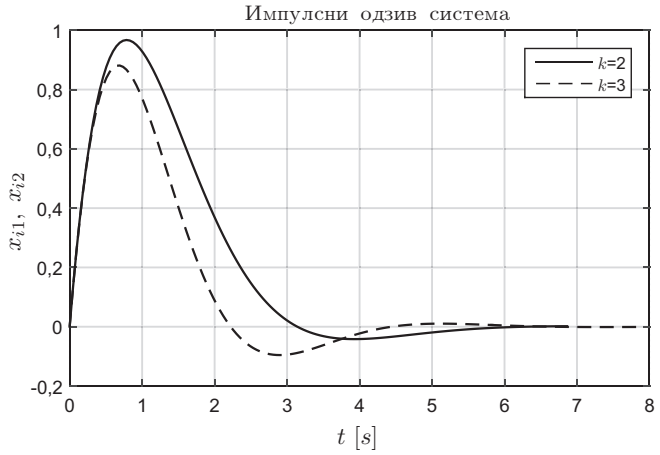
Пример 19.9. *Одредити импулсни одзив система*

$$\ddot{x}_i + 2\dot{x}_i + kx_i = 3x_u, \quad x_i(0^-) = 0,\ \dot{x}_i(0^-) = 0$$

за $k = 2$ *и* $k = 3$. *Импулсне одзиве приказати на истом графику.*

Решење: С обзиром на то да су дати нулти почетни услови, може се искористити функција `impulse()` за добијање импулсног одзива. У наставку је кратак скрипт којим се добија решење приказано на слици 19.10.

```
num = 3; den1 = [1 2 2]; den2 = [1 2 3];
sys1 = tf(num,den1); sys2 = tf(num,den2);
impulse(sys1,'-b',sys2,'--r')
title('Impulsni odziv sistema'), legend('k=2','k=3'), grid
xlabel('$t [s]$','Interpreter','Latex')
ylabel('$x_{i1}, x_{i2}$','Interpreter','Latex')
```



Слика 19.10: Импулсни одзив система из примера 19.9

19.4.3 Одређивање одзива система на произвољан улаз

За добијање одзива система на произвољну промену улаза може се користити функција `lsim()`, чији је основни облик позива:

$$\texttt{lsim}(sys, \mathbf{u}, \mathbf{t})$$

Оваквим позивом, без излазних аргумената, функција графички приказује одзив система. Вектор $\mathbf{t}$ представља вектор временски еквидистантних тренутака симулације, облика $\mathbf{t} = 0 : dt : t_f$, где је t_f крајње време симулације. Улаз $\mathbf{u}$ је низ који има онолико врста колико временских тренутака

$$\texttt{initial}(sys,\mathbf{x}_0,\mathbf{t})$$

када користи унапред дефинисани вектор временских тренутака $\mathbf{t}$. У свим наведеним случајевима, без излазних аргумената, функција генерише графички приказ одзива. Додавањем повратних вредности функције:

$$[\mathbf{x}_i,\mathbf{t},\mathbf{x}]=\texttt{initial}(...)$$

за све претходне случајеве, функција враћа вектор излаза $\mathbf{x}_i$ и вектор стања $\mathbf{x}$, као и сам вектор временских тренутака $\mathbf{t}$. Тада је за добијање графичког приказа неопходно користити функцију `plot()`. Треба напоменути да, с обзиром да се користи за модел описан у простору стања, дакле LTI објект типа SS, то функција дозвољава и да се позове директно са матрицама A, B, C и D, на пример као:

$$[\mathbf{x}_i,\mathbf{t},\mathbf{x}]=\texttt{initial}(A,B,C,D,\mathbf{x}_0)$$

Пример 19.13. *За систем описан диференцијалном једначином понашања*

$$\dddot{x}_i + 7\ddot{x}_i + 14\dot{x}_i + 8x_i = 5x_u$$

одредити модел у простору стања. За $x_u(t) = 0$ *и* $x_i(0) = 2$, $\dot{x}_i(0) = 1$ *и* $\ddot{x}_i(0) = 0,5$ *одредити кретање и одзив система.*

Решење: Избором фазних величина стања $x_1 = x_i$, $x_2 = \dot{x}_i$ и $x_3 = \ddot{x}_i$ добијају се скаларне једначине стања и једначина излаза:

$$\begin{aligned}
\dot{x}_1 &= x_2,\\
\dot{x}_2 &= x_3,\\
\dot{x}_3 &= -8x_1 - 14x_2 - 7x_3 + 5x_u,\\
x_i &= x_1.
\end{aligned}$$

Како се захтева кретање и одзив система у слободном радном режиму (за $x_u(t) = 0$), претходне једначине које описују систем се своде на

$$\begin{aligned}
\dot{x}_1 &= x_2,\\
\dot{x}_2 &= x_3,\\
\dot{x}_3 &= -8x_1 - 14x_2 - 7x_3,\\
x_i &= x_1,
\end{aligned}$$

одакле су матрице

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -8 & -14 & -7 \end{bmatrix},\ B = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},\ C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix},\ D = 0,$$

а почетни услови су $x_1(0) = 2$, $x_2(0) = 1$ и $x_3(0) = 0,5$. Следећи скрипт се може искористити за одређивање кретања и одзива система у Матлабу.