Kazalo

1 Uvod

Matlab je interaktivno programsko orodje, namenjeno predvsem za numerične izračune. Matematik Cleve Moler je napisal programe za prvo verzijo Matlaba v programskem jeziku Fortran v poznih 70-tih letih, kot pripomoček za poučevanje numeričnih metod. Pozneje so kodo prepisali v programski jezik C. Zaradi enostavne uporabe je Matlab postal orodje, ki so ga začeli na široko uporabljati. Dandanašnji se uporablja tako v industriji, kot na univerzah. Matlab ima nekaj značilnih prednosti pred drugimi numeričnimi knjižnicamii:

⋆ Omogoča hitro in enostavno pisanje programov.

⋆ Na razpolago so kakovostna orodja za vizualizacijo.

⋆ Program teče na več operacijskih sistemih. Njegove `m`-datoteke so neodvisne od operacijskega sistema.

⋆ V zadnjem času so dodali tudi Maplevo jedro in s tem omogočili, da lahko, podobno kot v Mathematici, v Matlabu računamo simbolično.

⋆ Na razpolago so posebni programski paketi za posamezna področja, kot na primer procesiranje signalov, slik, simbolično računanje, statistiko itd.

⋆ Na internetu lahko dobimo, brezplačno, mnogo programov za Matlab, ki jih prispevajo posamezni uporabniki.

⋆ Večina raziskovalcev na področju nmerične linearne algebre dovoljuje vgrajevanje najnovejših algoritmov v Matlab. Tako, da se v Matlabu nahajajo zadnja dognanja na tem področju.

Matlab je modereno programsko orodje za reševanje numeričnih problemov. Je primeren za pouk, raziskovanje in za reševanje praktičnih problemov.

Matlabov jezik ima bogate podatkovne strukture in je tudi objektno orientiran. Ker Matlab svoje datoteke interpretira, se pri tem izgubi precej dragocenega časa, vendar je mogoče kodo `m`-datotek prevesti in s tem odločno pospešiti izvajanje programa. Po drugi strani lahko ozka grla izvajanja programa, to so tisti deli, ki porabijo največ časa, zakodiramo v kakšen drug programski jezik, na primer C ali Fortran in prevedeno v `mex`-datoteko, ki jo Matlab zna uporabljati tako kot svojo lastno `m`-datoteko, le da prevedena koda teče mnogo hitreje.

2 UNIX

V primeru, ko zaganjamo Matlab v operacijskem sistemu UNIX, moramo vedeti nekaj o tem operacijskem sistemu.

Ukazi sistema UNIX	
`username`	uporabniško ime
`passwd`	geslo
`$`	začetek ukazne vrstice
`.`	trenutno vozlišče
`..`	eno vozlišče višje
`cd`	menjava vozlišč
`/`	ime najvišjega vozlišča
`pwd`	pot do trenutnega vozlišča
`mkdir`	nova mapa
`rmdir`	brisanje mape
`ls`	vsebina mape
`mv`	preimenovanje
`cp`	kopiranje
`rm`	brisanje
`man`	pomoč

Na začetku se moramo prijaviti v sistem. Sistem zahteva, da vpišemo uporabniško ime `username` in geslo `password`. Ko uspešno vpišemo ta dva podatka, se javi namizje sistema. Ko se odpremo terminalsko okno, je na začetku ukazne vrstice običajno izpisan znak `$`. Znajdemo se v korenu domačega drevesa vozlišč. V vsakem vozlišču se nahaja mapa, ki lahko vsebuje datoteke in druge mape. Za lažjo predstavo drevesa vozlišč rečemo, da mapa ustreza veji, medtem ko so datoteke listi drevesa. Iz korena (debla) domačega drevesa moramo priti do delovne mape. V delovni mapi bomo shranjevali programske in podatkovne datoteke. Po drevesu se pomikamo z ukazom `cd` *(change directory)*. Da ne pišemo celotne poti, uporabljamo relativna imena. Ime mape v vozlišču, kjer se trenutno se nahajamo, je `.`, ime mape v vozlišču neposredno nad njim je `..`, ime mape najvišjega domačega vozlišča je `~` medtem ko je ime korena sistema `/`. Če želimo priti iz mape `/home/users/studenti/vaje` v mapo `/home/users/studenti/vaje/nummet/MojaMapa` zadošča, da zapišemo samo relativno pot:

Unix
```
$ cd nummet/MojaMapa
```

Šele potem, ko se prepričamo, da se nahajamo v delovni mapi, začnemo delati. Trenutni položaj v drevesu vozlišč nam izda ukaz `pwd`:

Unix
```
$ pwd
/home/users/studenti/vaje/nummet/MojaMapa
```

Z njim dobimo izpis celotne poti od korena sistema do mape, v kateri se trenutno nahajamo. Z ukazom

Unix
```
$ cd ..
```

se bomo vrnili v mapo neposredno nad trenutno. Zapišimo zaporedje ukazov

Unix
```
$ pwd
/home/users/studenti/vaje/nummet/MojaMapa
$ cd ..
$ pwd
/home/users/studenti/vaje/nummet
```

Novo mapo odpremo z ukazom `mkdir` *(make directory)*. Postavimo se na vozlišče od koder bomo nadaljevali drevo z novo mapo, vejo. Na primer

Unix
```
$ cd /home/users/studenti/vaje/nummet
$ pwd
/home/users/studenti/vaje/nummet
$ mkdir NovaMapa
$ ls
NovaMapa        MojaMapa
```

Ukaz `ls` *(list)* izpiše vsebino mape kjer se trenutno nahajamo. Tako se prepričajmo, da smo novo mapo uspešno vstavili na pravo mesto. Lahko zahtevamo bolj selektivni izpis vsebine mape, če zapišemo na primer:

Unix
```
$ ls N*
NovaMapa
```

se bodo izpisala samo imena map, ki se začnejo z veliko črko `N`. Lahko pa zapišemo tudi

Unix
```
$ ls NovaMapa
NovaMapa
$ ls StaraMapa
ls: StaraMapa: No such file or directory
```

Na koncu se je izpisalo opozorilo da `StaraMapa` ne obstaja več. Če želimo odstraniti prazno mapo, storimo to z ukazom `rmdir` *(remove directory)*. Najprej se postavimo v vozlišče, kjer se nahaja, nato pa izvedemo zaporedje ukazov:

Unix
```
$ ls NovaMapa
NovaMapa
$ rmdir NovaMapa
$ ls NovaMapa
ls: NovaMapa: No such file or directory
```

Ime mape ali datoteke speremenimo z ukazom `mv` *(move)*. Postavimo se v vozlišče kjer se nahaja dana mapa ali datoteka in zapišemo

Unix
```
$ ls
MojaMapa ...
$ mv MojaMapa TvojaMapa
$ ls
TvojaMapa ...
```

Z ukazom `ls` ugotovimo stanje pred, oziroma po spremembi imena, in se prepričamo, da je preimenovanje uspešno izpeljano. Ukaz `cp` služi za kopiranje vsebine map oziroma datotek. Na primer, z zaporedjem ukazov

Unix
```
$ cp TvojaMapa MojaMapa
$ ls
MojaMapa TvojaMapa ....
```

dobimo kopijo mape `TvojaMapa` z vso vsebino vred z novim imenom `MojaMapa`. Podobno prepisujemo datoteke. Z `ls` se prepričamo, da se v trenutni mapi nahajata obe datoteki.

Če želimo izvedeti kaj več o uporabi posameznega ukaza, uporabimo ukaz `man` (okrajšava angleške besede *manual* priročnik). Poglejmo primer. Poglejmo kaj nam bo `man` povedal o uporabi `cp`.

Unix
```
$ man cp
CP(1)                          User Commands                          CP(1)

NAME
       cp - copy files and directories

SYNOPSIS
       cp [OPTION]... SOURCE DEST
       cp [OPTION]... SOURCE... DIRECTORY
       cp [OPTION]... --target-directory=DIRECTORY SOURCE...

DESCRIPTION
       Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.
       ....
```

Na zaslon se nam izpiše izčrpna razlaga kako se dan ukaz uporablja.

Kako uporabljamo disketno enoto Sistem `UNIX` ne loči med disketno enoto, diskovnimi enotami, cdromi in ostalimi mapami v drevesu vozlišč. Običajno se nahaja mapa, na katerero je pripeta vsebina diskete v disketni enoti na `/mnt/floppy`. Ko vstavimo disketo moramo disketno enoto prilepiti na drevo. Disketno enoto moramo ponovno odlepiti in znova prilepiti na drevo vsakokrat ko zamenjamo disketo. To počneta ukaza `mount` in `umount`. Da bi se izognili uporabi ukazov `mount` in `unmount`, vsakokrat ko zamenjamo desketo, uporabljamo sveženj programov, ki delajo z disketno enoto podobno kot v operacijskem sistemu DOS. Ti ukazi obidejo sistemski `mount` in `umount`.

Delo z disketno enoto

`mdir`	vsebina diskete
`mcd`	sprememba mape na disketi
`mcopy`	prepisovanje iz/na disketo
`mdel`	brisanje datotek na disketi

Ker je koda za prelom vrstice v operacijskem sistemu `UNIX` `0A` v operacijskem sistemu `DOS/WINDOWS` pa `0A 0D` imamo težave, če želimo datoteko, ki smo jo zapisali v operacijskem sistemu `UNIX`, prebrati v operacijskem sistemu `DOS/WINDOWS`. Temu se izognemo, če tekstovne datoteke prepisujemo z ukazom `mcopy` z opcijo `-t`. Vzemimo na primer, da hočemo prepisati datoteko `MojaDatoteka`, ki se nahaja v trenutni mapi, na disketo. To naredimo z ukazom:

Unix
```
$ mcopy -t MojaDatoteka a:MojaDatoteka
```

Če želimo prepisati datoteko iz diskete v trenutno mapo, potem zapišemo:

Unix
```
$ mcopy -t a:MojaDatoteka
```

Več o uporabi ukazov izvemo z `man`, ki mu sledi ime ukaza. Zapišite na primer v unixovo komandno vrstico `man mcopy` in poglejte, kaj je treba storiti, da se celotna vsebina diskete z drevesom vozlišč prepiše iz diskete v trenutno mapo. Priporočamo uporabo teh ukazov namesto originalnega `UNIX`-ovega ukaza `mount` in `umount` tudi zato, ker ti ohranijo dolga imena datotek operacijskega sistema `WINDOWS`.

3 Matlab

Začetek	
`matlab`	vstop v Matlab
`help`	pomoč
`lookfor`	ključna beseda
`format`	oblikovanje izpisa
`who`	informacija o spremenljivkah
`clear`	brisanje spremenljivk
`quit`	izhod iz Matlaba

Najučinkovitejši način učenja je neposredno preizkušanje ukazov in opazovanje njihovega odziva.

V tem razdelku si bomo ogledali nekaj osnovnih reči, kot so vpisovanje ukazov v ukazno vrstico, inicializacijo, ažurinaje, brisanje in shranjevanje spremenljivk.

Zagon Matlaba: Najprej si poglejmo nekaj najosnovnejših napotkov za delo z Matlabom.

Najprej odpremo terminalsko okno v operacijski sistem. Terminal ponudi ukazno vrstico, ki se v operacijskem sistemu `UNIX` začne običajno z znakom `$`. V ukazno vrstico sistema vpišemo ime programa `matlab` in pritisnemo tipko $\langle\hookleftarrow\rangle$. Program lahko zaženemo tudi s klikom miške na ustrezno ikono. Program odpre svoje terminalsko okno, ki ponudi ukazno vrstico. Ta se začne, za razliko od sistemske, z znakom `<<`.

Z Matlabom komuniciramo tako, da vpišemo ukaz v komandno vrstico in pritisnemo tipko $\langle\hookleftarrow\rangle$. Ta se bo odzval z izpisom rezultatov v naslednjih vrsticah terminala. Loči male in velike črke. Spremenljivki priredimo vrednost tako, da vpišemo v ukazno vrstico njeno ime, ki mu sledi prireditveni znak `=` in ustrezna vrednost. Ko pritisnemo tipko $\langle\hookleftarrow\rangle$ se izvrši prireditev. Če želimo pozneje izpisati trenutno vrednost spremenljivke, vpišemo njeno ime in pritisnemo tipko $\langle\hookleftarrow\rangle$. V naslednjih vrsticah se bo izpisala njena vrednost. Pri vsakem prirejanju spremenljivk se bo rezultat izpisal na zaslon. Sprotne izpise preprečimo tako, da postavimo na koncu ukazne vrstice `;` (podpičje).

Matlab uporablja par okroglih `( )` in par oglatih `[ ]` oklepajev, ki pa seveda nista ekvivalentna. Okrogli so namenjeni za grupiranje v aritmetičnih izrazih, funkcijske argumente in indekse polj, medtem ko so oglati rezervirani za inicializacijo matrik. Puščici navzgor in navzdol se uporabljata za sprehod skozi zgodovino ukazov. Star ukaz lahko ponovno izvedemo, ko se s pomočjo puščic prebijemo do njega, in pritisnemo tipko $\langle\hookleftarrow\rangle$.

Za pomoč služi ukaz `help`. V ukazno vrstico zapišemo `help`, ki mu sledi tema, po kateri sprašujemo. Če ne vemo natančno imena funkcije ali ukaza, si lahko pomagamo z ukazom `lookfor`, ki mu sledi ključna beseda. Matlab bo poiskal vse teme, ki vsebujejo v razlagi ustrezno ključno besedo. Matlab zapustimo tako, da vpišemo ukaz `quit`.

Poglejmo tabelo vseh osnovnih tipk, ki se uporabljajo pri urejanju ukazne vrstice.

Urejanje ukazne vrstice	
⟨↩⟩, ⟨Enter⟩	zaključek ukazne vrstice
⟨↑⟩	priklic predhodnje vrstice
⟨↓⟩	priklic naslednje vrstice
⟨←⟩	pomik za znak v levo
⟨→⟩	pomik za zank v desno
⟨Ctrl⟩⟨←⟩	pomik za eno besedo v levo
⟨Ctrl⟩⟨→⟩	pomik za eno besedo v desno
⟨Home⟩	na začetek vrstice
⟨End⟩	na konec vrstice
⟨Esc⟩	brisanje vrstice
⟨Del⟩	brisanje znaka pod utripačem
⟨Bs⟩, ⟨←—⟩	brisanje predhodnjega znaka

Ko vpisujemo ukaze v ukazno vrstico in pritisnemo tipko ⟨↩⟩, se rezultat izpiše na zaslon in vpiše v spremenljivko `ans`.

Računanje in spremenljivke

Primer 3.1. *Ukazno vrstico lahko uporabljamo kot kalkulator.*

Matlab
```
>> 2+sin(10)+3*6^2

ans =

  109.4560

>>
```

Primer 3.2. *Če želimo izračunano vrednost shraniti, jo priredimo spremenljivki.*

Matlab
```
>> x=3, a=2;

x =

    3

>> y=3*x+7

y =

    16

>>
```

Zberimo nekaj najpomembnejših pravil, ki smo jih omenili.

⋆ Ime spremenljivke je lahko sestavljeno iz velikih in malih črk, številskih znakov in podčrtajev. Naj se ne začne s številskim znakom ali podčrtajem.

⋆ Matlab loči velike in male črke.

⋆ Spremenljivki priredimo vrednost s prireditvenim operatorjem `=` .

⋆ Če želimo v ukazni vrstici zapisati več ukazov, jih ločimo z vejico `,` ali s podpičjem `;` .

⋆ Če zaključimo ukazno vrstico s podpičjem, preprečimo izpis rezultatov.

⋆ Če rezultata ne priredimo nobeni spremenljivki, se bo shranil v spremenljivko `ans`. Če ga želimo ohraniti, moramo njeno vsebino shraniti, ker se bo vrednost le-te slej ko prej prepisala.

Urejanje pomnilnika

⋆ `who, whos` Vsebino pomnilnika pregledamo s pomočjo ukazov `who` oziroma `whos`. Prvi samo našteje spremenljivke, drugi pa pove tudi njen tip in koliko pomnilnika zaseda.

Poglejmo vsebino pomnilnika potem ko smo izvedli ukaze iz primerov 3.1 in 3.2.

Matlab

```
>> who

Your variables are:

a    ans  x    y

>> whos
  Name      Size         Bytes  Class

    a         1x1             8  double array
    ans       1x1             8  double array
    x         1x1             8  double array
    y         1x1             8  double array

Grand total is 4 elements using 32 bytes

>>
```

⋆ `clear` Spremenljivke zbrišemo z ukazom `clear`, ki mu sledijo imena spremenljivk, ki jih želimo pozabiti, ločena s presledkom. Če izpišemo samo `clear`, potem bomo izgubili vse spremenljivke. Nadaljujemo s stanjem pomnilnika, kot je bilo v prejšnjem primeru. Zapišimo:

Matlab

```
>> clear a x
>> who

Your variables are:

ans  y

>> clear
>> who
>>
```

Konstante so v naprej inicializirane spremenljivke.

Konstante	
`pi`	število π
`i, j`	imaginarna enota $\sqrt{-1}$

V primeru ko ime konstante uporabimo za spremenljivko tako, da ji priredimo novo vrednost, bomo njeno prvotno vrednost izgubili. Staro vrednost dobimo nazaj tako, da zbrišemo spremenljivko, ki se skriva za imenom konstante.

Matlab

```
>> format long
>> pi

ans =

   3.14159265358979

>> pi = 3

pi =

     3

>> clear pi
>> pi

ans =

   3.14159265358979

>> format
>> a=3+i

a =

   3.0000 + 1.0000i

>> whos
  Name      Size         Bytes  Class

  a         1x1             16  double array (complex)
  ans       1x1              8  double array

Grand total is 2 elements using 24 bytes

>> i=3

i =

     3

>> a=3+i

a =
```

```
     6

>> whos
  Name      Size         Bytes  Class

  a         1x1              8  double array
  ans       1x1              8  double array
  i         1x1              8  double array

Grand total is 3 elements using 24 bytes

>> clear i
>> a=3+i

a =

   3.0000 + 1.0000i

>> whos
  Name      Size         Bytes  Class

  a         1x1             16  double array (complex)
  ans       1x1              8  double array

Grand total is 2 elements using 24 bytes

>>
```

Aritmetika IEEE

Konstante odvisne od platforme	
`isieee`	aritmetika IEEE ?
`computer`	tip računalnika
`eps`	razdalja od 1.0 do naslednjega predstavljivega števila v IEEE aritmetiki
`realmax`	največje predstavljivo število v tej aritmetiki
`realmin`	najmanjše predstavljivo pozitivno število
`inf`	neskončno
`NaN`	ni število (not a number)

Platforme, na katerih teče Matlab imajo večinoma vgrajeno aritmetiko s plavajočo piko, ki se podreja `IEEE` standardom. Matlabova aritmetika dvojne natančnosti je aritmetika s plavajočo piko in 64 bitov dolgo besedo. Območje aritmetike je približno od -10^{-308} do 10^{308} . V tej aritmetiki ne moremo predstaviti vseh realnih števil, ker je dolžina besede končna. Tista realna števila, ki jih lahko predstavimo, bomo imenovali *predstavljiva števila*.

⋆ `isieee` Logična funkcija `isieee` vrne rezultat 1, če imamo opraviti z aritmetiko tipa `IEEE` in vrednost 0, če ne. Od verzije Matlaba 6 naprej ta funkcija vedno vrne vrednost 1, to pomeni, da bo od verzije 6 naprej bo tekel Matlab samo na platformah z `IEEE` aritmetiko.

⋆ `computer` Funkcija `computer` vrne ime operacijskega sistema, na katerem se trenutno zaganja Matlab.

Matlab
```
>> isieee

ans =

     1

>> computer

ans =

LNX86

>>
```

⋆ `realmax` Število, ki ga vrne funkcija `realmax`, je največje predstavljivo realno število v dani aritmetiki.

⋆ `inf` Če je rezultat računske operacije večji od števila, ki ga vrne funkcija `realmax`, potem Matlab rezultatu priredi vrednost `inf`, neskončno. Podobno velja za vrednost izraza, ki je manjša od `-realmax`, temu se priredi vrednost `-inf`. Poglejmo primere:

Matlab
```
>> realmax

ans =

  1.7977e+308

>> -1.2*realmax

ans =

  -Inf

>> 2*realmax

ans =

   Inf

>>
```

⋆ `realmin` Konstanta `realmin` vrne najmanjše predstavljivo pozitivno število.

⋆ `NaN` Nedoločenemu izrazu Matlab priredi konstanto `NaN`, kratica za *not a number*, kar pomeni, da ni število. Rezultat računa, ki vsebuje vsaj en člen `NaN`, je vedno `NaN`. Rezultat logičnega izraza, ki vsebuje vsaj en člen `NaN`, je vedno napačen `false` oziroma 0. Poglejmo primere:

Matlab
```
>> 0/0
Warning: Divide by zero.

ans =
```

```
   NaN
>> inf/inf
ans =
   NaN
>> inf-inf
ans =
   NaN
>> NaN-NaN
ans =
   NaN
>> 0*NaN
ans =
   NaN
>> sin(3)*3/5 + NaN/6
ans =
   NaN
>> NaN==NaN
ans =
   0
>>
```

⋆ `eps` Konstanata `eps` je razdalja med številom 1 in naslednjim predstavljivim številom na realni premici. Če prištejemo številu 1 število `eps`, dobimo predstavljivo število in `1+eps` bo predstavljeno s številom različnim od 1.

Matlab

```
>> eps
ans =
   2.2204e-16
>> a=1+eps
a =
```

```
    1.0000
>> a-1
ans =
   2.2204e-16
>>
```

Če pa prištejemo številu 1 število `eps/2`, potem to število ni predstavljivo in rezultat bo predstavljen s s"tevilom 1.

Matlab

```
>> a=1+eps/2
a =
     1
>> a-1
ans =
     0
>>
```

Posledica tega je, da v aritmetiki s plavajočo piko končne natančnosti ne veljata zakona asociativnosti in komutativnosti. Napako pri računanju lahko radikalno zmanjšamo, če izberemo primerno zaporedje operacij.

Primer 3.3. *Poglejmo naslednja dva primera*

Matlab

```
>> 1+eps/2-1
ans =
     0
>> 1-1+eps/2
ans =
   1.1102e-16
>>
```

Vzemimo naslednji izraz

$$1 + \frac{eps}{2} + \frac{eps}{3} + \frac{eps}{4}$$

Če seštevamo v vrstnem redu kot je izraz zapisan, dobimo rezultat identično enak ena, če pa seštevamo v obratnem vrstnem redu, pa dobimo rezultat večji od 1.

Matlab

```
>> x=1+eps/2+eps/3+eps/4

x =

     1

>> x-1

ans =

     0

>> y=eps/4+eps/3+eps/2+1

y =

    1.0000

>> y-1

ans =

   2.2204e-16

>>
```

Pomoč v Matlabu Poglejmo, kako uporabljamo pomoč v Matlabu.

⋆ `help` Ko zapišemo v ukazno vrstico `help`, dobimo naslednji izpis:

Matlab

```
>> help

HELP topics:

matlab/general       -  General purpose commands.
matlab/ops           -  Operators and special characters.
matlab/lang          -  Programming language constructs.
matlab/elmat         -  Elementary matrices and matrix manipulation.
matlab/elfun         -  Elementary math functions.
matlab/specfun       -  Specialized math functions.
matlab/matfun        -  Matrix functions - numerical linear algebra.
matlab/datafun       -  Data analysis and Fourier transforms.
matlab/audio         -  Audio support.
matlab/polyfun       -  Interpolation and polynomials.
matlab/funfun        -  Function functions and ODE solvers.
matlab/sparfun       -  Sparse matrices.
matlab/graph2d       -  Two dimensional graphs.
matlab/graph3d       -  Three dimensional graphs.
matlab/specgraph     -  Specialized graphs.
```

```
matlab/graphics      -  Handle Graphics.
matlab/uitools       -  Graphical user interface tools.
matlab/strfun        -  Character strings.
matlab/iofun         -  File input/output.
matlab/timefun       -  Time and dates.
matlab/datatypes     -  Data types and structures.
matlab/verctrl       -  Version control.
matlab/demos         -  Examples and demonstrations.
...                  -  ...

>>
```

To je spisek tem po katerih se lahko vprašamo. Če se vprašamo po neki določeni temi, dobimo spisek vseh njenih funkcij in ukazov. Izberimo na primer `elfun`, elementarne funkcije Matlaba.

Matlab

```
>> help elfun

  Elementary math functions.

  Trigonometric.
    sin         - Sine.
    sinh        - Hyperbolic sine.
    asin        - Inverse sine.
    asinh       - Inverse hyperbolic sine.
    cos         - Cosine.
    cosh        - Hyperbolic cosine.
    acos        - Inverse cosine.
    acosh       - Inverse hyperbolic cosine.
    tan         - Tangent.
    tanh        - Hyperbolic tangent.
    atan        - Inverse tangent.
    atan2       - Four quadrant inverse tangent.
    atanh       - Inverse hyperbolic tangent.
    sec         - Secant.
    sech        - Hyperbolic secant.
    asec        - Inverse secant.
    asech       - Inverse hyperbolic secant.
    csc         - Cosecant.
    ...         - ...


>>
```

Zapišimo v ukazno vrstico `help`, ki mu sledi ime ene od funkcij. Na primer:

Matlab

```
>> help atan2

 ATAN2  Four quadrant inverse tangent.
    ATAN2(Y,X) is the four quadrant arctangent of the real parts of the
    elements of X and Y.  -pi <= ATAN2(Y,X) <= pi.

    See also ATAN.

>>
```

Sledil je kratek opis funkcije in spisek argumentov, ki jih sprejema in rezultatov, ki jih vrača. Na koncu nas Matlab opozori s `See also` na druge funkcije, ki so povezane z njo. Vsi ukazi in funkcije Matlaba se pišejo z malimi črkami. Pri izpisu pomoči, pa so vsi ukazi izpisani z velikimi črkami. To posebnost, ki je verjetno zgodovinski preostanek, si moramo dobro zapomniti. Če zapišemo v ukazno vrstico `help ATAN2`, bomo dobili odgovor, da je funkcija `ATAN2` nepoznana. Poglejmo!

Matlab

```
>> help ATAN2

ATAN2.m not found.

>>
```

⋆ `lookfor` Ukaz `loofor`, ki mu sledi ključna beseda, poišče vse tiste funkcije oziroma ukaze, ki v razlagi vsebujejo to besedo. Uporablja se v primeru, ko ne poznamo natančno ime funkcije, ki jo iščemo. Zadostuje, da poznamo eno od ključnih besed v zvezi z njo. Poglejmo na primer, kako poiščemo funkcije, ki so povezane s ključno besedo `atan`.

Matlab

```
>> lookfor atan
ATAN   Inverse tangent.
ATAN2  Four quadrant inverse tangent.
ATANH  Inverse hyperbolic tangent.
HATANO  Hatano Asymmetrical Equal Area Pseudocylindrical Projection
DELTATAN Delta function for TANSIG neurons.
QATAN4 Four-quadrant arctangent. (Utility Function)
ATAN   Symbolic inverse tangent.
ATANH  Symbolic inverse hyperbolic tangent.
BLKATAN This block defines an output angle that is the arctangent of two inputs.
BLKCOSATAN This block defines the cosine of an angle whose tangent is u1/u2.
BLKSINATAN This block defines the sine of an angle whose tangent is u1/u2.
STRADD Concatanate row strings (MEX file)
>>
```

Oblikovanje izpisa na zaslon . Ukaz `format` skrbi za obliko izpisa na zaslon.

⋆ `format` Na primer `format long` nastavi izpis na več decimalnih mest. Vpište v ukazno vrstico `help format`, če vas zanimajo še druge možnosti oblikovanja izpisa, ki jih ponuja `format`. Ukaz `format` nastavi v naprej določen način izpisa na štiri decimalna mesta, ki ustreza načinu `format short`.

Matlab

```
>> format
>> 1/3

ans =

    0.3333

>> help format

 FORMAT Set output format.
    All computations in MATLAB are done in double precision.
    FORMAT may be used to switch between different output
```

```
    display formats as follows:
      FORMAT         Default. Same as SHORT.
      FORMAT SHORT   Scaled fixed point format with 5 digits.
      FORMAT LONG    Scaled fixed point format with 15 digits.
      FORMAT SHORT E Floating point format with 5 digits.
      FORMAT LONG E  Floating point format with 15 digits.
      FORMAT SHORT G Best of fixed or floating point format with 5 digits.
      FORMAT LONG G  Best of fixed or floating point format with 15 digits.
      FORMAT HEX     Hexadecimal format.
      FORMAT +       The symbols +, - and blank are printed
                     for positive, negative and zero elements.
                     Imaginary parts are ignored.
      FORMAT BANK    Fixed format for dollars and cents.
      FORMAT RAT     Approximation by ratio of small integers.

    Spacing:
      FORMAT COMPACT Suppress extra line-feeds.
      FORMAT LOOSE   Puts the extra line-feeds back in.

>> format long
>> 1/3

ans =

   0.33333333333333

>> format rat
>> 1/3

ans =

     1/3

>> 8/6

ans =

     4/3

>> 0.5

ans =

     1/2

>> pi

ans =

   355/113

>>
```

Ukaz `format rat` nastavi na racionalni izpis vrednosti spremenljivk. Izpiše približne vrednosti spremenljivk v obliki okrajšanega ulomka. Racionalna oblika je samo za izpis podatkov, ne pa za hranjenje. Racionalni izpis spremenljivke ni enakovreden njeni vsebini. Poglejmo primer:

Matlab

```
>> 355/113 - pi

ans =

   2.6676e-07

>>
```

4 Matrike

Najpomembnejša podatkovna struktura v Matlabu je matrika oziroma polje.

Matrike in polja Ločevali bomo dve podatkovni strukturi, matrike in polja. Kakšna kje razlika med njima. V zapisu in predstavitvi v matlabu ni nobene razlike. Razlika je zgolj formalna. Oboje je pravokotna shema realnih oziroma komplesnih števil. Razlikujeta se le v algebrski strukturi definirani nad njima. Algebrska struktura matrik je struktura matrične algebre, medtem ko so operacije nad polji ustrezne operacije nad elementi polj, medtem tem ko pri matrikah ni vedno tako, primer je množenje matrik.

Dolgo časa je bila matrika edina podatkovna struktura Matlaba. Šele novejše verzije Matlaba poznajo tudi druge podatkovne strukture. Matrike in polja so predstavljena kot dvodimenzionalna (pravokotna) shema realnih števil dvojne natančnosti, `double`.

Nove verzije Matlaba poznajo tudi večdimenzionalna polja - tenzorje. Matrika reda $m \times n$ je shema števil razvrščenih v m vrstic in n stolpcev. Skalar je matrika oziroma polje reda 1×1.

Matrike

`[ ]`	matrični oklepaji
`,`	vejica, ločnica med elementi znotraj vrstic
`;`	podpičje, ločnica med vrsticami matrike
`( )`	doseganje posameznih elementov matrike
`size`	red matrike

4.1 Sestavljanje matrik

Matrike lahko vnesemo na več načinov. Najpreprostejši način je vnos matrike neposredno preko ukazne vrstice, vendar je to za matrike z mnogo elementi zelo zamudno opravilo. Nekatere posebne matrike lahko tvorimo s pomočjo Matlabovih funkcij. Matriko lahko preberemo tudi iz datoteke, ki hrani vrednosti njenih elementov. Z datokekami lahko prenašamo podatke med različnimi programi.

Poglejmo načine vnosa matrik pobliže.

Neposreden način vnosa matrik.

Primer 4.1. *Primera neposrednega vnosa matrik preko ukazne vrstice*

Matlab

```
>> A=[1,2,3,4;2,3,4,5]

A =

   1     2     3     4
   2     3     4     5

>> B=[1 2 3;2 3 4;1 1 1]

 B =

      1     2     3
      2     3     4
      1     1     1
```

```
>>
```

Pravilo 4.1. *Elelemente matrike vpisujemo po vrsticah. Znotraj vrstice jih ločujemo z vejico ali presledkom, medtem ko vrstice ločujemo s podpičjem.*

⋆ `size` Ukaz `size` vrne število vrstic in stolpcev matrike.

Preberimo število stolpcev in vrstic matrik iz primera 4.1.

Matlab

```
>> size(A)

ans =

     2     4

>> size(B)

ans =

    3     3
>>
```

Matrika `A` ima dve vrstici in štiri stolpce, medtem ko ima matrika `B` tri vrstice in tri stolpce. V novejših verzijah Matlaba lahko kličemo ukaz `size` z dvema argumentoma. Če je drugi argument enak 1 vrne število vrstic, če je enak 2, pa število stopcev. Vzemimo matriko `A` iz primera 4.1.

Matlab

```
>> size(A,1)

ans =

     2

>> size(A,2)

ans =

    4
>>
```

Matrike lahko sestavljamo. Če imata matriki `A` in `B` enako število vrstic, potem lahko tvorimo matriko, ki ima vrstice matrike `A` podaljšane z vrsticami matrike `B`. V primeru, ko imata matriki enako število stolpcev, lahko tvorimo matriko, ki ima stolpce matrike `A` podaljšane s stolpci matrike `B`. Poglejmo primera:

Matlab

```
>> A=rand(2,3)

A =

    0.9218    0.1763    0.9355
```

```
    0.7382    0.4057    0.9169

>> B=rand(2,3)

B =

    0.4103    0.0579    0.8132
    0.8936    0.3529    0.0099

>> [A,B]

ans =

    0.9218    0.1763    0.9355    0.4103    0.0579    0.8132
    0.7382    0.4057    0.9169    0.8936    0.3529    0.0099

>> [A;B]

ans =

    0.9218    0.1763    0.9355
    0.7382    0.4057    0.9169
    0.4103    0.0579    0.8132
    0.8936    0.3529    0.0099

>>
```

Pravilo 4.2. *Dve ali več matrik z enakim številom vrstic lahko združimo v novo matriko, ki ima vrstice prve matrike podaljšane z vrsticami naslednjih matrik v danem vrstnem redu. Matrike zapišemo v oglati oklepaj in jih ločimo z vejico ali praznim znakom. Podobno lahko združimo dve ali več matrik, ki imajo enako število stolpcev, v novo matriko, ki ima stolpce prve matrike podaljšane s stolpci naslednjih v danem vrstnem redu. V tem primeru ločimo matrike s podpičjem.*

Poglejmo še nekoliko bolj zapleten primer sestavljanja matrik.

Matlab

```
>> A=[1 2 3], B=[4 4], C=[5 6 7]

A =

     1     2     3


B =

     4     4


C =

     5     6     7

>> D=[A,B,C;ones(3,5),eye(3,3)]
```

```
D =

     1     2     3     4     4     5     6     7
     1     1     1     1     1     1     0     0
     1     1     1     1     1     0     1     0
     1     1     1     1     1     0     0     1

>>
```

Vnos matrik s pomočjo funkcij za tvorbo posebnih matrik Nekatere posebne matrike in vektorje sestavimo s pomočjo funkcij, kot so

Posebne matrike	
`eye`	enotna matrika
`zeros`	matrika ničel
`ones`	matrika enic
`rand`	matrika slučajnih vrednosti
`repmat`	matrika s ponavljajočo podmatriko
`blkdiag`	bločno diagionalna matrika
`gallery`	funkcija, ki vrača različne tipe matrik
`linspace`	vektor ekvidistantnih vrednosti
`:`	aritmetično zaporedje

Pravilo 4.3. *Funkcije kot so* `eye`, `zeros`, `ones` *in* `rand` *lahko kličemo z enim ali dvema argumentoma. Če kličemo z enim argumentom, potem dobimo rezultat kvadratno matriko reda, kot ga določa argument, sicer pa dobimo matriko s številom vrstic, kot ga določa prvi argument in številom stolpcev, kot ga določa drugi argument.*

⋆ `eye` Funkcija `eye` vrne enotno matriko. Če matrika, ki jo zahtevamo ni kvadratna, potem dobimo matriko, kjer so vse komponente z enakim indeksom vrstice in stolpca enake 1 ostale pa nič. Poglejmo primere

Matlab

```
>> eye(3)

ans =

     1     0     0
     0     1     0
     0     0     1

>> eye(2,3)

ans =

     1     0     0
     0     1     0

>> eye(3,2)

ans =
```

```
     1     0
     0     1
     0     0

>> eye(2,2)

ans =

     1     0
     0     1

>>
```

⋆ `zeros` Funkcija `zeros` vrne matriko ničel. Poglejmo primera:

Matlab

```
>> zeros(2)

ans =

     0     0
     0     0

>> zeros(3,2)

ans =

     0     0
     0     0
     0     0

>>
```

⋆ `ones` Funkcija `ones` vrne matriko enic ustreznega reda.

⋆ `rand` Funkcija `rand` vrne matriko slučajnih števil. Slučajna števila so porazdeljena enakomerno na intervalu $[0, 1]$. Če želimo sestaviti matriko slučajnih števil enakega reda kot je dana matrika `A` zapišemo `rand(size(A))`, podobno velja tudi za `zeros`, `ones` in `eye`. Poglejmo primer. Tvorimo matriko slučajnih vrednosti enakega reda kot matrika `A`.

Matlab

```
>> A=ones(3,5)

A =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1

>> B=rand(size(A))

B =
```

```
    0.9501    0.4860    0.4565    0.4447    0.9218
    0.2311    0.8913    0.0185    0.6154    0.7382
    0.6068    0.7621    0.8214    0.7919    0.1763

>>
```

⋆ `repmat` Funkcija `repmat` vrne matriko s ponavljajočo se podmatriko. Z `B = repmat(A,m,n)` oziroma `B = repmat(A,[m,n])` dobimo matriko, ki je sestavljena z $m \times n$ kopij matrike `A` v `m` vrsticah in `n` stolpcih.

Matlab

```
>> A=[1 2; 3 4]

A =

     1     2
     3     4

>> B=repmat(A,3,4)

B =

     1     2     1     2     1     2     1     2
     3     4     3     4     3     4     3     4
     1     2     1     2     1     2     1     2
     3     4     3     4     3     4     3     4
     1     2     1     2     1     2     1     2
     3     4     3     4     3     4     3     4

>>
```

Če je `A` skalarna matrika, dobimo matriko enakih vrednosti danega reda. Ta ukaz je mnogo hitrejši kot pa množenje z matriko `ones`.

Matlab

```
>> A=repmat(2,3,4)

A =

     2     2     2     2
     2     2     2     2
     2     2     2     2

>> A=2*ones(3,4)

A =

     2     2     2     2
     2     2     2     2
     2     2     2     2
```

```
>>
```

⋆ `blkdiag` Z `blkdiag` dobimo bločno diagonalne matrike. Naj spregovori primer

Matlab

```
>> A=[1 2 3; 4 5 6]; B=[2 3; 4 5; 3 4]; C=rand(2);
>> D=blkdiag(A,B,C)

D =

    1.0000    2.0000    3.0000         0         0         0         0
    4.0000    5.0000    6.0000         0         0         0         0
         0         0         0    2.0000    3.0000         0         0
         0         0         0    4.0000    5.0000         0         0
         0         0         0    3.0000    4.0000         0         0
         0         0         0         0         0    0.7027    0.4449
         0         0         0         0         0    0.5466    0.6946

>>
```

⋆ `linspace` Funkcija `linspace(a,b,n)` vrne vektor, (enovrstično matriko), katerega prvi element je `a` in zadnji je `b`, med njima pa je vrinjeno $n-2$ ekvidistantnih vrednosti, tako da ima rezultirajoči vektor natanko n elementov. Če želimo sestaviti vektor 5 ekvidistantnih vrednosti, ki se začne z vrednostjo 2 in konča z 11 potem bomo napisali

Matlab

```
>> linspace(2,11,5)

ans =

    2.0000    4.2500    6.5000    8.7500   11.0000

>>
```

⋆ `:` Dvopičje `:` uporabljamo za sestavljanje ekvidistantnih vektorjev, kjer podamo razliko med dvema sosednjima členoma. Uporabljamo ga tudi pri izbiranju vrstic in stolpcev matrik, ter podmatrik. Če zapišemo `a:b`, dobimo aritmetično zaporedje, ki se začne z `a` prirastek, ker ni omenjen, je enak 1, zgornja meja pa je `b`. Če zapišemo, `a:d:b`, smo predpisali še prirastek, ki je v tem primeru enak `d`. Meji in prirastek so realna števila.

Matlab

```
>> a=1:10

a =

     1     2     3     4     5     6     7     8     9    10

>> a=10:-1:2
```

```
a =

    10     9     8     7     6     5     4     3     2

>> a=1:2:10

a =

     1     3     5     7     9

>> a=2:3:10

a =

     2     5     8

>> a=1:-1:10

a =

   Empty matrix: 1-by-0

>> a=2:-1

a =

   Empty matrix: 1-by-0

>> a=2:-1:-1

a =

     2     1     0    -1

>> 0.3:0.2:3.6

ans =

  Columns 1 through 7

    0.3000    0.5000    0.7000    0.9000    1.1000    1.3000    1.5000

  Columns 8 through 14

    1.7000    1.9000    2.1000    2.3000    2.5000    2.7000    2.9000

  Columns 15 through 17

    3.1000    3.3000    3.5000

>>
```

Vnos matrik, katerih vsebina je zapisana na zunanjih datotekah.

Delo z datotekami	
`load`	naloži
`save`	shrani

Če želimo obdelovati podatke, ki smo jih dobili s pomočjo nekega drugega programa, v Matlabu, potem te podatke shranimo na datoteko, ki jo bo Matlab razumel in jo preberemo v Matlabu.

⋆ `load` Datoteka, ki hrani vrednosti matrike je lahko binarna ali tekstovna (ASCII) datoteka. V tekstovni datoteki hrani Matlab elemente matrike, zapisane v ASCII obliki, po vrsticah. Vrstica besedila je vrstica matrike. Posamezni elementi so ločeni s presledkom ali vejico. Vsaka vrstica mora imeti enako število elementov. Sestavimo s pomočjo urejevalnika besedil datoteko z naslednjo vsebino:

txt

```
1 2 12 11 0
3 3 1 2 0
10e-2 12.0  .434e+3 2 -3.0
```

Shranimo vsebino v datoteko z imenom `a.txt`, nato pa zapišemo v ukazno vrstico:

Matlab

```
>> load a.txt
```

Vsebina datoteke se je shranila v matriko `a`. Prikličimo vsebino na zaslon

Matlab

```
>> a

a =

    1.0000    2.0000   12.0000   11.0000         0
    3.0000    3.0000    1.0000    2.0000         0
    0.1000   12.0000  434.0000    2.0000   -3.0000

>>
```

Matlab bo izbral ime spremenljivke s pomočjo imena datoteke, ki je hranila njene vrednosti. Ime datoteke je sestavljeno iz osnove in razširitve *(obrazila)*, ki sta ločena s piko ⟨`name`⟩.⟨`ext`⟩. Ime spremenljivke ⟨`name`⟩ je osnova imena datoteke, v našem primeru je to `a`. Paziti moramo, da je osnova imena datoteke zapisana tako, da lahko postane ime spremenljivke. Torej naj se ne začne s posebnim ali numeričnim znakom, od posebnih znakov naj vsebuje le podčrtaj. Če temu ni tako, bo matlab sam določil ime spremenljivke.

⋆ `save` Z ukazom `save` shranimo vrednosti spremenljivke. Ta ukaz lahko spravi spremenljivko na tekstovno ali binarno datoteko. Če bomo podatke potrebovali za poznejšo obdelavo v Matlabu, je primernejša Matlabova binarna `mat`-datoteka, ki lahko hrani več spremenljivk z njihovimi izvirnimi imeni. Če pa želimo vrednosti shraniti za poznejšo obdelavo v kakšnem drugem programu, na primer v Mathematici, je primerneje, če vrednosti shranimo na tekstovno datoteko. Dobra lastnost tekstovne datoteke je čitljivost. Tekstovno datoteko lahko enostavno popravljamo in prilagajamo v navadnem urejevalniku besedil. Slaba stran pa je, da zavzame mnogo več prostora in, da ena datoteka lahko hrani le eno matriko. Ime spremenljivke ohranimo, tako da ustrezno datoteko imenujemo z njenim imenom in dodamo obrazilo `.txt`. Če zapišemo v ukazno vrstico Matlaba `save`, bomo shranili celotno stanje pomnilnika na

datoteko z imenom `matlab.mat`. To je ugodno v primeru, ko želimo trenutno stanje ohraniti in zapustiti Matlab. Ko ga ponovno zaženemo zapišemo v ukazno vrstico `load` in stanje zapisano na datoteki `matlab.mat` se bo prepisalo nazaj v pomnilnik. V primeru, ko delamo več različnih stvari, bomo shranili vsako pod svojim imenom. Zapisali bomo na primer `save delo19022003`. Stanje se bo shranilo na datoteko z imenom `delo19022003.mat`. Ko želimo z delom nadaljevati, zaženemo Matlab in zapišemo v ukazno vrstico `load delo19022003` in nadaljujemo z delom. Včasih pa ne želimo shraniti celotnega stanja pomnilnika, ampak samo nekatere spremenljivke. To storimo tako, da pri ukazu `save` povemo še imena spremenljivk, ki jih želimo shraniti.

Do sedaj smo shranjevali podatke na binarne datoteke tipa `.mat`. Kako vrednost spremenljivke `a` shranimo na tekstovno datoteko. Če želimo shraniti vsebino spremenljivke `a` na datoteko imenom `ax.txt`, potem postopamo takole

Matlab

```
>> save ax.txt -ascii a
```

Z urejevalnikom besedil, se lahko prepričamo, kako je Matlab shranil vrednosti spremenljivke `a` na datoteko `ax.txt`. Poglejmo si primere shranjevanja podatkov v Matlabu.

Matlab

```
>> a=[1,3]

a =

     1     3

>> b=[3;4;5]

b =

     3
     4
     5

>> who

Your variables are:

a         b

>> save

Saving to: matlab.mat

>> clear
>> who
>> load

Loading from: matlab.mat

>> who

Your variables are:
```

```
a         b

>> c=[3;4;5]

c =

     3
     4
     5

>> who

Your variables are:

a         b         c

>> save ac a c
>> clear
>> who
>> load ac
>> who

Your variables are:

a         c

>> save ax.txt -ascii a
>> clear
>> load ax.txt
>> who

Your variables are:

ax

>> ax

ax =

     1     3

>>
```

Preverimo vsebino ASCII datoteke `ax.txt` s pomočjo urejevalnika besedil. V datoteki najdemo vrstico:

txt

```
   1.0000000e+00   3.0000000e+00
```

4.2 Podmatrike

Posamezne elemente matrike izbiramo tako, da poleg imena matrike, zapišemo v okroglem oklepaju indeksa vrstice in stolpca tega elementa. Indeksa ločimo z vejico. Tako elemente matrike lahko prirejamo

drugim spremenljivkam in jih tudi spreminjamo.

Primer 4.2. *Sestavimo matriko slučajnih vrednosti reda* 3×5. *Element* `A(2,3)` *priredimo spremenljivki* `a` *in ga na koncu nadomestimo s 3;*

Matlab

```
>> A=rand(4,5)

A =

    0.8318    0.3046    0.3028    0.3784    0.4966
    0.5028    0.1897    0.5417    0.8600    0.8998
    0.7095    0.1934    0.1509    0.8537    0.8216
    0.4289    0.6822    0.6979    0.5936    0.6449

>> a=A(2,3)

a =

    0.5417

>> A(2,3)=3

A =

    0.8318    0.3046    0.3028    0.3784    0.4966
    0.5028    0.1897    3.0000    0.8600    0.8998
    0.7095    0.1934    0.1509    0.8537    0.8216
    0.4289    0.6822    0.6979    0.5936    0.6449

>>
```

Kot smo že omenili, lahko operator dvopičje : uporabljamo tudi za izbiranje podmatrik.

Primer 4.3. *Spet prepustimo razlago posebnim primerom*

Matlab

```
>> A=rand(4,7)

A =

    0.8180    0.3412    0.8385    0.5466    0.7948    0.1730    0.8757
    0.6602    0.5341    0.5681    0.4449    0.9568    0.9797    0.7373
    0.3420    0.7271    0.3704    0.6946    0.5226    0.2714    0.1365
    0.2897    0.3093    0.7027    0.6213    0.8801    0.2523    0.0118

>> A(:,2)

ans =

    0.3412
    0.5341
    0.7271
    0.3093
```

```
>> A(2,:)

ans =

    0.6602    0.5341    0.5681    0.4449    0.9568    0.9797    0.7373

>> A(1:2:end,:)

ans =

    0.8180    0.3412    0.8385    0.5466    0.7948    0.1730    0.8757
    0.3420    0.7271    0.3704    0.6946    0.5226    0.2714    0.1365

>> A(end:-1:1,1:2:end)

ans =

    0.2897    0.7027    0.8801    0.0118
    0.3420    0.3704    0.5226    0.1365
    0.6602    0.5681    0.9568    0.7373
    0.8180    0.8385    0.7948    0.8757

>> A(2:4,1:3)

ans =

    0.6602    0.5341    0.5681
    0.3420    0.7271    0.3704
    0.2897    0.3093    0.7027

>>
```

Če trenutno ne hranimo posebej števila vrstic in stolpcev matrike, uporabimo besedico `end`*, ki predstavlja število vrstic oziroma stolpcev, glede na to kje se nahaja.*

Elemente matrike lahko izbiramo tudi posredno.

Matlab

```
>> a=[1 2 4]

a =

     1     2     4

>> b=[3 1 1]

b =

     3     1     1

>> A(a,b)

ans =
```

```
    0.8385    0.8180    0.8180
    0.5681    0.6602    0.6602
    0.7027    0.2897    0.2897

>>
```

Če je `C==A(a,b)`*, potem so elementi matrike C enaki* $C_{ij} = A_{a_i b_j}$*, oziroma je* `c(i,j)==A(a(i),b(j))` *za vse* `i,j`.

4.3 Zapis matrike v pomnilniku

Če izbiramo elemente matrike z enim samim indeksom, potem obravnava Matlab matriko kot vektor. Vrstni red elementov je enak vrstnemu redu kakor so zapisani v pomnilniku. Ker je Matlab otrok Fortrana, je zapis matrike v pomnilniku tak, kot v tem programskem jeziku, to je po stolpcih, ne po vrsticah, kot je v novejših programskih jezikih, na primer Pascal in C.

Primer 4.4. *Poglejmo naslednje primere.*

Matlab

```
>> A=[1 2; 3 4]

A =

     1     2
     3     4

>> A(1)

ans =

     1

>> A(2)

ans =

     3

>> A(3)

ans =

     2

>>
```

Najlepše bomo videli, če zapišemo

Matlab

```
>> A(:)'

ans =

     1     3     2     4
```

```
>>
```

Z `A(:)`*, smo zahtevali vektor vseh elementov matrike v vrstnem redu, kakor so zapisani v pomnilniku.* `A(:)` *je vektor stolpec. Transponirali smo ga* `A(:)'`*, da bi prihranili prostor pri izpisu.*

Če zapišemo `A(B)`*, kjer sta* `A` *in* `B` *matriki, potem mora matrika* `B` *vsebovati samo naravna števila med 1 in številom elementov matrike* `A`*. Rezultat je matrika enakega reda kot je matrika* `B` *z elementi matrike* `A` *v vrstnem redu, kot jih naslavljajo elementi matrike* `B`*.*

Če je `C==A(B)`*, potem so elementi matrike C enaki* $C_{i,j} = A_{B_{ij}}$*, oziroma je izpolnjeno* `C(i,j)==A(B(i,j))` *za vse* `i,j`*. Naj spregovori naslednji primer.*

Matlab

```
>> A=[1,2,3,4;5,6,7,8]

A =

     1     2     3     4
     5     6     7     8

>> B=[1,2;3,4;5,6]

B =

     1     2
     3     4
     5     6

>> A(B)

ans =

     1     5
     2     6
     3     7

>> A(B')

ans =

     1     2     3
     5     6     7

>>
```

Poglejmo nekaj posebnih primerov inicializacije matrik. Kaj se zgodi, če bi zapisali na primer `A(5,5)=1`, ko matrika `A` še ne bi bila inicializirana. Matlab bo zgradil matriko minimalnega reda, tako da bo ustrezen element lahko inicializiran. Vsi ostali elementi so enaki nič. Če pa matrika že obstaja in poskušamo določiti vrednost elementu, ki je zunaj območja indeksov vrstic ali stolpcev matrike, potem bo Matlab razširil prvotno matriko z ničelnimi elementi minimalno tako, da sta dani element in stara matrika vsebovana v novi matriki. Takšen dinamičen način dodeljevanja pomnilnika je lahko nevaren, zato se ga bomo poskušali izogibati, če bo le mogoče. Nevaren je v primeru, če smo pozabili, da je

matrika že inicializirana. V tem primeru matrika lahko vsebuje komponente različne od nič, za katere ne bi vedeli.

Primer 4.5. *Poglejmo primera*

Matlab

```
>> clear
>> A(2,4)=1

A =

     0     0     0     0
     0     0     0     1

>> A(5,6)=5

A =

     0     0     0     0     0     0
     0     0     0     1     0     0
     0     0     0     0     0     0
     0     0     0     0     0     0
     0     0     0     0     0     5

>>

>> A=rand(3,2)

A =

    0.9501    0.4860
    0.2311    0.8913
    0.6068    0.7621

>> A(4,5)=16

A =

    0.9501    0.4860         0         0         0
    0.2311    0.8913         0         0         0
    0.6068    0.7621         0         0         0
         0         0         0         0   16.0000

>>
```

5 Algebrske operacije nad matrikami in polji

Algebrske operacije nad matrikami in polji	
`+`	vsota matrik/polj
`-`	razlika matrik/polj
`*`	produkt matrik
`.*`	produkt polj
`/`	desno deljenje matrik
`./`	desno deljenje polj
`\`	levo deljenje matrik
`.\`	levo deljenje polj
`^`	potenciranje matrik
`.^`	potenciranje polj
`'`	adjungiranje matrik
`.'`	transponiranje matrik in polj
`( )`	določanje vrstnega reda operacij

Za aritmetične operacije veljajo enaki zakoni prednosti, kot smo jih navajeni od drugod oziroma jih pozna večina računalnikov, kalkulatorjev in programskih jezikov. Produkt veže ožje kot vsota ali razlika, deljenje ožje kot produkt in potenciranje veže ožje kot druge operacije. Zaporedna deljenja se izvršijo po vrsti od leve proti desni, zaporedna potenciranja prav tako.

Primer 5.1. *Zapišimo nekaj primerov aritmetičnih izrazov in opazujmo prednost (prioriteto) operacij.*

Matlab

```
>> 1+3*2

ans =

     7

>> (1+3)*2

ans =

     8

>> 1+(3*2)

ans =

     7

>> 3/2*4

ans =

     6

>> 3/(2*4)

ans =
```

```
   0.37500000000000
>> (3/2)*4
ans =
     6
>> 2/3^4
ans =
   0.02469135802469
>> (2/3)^4
ans =
   0.19753086419753
>> 2/(3^4)
ans =
   0.02469135802469
>> 2/3/4
ans =
   0.16666666666667
>> 2/(3/4)
ans =
   2.66666666666667
>> (2/3)/4
ans =
   0.16666666666667
>> 2*3^4/3
ans =
    54
>> 2*3^(4/3)
ans =
```

```
   8.65349742184445

>> 2*(3^4)/3

ans =

    54

>>
```

Naj še enkrat opozorimo, da potenciranje potenc Matlab računa od leve proti desni. Torej, če zapišemo `3^3^3`*, dobimo vrednost* $(3^3)^3$*.*

Matlab

```
>> format rat
>> 3^3^3

ans =

  19683

>> (3^3)^3

ans =

  19683

>> 3^(3^3)

ans =

7625597484987

>>
```

⋆ `+` Vsota matrik je vsota po komponentah. Dimenzije sumandov se morajo ujemati.
`C=A+B`, $c_{ij} = a_{ij} + b_{ij}$. Primer vsote matrik

Matlab

```
>> A=repmat(3,3,4)

A =

     3     3     3     3
     3     3     3     3
     3     3     3     3

>> B=4*eye(3,4)

B =

     4     0     0     0
     0     4     0     0
     0     0     4     0
```

```
>> A+B

ans =

     7     3     3     3
     3     7     3     3
     3     3     7     3

>>
```

⋆ `-` Razlika matrik je razlika po komponentah. Dimenzije obeh se morajo ujemati.
`C=A-B`, $c_{ij} = a_{ij} - b_{ij}$. Poglejmo kakšna je razlika gornjih matrik

Matlab

```
>> A-B

ans =

    -1     3     3     3
     3    -1     3     3
     3     3    -1     3

>>
```

⋆ `*` Produkt matrik je matrika skalarnih produktov vrstic prvega s stolpci drugega faktorja. Število stolpcev prvega faktorja mora biti enako številu vrstic drugega.
`C=A*B`, $c_{ik} = \sum_j a_{i,j} b_{j,k}$

Matlab

```
>> A=repmat([1,2],3,2)

A =

     1     2     1     2
     1     2     1     2
     1     2     1     2

>> B=repmat([2;3],2,3)

B =

     2     2     2
     3     3     3
     2     2     2
     3     3     3

>> A*B

ans =
```

```
    16    16    16
    16    16    16
    16    16    16

>>
```

⋆ `.*` Produkt polj je produkt po komponentah. Faktorja morata biti enakega reda.
`C=A.*B`, $c_{ij} = a_{ij}b_{ij}$. Poglejmo primer produkta polj

Matlab

```
>> A=repmat([1,2],3,2)

A =

     1     2     1     2
     1     2     1     2
     1     2     1     2

>> B=repmat([3,4],3,2)

B =

     3     4     3     4
     3     4     3     4
     3     4     3     4

>> A.*B

ans =

     3     8     3     8
     3     8     3     8
     3     8     3     8

>>
```

⋆ `/` Desno deljenje matrik je množenje z inverzno matriko z desne.
`C=A/B`, $C = A \cdot B^{-1}$ Poglejmo primer desnega deljenja matrik:

Matlab

```
>> A=repmat([1,2],3,2)

A =

     1     2     1     2
     1     2     1     2
     1     2     1     2

>> B=rand(4)

B =
```

```
    0.4399    0.8392    0.6072    0.4514
    0.9334    0.6288    0.6299    0.0439
    0.6833    0.1338    0.3705    0.0272
    0.2126    0.2071    0.5751    0.3127

>> X=A/B

X =

    6.4154   -5.8303    6.0987   -2.5775
    6.4154   -5.8303    6.0987   -2.5775
    6.4154   -5.8303    6.0987   -2.5775

>> X*B

ans =

    1.0000    2.0000    1.0000    2.0000
    1.0000    2.0000    1.0000    2.0000
    1.0000    2.0000    1.0000    2.0000

>>
```

⋆ `./` Desno deljenje polj je deljenje po komponentah. Red matrik se mora ujemati.
`C=A./B`, $c_{ij} = a_{ij}/b_{ij}$.

⋆ `\` Levo deljenje matrik je množenje z leve z inverzno matriko.
`C=A\B`, $C = A^{-1} \cdot B$

Primer 5.2. *Rešimo sistem linearnih enačb $Ax = b$. Vzemimo, da matrika A ni singularna. Rešitev sistema je v tem primeru enaka $x = A^{-1}b$. Rešitev lahko izrazimo z levim deljenjem matrik* `x=A\b`.

Matlab

```
>> A=rand(5)

A =

    0.8939    0.4692    0.5155    0.7604    0.7833
    0.1991    0.0648    0.3340    0.5298    0.6808
    0.2987    0.9883    0.4329    0.6405    0.4611
    0.6614    0.5828    0.2259    0.2091    0.5678
    0.2844    0.4235    0.5798    0.3798    0.7942

>> b=rand(5,1)

b =

    0.0592
    0.6029
    0.0503
    0.4154
    0.3050
```

```
>> x=A\b

x =

   -0.6427
    0.2234
   -2.2862
    0.0317
    2.1489

>>
```

Še preizkus:

Matlab

```
>> A*x-b

ans =

   1.0e-15 *

    0.0139
   -0.1110
    0.0069
         0
    0.1110

>>
```

⋆ `.\` Levo deljenje polj
`C=A.\B`, $c_{ij} = b_{ij}/a_{ij}$.

⋆ `^` Potenciranje matrike. Če je potenčni eksponent naravno število, potem je to matrični produkt ustreznega števila danih matrik. Matrika mora biti kvadratna.
`A^n`, $A^n = A \cdot A \cdots A$, n faktorjev.
Če je eksponent negativno celo število, potem je enak matričnemu produktu ustreznega števila inverznih matrik k dani matriki. Matrika mora biti kvadratna in nesingularna.
`A^(-n)`, $A^{-n} = A^{-1} \cdot A^{-1} \cdots A^{-1}$, n faktorjev.

⋆ `.^` Potenciranje polja je potenciranje po komponentah.

⋆ `'` Operacija `'` je adjungiranje matrike. Matriki zamenjamo vrstice s stolpci in konjugiramo njene vrednosti. Če ima matrika realne vrednosti, potem ta operacija samo zamenja vrstice s stolpci.

⋆ `.'` Operacija transponiranja matrike oziroma polja, zamenja vrstice s stolpci ali z drugimi besedami prezrcali polje preko glavne diagonale.

Matlab

```
>> A
```

```
A =

     1     2     1     2
     1     2     1     2
     1     2     1     2

>> A'

ans =

     1     1     1
     2     2     2
     1     1     1
     2     2     2

>> clear i
>> z=1+3*i

z =

   1.0000 + 3.0000i

>> z'

ans =

   1.0000 - 3.0000i

>> A=[1,1+2*i;i,-2*i]

A =

   1.0000             1.0000 + 2.0000i
        0 + 1.0000i        0 - 2.0000i

>> A'

ans =

   1.0000                  0 - 1.0000i
   1.0000 - 2.0000i        0 + 2.0000i

>> A.'

ans =

   1.0000                  0 + 1.0000i
   1.0000 + 2.0000i        0 - 2.0000i

>>
```

Preden smo uporabili `i` kot imaginarno enoto, smo zvedli `clear i`, da smo inicializirali konstanto `i`, če je bi bila povožena s kakšno drugo vrednostjo.

Pravilo 5.1. *Če je pri operacijah* `+`, `-`, `*`, *in* `/` *eden od operandov skalarna matrika, je operacija z manjšimi omejitvami dovoljena. V primeru vsote in razlike se ustrezni skalar prišteje oziroma odšteje vsakemu členu matrike. V primeru produkta se obravnava kot produkt vektorja oziroma matrike s skalarjem, v primeru deljenj pa produkt z recipročno vrednostjo skalarja. Deljenje skalarja z matriko reda več kot* 1×1 *ni dovoljeno.*

Primer 5.3. *Naj morebitne nejasnosti razjasnijo naslednji primeri:*

Matlab

```
>> A=repmat(1:3,3,1)

A =

     1     2     3
     1     2     3
     1     2     3

>> A+3

ans =

     4     5     6
     4     5     6
     4     5     6

>> 3-A

ans =

     2     1     0
     2     1     0
     2     1     0

>> A-1

ans =

     0     1     2
     0     1     2
     0     1     2

>> A*4

ans =

     4     8    12
     4     8    12
     4     8    12

>> 3*A

ans =

     3     6     9
     3     6     9
     3     6     9
```

```
>> A/3

ans =

    0.3333    0.6667    1.0000
    0.3333    0.6667    1.0000
    0.3333    0.6667    1.0000

>> 4\A

ans =

    0.2500    0.5000    0.7500
    0.2500    0.5000    0.7500
    0.2500    0.5000    0.7500

>> 4/A
??? Error using ==> /
Matrix dimensions must agree.

>>
```

V zadnjem primeru nas je Matlab opozoril, da takšno deljenje ni dovoljeno.

6 Funkcije

6.1 Elementarne funkcije

Poglejmo nekatere elementarne funkcije implementirane v Matlabu. Argumenti elementarnih funkcij so lahko matrike, v tem primeru funkcija vrne matriko vrednosti funkcije na elementih argumenta. Več o elementarnih funkcijah v Matlabu bomo izvedeli, če v ukazno vrstico zapišemo
`>> help elfun`.

Trigonometrične funkcije	
`sin`	sinus
`asin`	arkus sinus
`cos`	kosinus
`acos`	arkus kosinus
`tan`	tangens
`atan`	arkus tanges
`atan2`	argument

⋆ `atan2` Morda bi se na tem mestu posebej ustavili pri funkciji `atan2`. Funkcijo kličemo z dvema argumentoma `fi=atan2(y,x)`. Rezultat je kot *(merjen v radianih)* med pozitivno smerjo osi x in vektorjem položaja točke (x, y) v ravnini, nahaja se na intervalu $(-\pi, \pi]$.

Eksponentna funkcija in logaritem	
`exp`	eksponentna funkcija e^x
`log`	naravni logaritem
`log10`	desetiški logaritem
`log2`	logaritem z osnovo 2
`pow2`	eksponentna funkcija 2^x
`sqrt`	kvadratni koren

⋆ `sqrt` Z `sqrt(x)` Matlab kliče posebno proceduro za računanje kvadratnega korena, medtem ko se s klicem `x.^(1/2)` kvadratni koren izračuna s pomočjo logaritemske in eksponentne funkcije `exp(1/2*log(x))` in je časovno manj ugoden.

6.2 Funkcije nad kompleksnimi števili

Funkcije nad kompleksnimi števili	
`abs`	absolutna vrednost
`angle`	argument
`conj`	konjugirana kompleksna vrednost
`real`	realni del
`imag`	imaginarni del

⋆ `abs` Funcija `abs` vrne absolutno vrednost kompleksnega števila. Če je $z = x + i\,y$, je $|z| = \sqrt{x^2 + y^2}$. Za realna števila pa je

$$|x| = \begin{cases} -x, & x < 0 \\ x, & x \geq 0 \end{cases}$$

⋆ `angle` Funkcija `angle` vrne kot med pozitivnim poltrakom realne osi in radijvektorjem argumenta. Kot je izražen v radianih v razponu od $-\pi < \alpha \leq \pi$. Kot je v prvem in drugem kvadrantu pozitiven v tretjem in četrtem pa negativen. Izraz $angle(z) == atan2(y, x)$ zavzame vrednost `true` za vsa kompleksna števila $z = x + i\,y$.

⋆ `conj` Funkcija `conj` vrne konjugirano (kompleksno) vrednost argumenta. Če je $z = x + i\,y$, potem je `conj(z)` enak $x - i\,y$.

⋆ `real, imag` Funkciji `real` in `imag` vrneta realni oziroma imaginarni del argumenta.

6.3 Rezanje, zaokrožavanje in ostanki

Rezanje, zaokroževanje in ostanki	
`fix`	reznaje proti nič
`floor`	navzdol, proti $-\infty$
`ceil`	navzgor, proti $+\infty$
`round`	zaokroževanje
`mod`	modul (računanje po modulu)
`rem`	ostanek pri deljenju
`sign`	predznak

⋆ `fix` Funkcija `fix` odreže decimalna mesta za decimalno piko.

⋆ `floor` Funkcija `floor`, v angleškem jeziku *floor* pomeni *tla*, vrne največje celo število, ki ne presega argumenta.

⋆ `ceil` Funkcija `ceil`, v anglešlkem jeziku *ceil* pomeni *strop*, vrne najmanjše celo število, ki je večje ali enako argumentu.

⋆ `round`

⋆ `mod` Funkcija `mod` je predzančen ostanek pri deljenju. Velja `mod(x,y)== x - y.*floor(x./y)`. Če je `y = 0`, potem je po definiciji velja `mod(x,0)==x`. Argumenta sta realni števili ali par realnih polj enake dimenzije.

⋆ `rem` Funkcija `rem`, podobno kot funkcija `mod`, poišče ostanek pri deljenju.

Velja `rem(x,y)==x - y.*fix(x./y)`, če je `y = 0`. Vrednosti `mod(x,0)` Matlab priredi `NaN`. Razlika je v tem, da je `rem(x,y)` istega znaka kot `x`, medtem ko je `mod(x,y)` istega znaka kot argument `y`. Obe funkciji (`mod` in `rem`) sta enaki, če sta argumenta istega znaka, in različni, če sta argumenta različnih znakov.

⋆ `sign` Funcija vrne predznak argumenta.

$$\operatorname{sign}(x) = \begin{cases} -1, & x < 0 \\ 0, & x = 0 \\ 1, & x > 0 \end{cases}$$

Primer 6.1. *Poglejmo si nekaj primerov*

Matlab

```
>> fix([1/3,-1/4,1.3,-3.2,5.1,5.6,-7.7])

ans =

     0     0     1    -3     5     5    -7

>> floor([1/3,-1/4,1.3,-3.2,5.1,5.6,-7.7])

ans =

     0    -1     1    -4     5     5    -8

>> ceil([1/3,-1/4,1.3,-3.2,5.1,5.6,-7.7])

ans =

     1     0     2    -3     6     6    -7

>> round([1/3,-1/4,1.3,-3.2,5.1,5.6,-7.7])

ans =

     0     0     1    -3     5     6    -8

>>
```

Poglejmo razliko med operatorjema `mod` *in* `rem`.

Matlab

```
>> mod(10,3)

ans =

     1

>> rem(10,3)

ans =

     1

> mod(-10,3)

ans =

     2

>> rem(-10,3)

ans =

    -1

>>
```

6.4 Vektorske funkcije

Vektor je matrika z enim samim stolpcem ali z eno samo vrstico. Poglejmo si nekaj značilnih funkcij, ki so definirane na vektorjih.

Osnovne vektorske funkcije

`length`	število komponent
`min`	minimalna komponenta
`max`	maksimalna komponenta
`mean`	srednja vrednost
`std`	standardna deviacija
`var`	disperzija
`sort`	urejanje
`sum`	vsota komponent
`prod`	produkt komponent
`cumsum`	kumulativna vsota
`cumprod`	kumulativni produkt
`diff`	razlika
`primes`	praštevila
`dot`	skalarni produkt
`cross`	vektorski produkt

Zgornje funkcije sprejemajo kot argumente tudi splošnejše matrike, ne samo vektorje. V tem primeru se ustrezna funkcija izvrši na stolpcih matrike, če posebej ne zahtevamo drugače.

⋆ `length` Funkcija `length` vrne število komponent, dolžino vektorja, ne glede na to, ali je to vektor stolpec, ali vektor vrstica. Pri matrikah vrne vrednost `max(size(·))`.

Matlab

```
>> x=[1 2 3 4 5 6]

x =

     1     2     3     4     5     6

>> length(x)

ans =

     6

>> length(x')

ans =

     6


>> A=rand(4,3)

A =

    0.9501    0.8913    0.8214
    0.2311    0.7621    0.4447
    0.6068    0.4565    0.6154
    0.4860    0.0185    0.7919

>> length(A)

ans =

     4

>> A=rand(3,4)

A =

    0.9218    0.4057    0.4103    0.3529
    0.7382    0.9355    0.8936    0.8132
    0.1763    0.9169    0.0579    0.0099

>> length(A)

ans =

     4

>>
```

⋆ `min, max` Funkciji `min` in `max` vračata vrednost najmanjše oziroma največje komponente vektorja. Funkciji vračata lahko tudi dve vrednosti. Druga vrnjena vrednost je indeks ustreznega elementa. Poglejmo si to na primeru funkcije `max`.

Matlab

```
>> x=rand(1,5)

x =

    0.1389    0.2028    0.1987    0.6038    0.2722

>> m=max(x)

m =

    0.6038

>> [m,i]=max(x)

m =

    0.6038


i =

     4

>> x(i)

ans =

    0.6038

>>
```

Naj bo argument funkcije splošnejša matrika.

Matlab

```
>> A=rand(2,3)

A =

    0.4186    0.5252    0.6721
    0.8462    0.2026    0.8381

>> [m,i]=max(A)

m =

    0.8462    0.5252    0.8381


i =
```

```
     2     1     2

>>
```

Rezultat je vrstica maksimalnih elementov v stolpcih in vrstica njihovih indeksov. Če želimo izračunati maksimalne vrednosti v vrsticah, potem je klic funkcije naslednji

Matlab

```
>> A

A =

    0.4186    0.5252    0.6721
    0.8462    0.2026    0.8381

>> [m,i]=max(A,[],2)

m =

    0.6721
    0.8462


i =

     3
     1

>>
```

Drugi argument mora biti prazna matrika, tretji pa je dimenzija, vzdolž katere izvedemo funkcijo. Drugi argument je prazna matrika zato, da se ta operacija loči od matrične operacije `max(A,B)`, kjer iščemo večjega od istoležnih elementov dveh matrik, ki imata enako število stolpcev in vrstic. Poglejmo razliko

Matlab

```
>> max(1,2)

ans =

     2

>> max(1,[],2)

ans =

     1
```

⋆ `mean` Je povprečna vrednost elementov vektoja:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

oziroma, če zapišemo v sintaksi Matlaba:
`mean(x)==sum(x)/length(x)`.

⋆ `var` Je disperzija elementov vektorja:

$$\mathrm{var}(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2$$

oziroma v sintaksi Matlaba:
`var(x)==sum((x-mean(x)).^2)/(length(x)-1)`.

⋆ `std` Je standardna deviacija elementov vektorja, `std==sqrt(var(x))`.

⋆ `sort` Funkcija `sort` razvršča elemente vektorja v naraščajočem vrstnem redu. Funkcija `sort` vrne dva vektorja prvi je vektor z urejenimi elementi, drugi pa je vektor s permutacijo indeksov, ki je potrebna, da se uredi vhodni vektor. Če je argument matrika, potem ukaz `sort` uredi stolpce matrike in poleg tako urejene matrike vrne tudi matriko stolpcev permutacij indeksov.

Primer 6.2. *Natančno prouči naslednji primer.*

Matlab

```
>> a=rand(5,4)

a =

    0.4057    0.0579    0.2028    0.0153
    0.9355    0.3529    0.1987    0.7468
    0.9169    0.8132    0.6038    0.4451
    0.4103    0.0099    0.2722    0.9318
    0.8936    0.1389    0.1988    0.4660

>> [s,i]=sort(a)

s =

    0.4057    0.0099    0.1987    0.0153
    0.4103    0.0579    0.1988    0.4451
    0.8936    0.1389    0.2028    0.4660
    0.9169    0.3529    0.2722    0.7468
    0.9355    0.8132    0.6038    0.9318


i =

     1     4     2     1
     4     1     5     3
     5     5     1     5
     3     2     4     2
     2     3     3     4

>>
```

Če želimo v matriki `A` *urediti vrstice, potem to storimo z ukazom* `sort(A,2)`. *Drugi parameter je dimenzija vzdolž katere želimo urejati. To pomeni, da* `sort(A,1)` *deluje enako kot* `sort(A)`. *Kaj več boste izvedeli z ukazom* `help`

`sort`. *Sorodna funkcija funkciji* `sort` *je* `sortrows`. *Bralec naj sam ugotovi, kako deluje, za začetek si naj ogleda, kaj pove ukaz* `help sortrows`.

⋆ `sum, prod` Funkcija `sum` vrne vsoto elementov vektorja, medtem ko `prod` vrne produkt komponent. Če je argument splošnejša matrika, potem je rezultat vrstica, katere komponente so vsote oziroma produkti elementov ustreznih stolpcev. Funkciji sprejemata tudi dva argumenta, drugi je dimenzija vzdolž katere računamo. Če je ta 1, se izvrši operacija na stolpcih, če pa je 2, se izvrši na vrsticah matrike. Poglejmo primere uporabe:

Matlab
```
>> x=2:3:20

x =

     2     5     8    11    14    17    20

>> sum(x)


ans =

    77

>> prod(x)

ans =

     4188800

>>
```

S pomočjo funkcije `prod` lahko računamo faktoriele.

Matlab
```
>> prod(1:5)

ans =

   120

>>
```

Če je argument splošnejša matrika,

Matlab
```
>> A=repmat(1:4,3,1)

A =

     1     2     3     4
     1     2     3     4
     1     2     3     4
```

```
>> sum(A)

ans =

     3     6     9    12

>> sum(A,1)

ans =

     3     6     9    12

>> sum(A,2)

ans =

    10
    10
    10

>>
```

Podobno velja tudi za funkcijo `prod`.

⋆ `cumsum, cumprod` Funkciji `cumsum` in `cumprod` izračunata kumulativno vsoto in kumulativni produkt. Kot pri drugih funkcijah, lahko tudi tu povemo, ali želimo operacijo izvršiti po stolpcih, ali po vrsticah.

Primer 6.3. *Uporabimo funkcijo* `cumprod` *za izračun števila e.*

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

Poglejmo kako natančen bo rezultat, če seštejemo 11 oziroma 21 členov vrste.

Matlab

```
>> a = vpa(1+ sum(1./cumprod(1:10)))

a =

2.7182818011463849572351136885118

>> a = vpa(1+ sum(1./cumprod(1:20)))

a =

2.7182818284590455348848081484903

>> vpa(exp(1))

ans =

2.7182818284590450907955982984276
```

```
>>
```

Uporabili smo ukaz `vpa`*, ki računa na več decimalnih mest. Več o tem dobimo z* `help vpa`*.*

6.5 Matrične funkcije

Preoblikovanje matrik

Preoblikovanje matrik	
`triu`	zgornje trikotni del matrike
`tril`	spodnje trikotni del matrike
`diag`	diagonala matrike
`reshape`	sprememba oblike
`fliplr`	zrcaljenje matrike levo desno
`flipud`	zrcaljenje matrike gor dol
`transpose`	transponiranje matrike
`ctranspose`	adjungiranje ali konjugirano transponiranje

⋆ `triu` Funkcija `triu` izloči zgornjetrikotno matriko. Lahko sprejema en argument, to je matriko, iz katere se izloči zgornjetrikotni del, oziroma dva argumenta, prvi je matrika, drugi pa je celo število, ki pove glede na katero vzporednico glavne diagonale bomo izbirali elemente. Indeks glavne diagonale je 0. Indeksi vzporednih diagonal v zgornje trikotni matriki so pozitivni, v spodnje trikotni matriki pa negativni.

Matlab

```
>> A=rand(3,4)

A =

    0.6822    0.1509    0.8600    0.4966
    0.3028    0.6979    0.8537    0.8998
    0.5417    0.3784    0.5936    0.8216

>> triu(A)

ans =

    0.6822    0.1509    0.8600    0.4966
         0    0.6979    0.8537    0.8998
         0         0    0.5936    0.8216

>> triu(A,0)

ans =

    0.6213    0.5226    0.9797    0.8757
         0    0.8801    0.2714    0.7373
         0         0    0.2523    0.1365

>> triu(A,2)
```

```
ans =

         0         0    0.9797    0.8757
         0         0         0    0.7373
         0         0         0         0

>> triu(A,-1)

ans =

    0.6822    0.1509    0.8600    0.4966
    0.3028    0.6979    0.8537    0.8998
         0    0.3784    0.5936    0.8216

>>
```

⋆ `tril` Podobno kot `triu` deluje funkcija `tril`, le da ta izbira člene matrike pod ustrezno vzpordnico diagonali, torej:

Matlab

```
>>  A=rand(3,4)

A =

    0.0118    0.2987    0.4692    0.5828
    0.8939    0.6614    0.0648    0.4235
    0.1991    0.2844    0.9883    0.5155

>> tril(A)

ans =

    0.0118         0         0         0
    0.8939    0.6614         0         0
    0.1991    0.2844    0.9883         0

>> tril(A,1)

ans =

    0.0118    0.2987         0         0
    0.8939    0.6614    0.0648         0
    0.1991    0.2844    0.9883    0.5155

>> tril(A,-1)

ans =

         0         0         0         0
    0.8939         0         0         0
    0.1991    0.2844         0         0
```

```
>>
```

⋆ `diag` Funkcija `diag` združuje dve različni funkciji, glede na to kakšen je njen argument.

1. Če je argument vektor, to je matrika z enim samim stolpcem, ali matrika z eno samo vrstico, je rezultat diagonalna matrika, ki hrani v diagonali elemente tega vektorja.

Matlab

```
>> x=[1,2,3]

x =

     1     2     3

>> diag(x)

ans =

     1     0     0
     0     2     0
     0     0     3

>> x=[1;2;3]

x =

     1
     2
     3

>> diag(x)

ans =

     1     0     0
     0     2     0
     0     0     3

>>
```

2. Če je argument funkcije `diag` matrika, potem je rezultat vektor stolpec s komponentami iz glavne diagonale te matrike.

Matlab

```
>> A=rand(3)

A =

    0.1389    0.6038    0.0153
    0.2028    0.2722    0.7468
    0.1987    0.1988    0.4451

>> diag(A)
```

```
ans =

    0.1389
    0.2722
    0.4451

>> A=rand(4,3)

A =

    0.6721    0.3795    0.4289
    0.8381    0.8318    0.3046
    0.0196    0.5028    0.1897
    0.6813    0.7095    0.1934

>> diag(A)

ans =

    0.6721
    0.8318
    0.1897

>>
```

⋆ `reshape` Funkcija `reshape` spremeni obliko matrike. Pri tem se ne spremeni število, niti vrstni red elementov matrike v pomnilniku. Poglejmo naslednji primer

Matlab

```
>> A=[1 3 5; 2 4 6]

A =

     1     3     5
     2     4     6

>> A(:)'

ans =

     1     2     3     4     5     6

>> B=reshape(A,3,2)

B =

     1     4
     2     5
     3     6

>> B(:)'

ans =
```

```
     1     2     3     4     5     6

>>
```

⋆ `fliplr` Funkcija `fliplr` izbere stolpce matrike v obratnem vrstnem redu, medtom ko `flipud` izbere vrstice v obratnem redu.

⋆ `transpose` Funkcija `transpose` zamenja vrstice s stolpci. Poglejmo si nekaj primerov. Kot smo že omenili transponiranje matrike kratko označimo z `A.'`.

Matlab

```
>> A

A =

     1     3     5
     2     4     6

>> fliplr(A)

ans =

     5     3     1
     6     4     2

>> flipud(A)

ans =

     2     4     6
     1     3     5

>> transpose(A)

ans =

     1     2
     3     4
     5     6

>> A.'

ans =

     1     2
     3     4
     5     6

>>
```

⋆ `ctranspose` Funkcija zamenja vrstice s stolpci in hkrati konjugira elemente matrike. To funkcijo oziroma enočleno involutivno operacijo nad matrikami oznmačimo z `'` .

Matlab

```
>> clear  i
>> format rat
>> A=fix(3*rand(3))+i*fix(3*rand(3))

A =

     1                    1       +    2i           2       +    2i
     1       +    2i      0       +    2i           2
     2       +    2i      1                         1       +    2i

>> A'

ans =

     1                    1       -    2i           2       -    2i
     1       -    2i      0       -    2i           1
     2       -    2i      2                         1       -    2i

>>
```

Še dve matrični operaciji

Dva matrična operanda	
`min`	manjši od obeh
`max`	večji od obeh

⋆ `max` Funkcija `max` sprejema dva argumenta, polji enakih dimenzij, in vrne polje iste dimenzije z elementi, karerih vrednost je večja od istoležnih elementov polj v argumentih.

⋆ `min` Funkcija `min` sprejema dva argumenta, kot pri funkciji `max`, le da je rezultirajoče polje polje minimalnih vrednosti.

7 Linearna algebra

Linearna algebra	
`null`	ničelni prostor
`orth`	ortogonalna baza prostora stolpcev matrike
`rank`	rang matrike
`det`	determinanta
`rref`	Gauss-Jordanova eliminacija
`eig`	lastne vrednosti in lastni vektorji matrike
`cond`	pogojenost
`norm`	norma matrike
`inv`	inverzna matrika
`pinv`	psevdoinverzna matrika
`svd`	razcep matrike po singularnih vrednostih

⋆ `null` Funkcija `null` vrne ortogonalno bazo ničelnega podprostora matrike, ali z drugimi besedami, lastnega podprostora, ki pripada lastni vrednosti 0. Ali še drugače: ničelni prostor matrike A je množica vektorjev x, za katere velja $A\,x = 0$.

Matlab

```
>> A=[1 2 3 4; 5 6 7 8; 9 10 11 12]

A =

     1     2     3     4
     5     6     7     8
     9    10    11    12

>> x=null(A)

x =

   0.44585517174707   0.31814016694908
  -0.83160124221985  -0.09186606521668
   0.32563696919850  -0.77068837041389
   0.06010910127428   0.54441426868149

>> A*x

ans =

   1.0e-14 *

  -0.02498001805407   0.00832667268469
  -0.01110223024625   0.09159339953158
   0.00277555756156   0.17486012637846

>> y=x*[33;44]

y =

  28.71138801341301
 -31.48494786278919
 -23.16426831466061
  25.93782816403680

>> A*y

ans =

   1.0e-14 *

  -0.02498001805407   0.00832667268469
  -0.01110223024625   0.09159339953158
   0.00277555756156   0.17486012637846

>> y=x*[33;44]

y =
```

```
  28.71138801341301
 -31.48494786278919
 -23.16426831466061
  25.93782816403680

>> A*y

ans =

   1.0e-13 *

  -0.03552713678801
   0.39079850466806
   0.81712414612412

>>
```

Stolpca matrike `x` tvorita bazo ničelnega podprostora matrike `A`. Ničelni prostor naše matrike je dvodimenzionalen. Seveda je poljubna linearna kombinacija stolpcev matrike `x` tudi v ničelnemu prostoru matrike `A`. To smo preverili na koncu. Vrednosti, zaradi zaokrožitvenih napak, niso natanko enake nič.

⋆ `orth` Funkcija `orth` vrne ortogonalno bazo prostora, katerega generatorji so stolpci matrike.

Primer 7.1. *Poglejmo bazo prostora stolpcev matrike* `A`.

Matlab

```
A =

     1     2     3     4
     5     6     7     8
     9    10    11    12

>> z=orth(A)

z =

  -0.20673589125763   0.88915330770303
  -0.51828873789912   0.25438183406107
  -0.82984158454060  -0.38038963958089

>> rank(A)

ans =

     2

>>
```

Naj si bralec sam razloži naslednji račun.

Matlab

```
>> x=null(A)

x =
```

```
   0.44585517174707   0.31814016694908
  -0.83160124221985  -0.09186606521668
   0.32563696919850  -0.77068837041389
   0.06010910127428   0.54441426868149

>> z=orth(A')

z =

  -0.40361757215215   0.73286619205009
  -0.46474413034915   0.28984977713639
  -0.52587068854614  -0.15316663777731
  -0.58699724674314  -0.59618305269101

>> z'*x

ans =

   1.0e-15 *

   0.09237741322755  -0.09638557313396
  -0.26401339339559   0.16200690962365

>>
```

⋆ `rank` Funkcija `rank` pove, kolikšen je rang matrike ali z drugimi besedami, kolikšno je maksimalno število linearno neodvisnih vrstic oziroma stolpcev. V primeru 7.1 smo videli, da ima baza prostora stolpcev matrike `A` dva vektorja, torej je imenzija prostora stolpcev 2, oziroma rang matrike je 2.

⋆ `det` Funkcija `det` izračuna determinanto kvadratne matrike.

Matlab

```
>> B=1:16

B =

  Columns 1 through 12

     1     2     3     4     5     6     7     8     9    10    11    12

  Columns 13 through 16

    13    14    15    16

>> C=reshape(B,4,4)

C =

     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16
```

```
>> det(C)

ans =

     0

>> rank(C)

ans =

     2

>> rank(1./C)

ans =

     4

>>
```

Determinanta kvadratne matrike je različna od nič natanko tedaj, ko je rang enak redu matrike.

⋆ `rref` Funkcija `rref` naredi Gauss-Jordanovo eliminacijo na vrsticah matrike, ali z drugimi besedami, reducira matriko s pomočjo elementarnh transformacija na vrsticah. Če je matrika kvadratna in je njena determinanta različna od nič, potem je rezultirajoča matrika enotna matrika enakega reda. Število od nič različnih elementov (število enic) na glavni diagonali rezultirajoče matrike je enako rangu prvotne matrike.

Pravilo 7.1. *Elementarne transformacije na vrsticah so:*

1. *množenje vrstice s številom različnim od nič,*
2. *zamenjava vrstic in*
3. *prištevanje eni vrstici linearno kombinacijo ostalih.*

Poglejmo primer

Matlab

```
A =

     1     2     3     4
     5     6     7     8
     9    10    11    12

>> rref(A)

ans =

     1     0    -1    -2
     0     1     2     3
     0     0     0     0

>> C
```

```
C =

     1     5     9    13
     2     6    10    14
     3     7    11    15
     4     8    12    16

>> rref(C)

ans =

     1     0    -1    -2
     0     1     2     3
     0     0     0     0
     0     0     0     0

>> D=floor(5*rand(4,4))

D =

     4     4     4     4
     1     3     2     3
     3     2     3     0
     2     0     3     2

>> rref(D)

ans =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1

>>
```

Z `rref` lahko poiščemo inverzno matriko in pa rešujemo sisteme enačb.

Matlab

```
>> E=[D,eye(size(D))]

E =

     4     4     4     4     1     0     0     0
     1     3     2     3     0     1     0     0
     3     2     3     0     0     0     1     0
     2     0     3     2     0     0     0     1

>> F=rref(E)

F =

  Columns 1 through 7
```

```
    1.0000         0         0         0    0.7000   -0.8000   -0.2000
         0    1.0000         0         0   -0.0750    0.3000    0.2000
         0         0    1.0000         0   -0.6500    0.6000    0.4000
         0         0         0    1.0000    0.2750   -0.1000   -0.4000

  Column 8

   -0.2000
   -0.3000
    0.4000
    0.1000

>> D*F(:,5:8)

ans =

    1.0000   -0.0000         0    0.0000
   -0.0000    1.0000         0    0.0000
   -0.0000   -0.0000    1.0000    0.0000
   -0.0000   -0.0000         0    1.0000

>>
```

Matrika `F(:,5:8)` je inverzna matriki `D`. Če matriko `D` razširimo s stolpcem, `b` potem je zadnji stolpec matrike `rref([D,b])` rešitev enačbe `A*x=b`.

Pravilo 7.2. *Ukaz* `rref` *pretvori razširjeno matriko* $[A, B]$*, s pomočjo elementarnih transformacij, v matriko* $[I, A^{-1}B]$*, če je matrika A kvadratna matrika z determinanto različno od nič. Matrika I je enotna matrika enakega reda kot matrika A.*

Rešimo sistem `D*x=b` in na koncu preverimo rešitev.

Matlab

```
>> D

D =

     4     4     4     4
     1     3     2     3
     3     2     3     0
     2     0     3     2

>> b=[2;3;4;7];
>> x=rref([D,b])

x =

    1.0000         0         0         0   -3.2000
         0    1.0000         0         0   -0.5500
         0         0    1.0000         0    4.9000
         0         0         0    1.0000   -0.6500

>> D*x(:,end)-b
```

```
ans =

   1.0e-15 *

    0.4441
    0.4441
         0
    0.8882

>>
```

⋆ `eig` Funkcija `eig` vrne lastne vrednosti in lastne vektorje matrike. Lastne vrednosti matrike A so vrednosti λ, za katere najdemo vektor $x \neq 0$, da je $Ax = \lambda x$. Lastne vrednosti so ničle karakterističnega polinoma $|A - \lambda I| = 0$ oziroma `det(A-lambda*I)==0`. Algebrska kratnost lastne vrednosti je enaka stopnji ničle karakterističnega polinoma. Lastni podprostor, ki pripada lastni vrednosti, je enak ničelnemu prostoru matrike $A - \lambda I$. Poglejmo primer:

Matlab

```
>> A=diag(fix(4*rand(5,1)))+diag(fix(2*rand(4,1)),1)+diag(fix(2*rand(3,1)),2)

A =

     2     0     1     0     0
     0     2     0     1     0
     0     0     2     0     0
     0     0     0     0     1
     0     0     0     0     1

>> [a,b]=eig(A)

a =

    1.0000         0   -1.0000         0         0
         0    1.0000         0   -0.4472   -0.5774
         0         0    0.0000         0         0
         0         0         0    0.8944    0.5774
         0         0         0         0    0.5774


b =

     2     0     0     0     0
     0     2     0     0     0
     0     0     2     0     0
     0     0     0     0     0
     0     0     0     0     1

>>
```

Lastne vrednosti matrike a so:

Matlab

```
>> diag(b)'
```

```
ans =

     2     2     2     0     1

>>
```

Algebrska kratnost lastne vrednosti 2 je 3, medtem ko sta algebrski kratnosti lastnih vrednosti 0 in 1 enaki 1. Izberimo lastne vektorje, ki pripadajo posamezni lastni vrednosti.

Matlab

```
>> u=diag(b)'

u =

     2     2     2     0     1

>> lv=unique(u)

lv =

     0     1     2

>> a(:,find(u==lv(1)))'

ans =

         0   -0.4472         0    0.8944         0

>> a(:,find(u==lv(2)))'

ans =

         0   -0.5774         0    0.5774    0.5774

>> a(:,find(u==lv(3)))'

ans =

    1.0000         0         0         0         0
         0    1.0000         0         0         0
   -1.0000         0    0.0000         0         0

>> rank(a(:,find(u==lv(1)))')

ans =

     1

>> rank(a(:,find(u==lv(2)))')

ans =

     1
```

```
>> rank(a(:,find(u==lv(3)))')

ans =

     2

>>
```

V drugem delu smo pogledali, koliko je rang matrike lastnih vektorjev, ki pripadajo posamezni lastni vrednosti, torej dimenzijo lastnega podprostora, ki pripada dani lastni vrednosti. To dimenzijo imenujemo tudi geometrijska kratnost lastne vrednosti.

Pravilo 7.3. *Ena je manj ali enako geometrijski kratnosti lastne vrednosti, ta pa je manj ali enako algebrski kratnosti.*

⋆ `cond` Funkcija `cond` izračuna število občutljivosti matrike, to je $||A|| \cdot ||A^{-1}||$.

⋆ `norm` Funkcija `norm` poišče različne norme matrike.

1. Evklidska norma vektorja je
$$||x||_2 = \sum_i |x_i|^2$$
Evklidska norma matrike A je
$$||A||_2 = \max_{||x||_2=1} ||Ax||_2$$
2. Neskončna norma vektorja
$$||x||_\infty = \max_i |x_i|$$
Neskončna norma matrike
$$||A||_\infty = max_i \sum_j |a_{ij}|$$
3. Prva norma vektorja
$$||x||_1 = \sum_i |x_i|$$
Prva norma matrike
$$||A||_1 = max_j \sum_i |a_{ij}|$$

⋆ `inv` Funkcija `inv` vrne inverzno matriko, `A^(-1)==inv(A)`. V programu je varneje uporabljati funkcijo `inv`, ker bo v tem primeru gotovo uporabljen algoritem za iskanje inverzne matrike in ne splošnejši algoritem za potenciranje matrike, ki je lahko časovno bolj zahteven.

⋆ `pinv` Funkcija `pinv` vrne psevdoinverzno matriko. Psevdoinverzna matrika k matriki A je matrika P za katero velja:
$$APA = A \quad \text{in} \quad PAP = P$$
Če je determinanta matrike različna od nič, je psevdoinverzna matrika enaka inverzni matriki.

Pravilo 7.4. *Za psevdoinverzno matriko P k matriki A velja*

1. *matrika P je enakega reda kot matrika A'.*
2. *Matriki AP in PA sta hermitski, to pomeni, da je* `(AP)'==AP` *in* `(PA)'==PA`.

3. *Če je* `det` $A \neq 0$, *potem je* $P = A^{-1}$.
4. *Če je matrika* A *polnega ranga, to pomeni, da je* `rank(A)==min(size(A))`, *potem je* $P = (A'A)^{-1}A'$ *v primeru, ko je število vrstic večje od števila stolpcev in* $P = A'(AA')^{-1}$, *ko je število vrstic manjše od števila stolpcev.*

Matlab

```
>> A

A =

     1     2     3     4
     5     6     7     8
     9    10    11    12

>> P=pinv(A)

P =

   -0.3750   -0.1000    0.1750
   -0.1458   -0.0333    0.0792
    0.0833    0.0333   -0.0167
    0.3125    0.1000   -0.1125

>> A*P*A-A

ans =

   1.0e-14 *

   -0.1776   -0.1332   -0.0888   -0.0444
   -0.2665   -0.2665   -0.2665   -0.1776
   -0.3553   -0.3553   -0.3553   -0.3553

>> P*A*P-P

ans =

   1.0e-15 *

   -0.2220   -0.0694    0.0833
   -0.0833   -0.0208    0.0278
    0.0694    0.0208   -0.0382
    0.2220    0.0555   -0.0971
```

⋆ `svd` Z `svd` dobimo razcep matrike $A = U\Sigma V^{-1}$, kjer sta matriki U in V ortogonalni matriki, medtem ko diagonalna matrika Σ nosi singularne vrednosti matrike A.

Pravilo 7.5. *Singularne vrednosti so kvadratni koreni lastnih vrednosti matrike* $A'A$ *oziroma* AA'. *Druga norma matrike je enaka največji singularni vrednsti.*

Psevdoinverzno matriko dobimo lahko tudi s pomočjo funkcije `svd`.

Funkcijo pokličemo takole `[U,S,V]=svd(A)`. Rezultat sta dve ortogonalni matriki `U` in `V` in diagonalna matrika singularnih vrednosti. Število elementov različnih od nič na diagonali je enako rangu matrike `A`. Velja `A==U*S*V'`. Izberimo matriko `A` in poiščimo njen razcep po singularnih vrednostih.

Matlab

```
A =

     1     2     3     4
     5     6     7     8
     9    10    11    12

>> [U,S,V]=svd(A)

U =

   -0.2067    0.8892    0.4082
   -0.5183    0.2544   -0.8165
   -0.8298   -0.3804    0.4082


S =

   25.4368         0         0         0
         0    1.7226         0         0
         0         0    0.0000         0


V =

   -0.4036   -0.7329    0.4459    0.3181
   -0.4647   -0.2898   -0.8316   -0.0919
   -0.5259    0.1532    0.3256   -0.7707
   -0.5870    0.5962    0.0601    0.5444

>> U*S*V'

ans =

    1.0000    2.0000    3.0000    4.0000
    5.0000    6.0000    7.0000    8.0000
    9.0000   10.0000   11.0000   12.0000

>>
```

Sedaj pa poglejmo, kako so povezane matrike `U`, `S` in `V` in psevdoinverzna matrika matrike `A`. Psevdoinverzna matrika `pinv(A)` je enaka matriki `V'*T*U`, kjer je matriko `T` izračunamo takole:

Matlab

```
>> T=zeros(size(S));
>> T(S~=0)=1./S(S~=0);
```

Na diagonali matrike `T` so recipročne vrednosti od nič različnih elementov matrike `S`. Ker pa račun v aritmetiki s plavajočo piko ni točen, se odločimo za prag. Vse elemente, katerih absolutna vrednost je manjša od praga, bomo šteli, da so enaki nič. Vzemimo, da je prag `10*eps`.

Matlab

```
>> S(abs(S)<10*eps)=0

S =

   25.4368         0         0         0
         0    1.7226         0         0
         0         0         0         0

>> S(S~=0)=1./S(S~=0)

S =

    0.0393         0         0         0
         0    0.5805         0         0
         0         0         0         0

>> V*S'*U'

ans =

   -0.3750   -0.1000    0.1750
   -0.1458   -0.0333    0.0792
    0.0833    0.0333   -0.0167
    0.3125    0.1000   -0.1125

>> pinv(A,10*eps)

ans =

   -0.3750   -0.1000    0.1750
   -0.1458   -0.0333    0.0792
    0.0833    0.0333   -0.0167
    0.3125    0.1000   -0.1125

>>
```

Singularne vrednosti, različne od nič, matrike `A` so lastne vrednosti, različne od nič, simetričnih matrik `A'*A` in `A*A'`.

Matlab

```
>> sqrt(eig(A'*A))

ans =

         0
         0
    1.7226
   25.4368

>> sqrt(eig(A*A'))

ans =

    0.0000
```

```
    1.7226
   25.4368

>> S

S =

   25.4368         0         0         0
         0    1.7226         0         0
         0         0    0.0000         0

>>
```

Vzemimo predoločeni sistem enačb z matriko polnega ranga, $A\,x = b$. Predoločeni pomeni, da ima sistem več enačb kot neznank. Rešitev takšnega sistema v splošnem ne obstaja. Zato iščemo vektor x, ki minimizira normo razlike $||A\,x - b||_2$. Ker so stolpci matrike A linearno neodvisni (matrika je polnega ranga in ima več vrstic kot stolpcev), je matrika $A'A$ nesingularna.

Pravilo 7.6. *Rešitev enačbe $A'A\,x = A'b$ minimizira zgornji izraz. Ker je matrika $A'A$ nesigularna, je rešitev enolična in je enaka $x = (A'A)^{-1}A'b$.*

Rešitev lahko v Matlabu poiščemo tudi s psevdoinverzom matrike A ali z levim deljenjem.

Matlab

```
>> A=floor(5*rand(5,3))

A =

     4     1     0
     4     4     3
     2     0     1
     4     0     0
     0     1     0

>> b=floor(3*rand(5,1))

b =

     2
     1
     2
     1
     1

>> x=(A'*A)^(-1)*A'*b

x =

    0.4098
   -0.0865
    0.0301

>> A*x-b

ans =
```

```
   -0.4474
    0.3835
   -1.1504
    0.6391
   -1.0865

>> x=pinv(A)*b

x =

    0.4098
   -0.0865
    0.0301

>> x=A\b

x =

    0.4098
   -0.0865
    0.0301

>>
```

8 Relacijski in logični operatorji ter množice

8.1 Relacijski operatorji

Relacijski operatorji	
`==`	enako
`~=`	ni enako
`<`	manjše
`>`	večje
`<=`	manjše ali enako
`>=`	večje ali enako
`logical`	določi logični tip
`find`	indeksi elementov, ki ustrezajo pogoju so različni od nič

Relacijski operatorji določajo dvočlene operacije nad parom realnih polj enakega reda. Rezultat je polje boolovih ali logičnih vrednosti istega reda. Logične vrednosti so predstavljene z realnima vrednostima 0, `false` in 1, `true`.

⋆ `==` Enakost v Matlabu označimo z dvojnim enačajem `==`, enojni je rezerviran za prirejanje. Matlab primerja istoležne elemente v operandih in zapiše v rezultat logično vrednost 1, `true`, če sta ta enaka in 0, `false`, če nista.

Rezultirajoče polje je polje realnih ničel in enic. Vendar pa polje ni popolnoma enako običajnemu realnemu polju ničel in enic, ki ga dobmo z aritmetičnimi operacijami. Polje logičnih vrednosti je posebej označeno.

Primer 8.1.

Matlab

```
>> clear
>> A=[1 2 3 4; 3 4 2 1; 5 3 2 6]

A =

     1     2     3     4
     3     4     2     1
     5     3     2     6

>> B=[2 1 5 4; 3 4 5 1; 4 3 1 6]

B =

     2     1     5     4
     3     4     5     1
     4     3     1     6

>> C=A==B

ans =

     0     0     0     1
     1     1     0     1
     0     1     0     1
```

```
>> whos
  Name      Size         Bytes  Class

  A         3x4             96  double array
  B         3x4             96  double array
  C         3x4             96  double array (logical)

Grand total is 36 elements using 288 bytes

>>
```

Vidimo, kako si Matlab zapomni polje logičnih vrednosti. Če postane takšno polje operand aritmetične operacije, ga Matlab obravnava kot polje realnih vrednosti 0 in 1.

Matlab

```
>> C=3*C

C =

     0     0     0     3
     3     3     0     3
     0     3     0     3

>> whos
  Name      Size         Bytes  Class

  A         3x4             96  double array
  B         3x4             96  double array
  C         3x4             96  double array

Grand total is 36 elements using 288 bytes

>>
```

Polje `C` je postalo polje običajnih realnih vrednosti. Izgubilo je atribut `logical`.

V zvezi z relacijo enakosti naj opozorimo na naslednji primer.

Primer 8.2. *Če se vprašamo, ali je* `NaN` *enako* `NaN` *dobimo odgovor:*

Matlab

```
>> NaN==NaN

ans =

     0

>>
```

Rezultat relacijske operacije je enak 0 `false`, če je vsaj en operand enak `NaN`.

⋆ `~=` Operator `~=` določa relacijo neenakosti. Če sta istoležna elementa v operandih različna je rezultat 1, `true`, če pa sta enaka, potem je rezultat 0, `false`. Vzemimo matriki `A` in `B` iz primera (8.1).

Matlab
```
>> C=A~=B

C =

     1     1     1     0
     0     0     1     0
     1     0     1     0

>>
```

⋆ `<, >, <=, >=` Operatorji `<`, `>`, `<=` in `>=` določajo operacije manjši, večji, manjši ali enak in večji ali enak. Vzemimo ponovno matriki `A` in `B` iz primera (8.1).

Matlab
```
>> A<B

ans =

     1     0     1     0
     0     0     1     0
     0     0     0     0

>> A<=B

ans =

     1     0     1     1
     1     1     1     1
     0     1     0     1

>>
```

⋆ `logical` Funkcija `logical` pretvori realno matriko v matriko tipa `logical`, pri tem matrika zadrži svoje realne vrednosti.

Matlab
```
>> A=[0 1 2 3; 4 5 0 0]

A =

     0     1     2     3
     4     5     0     0

>> B=logical(A)

B =

     0     1     2     3
     4     5     0     0
```

```
>> whos
  Name      Size         Bytes  Class

  A         2x4             64  double array
  B         2x4             64  double array (logical)

Grand total is 16 elements using 128 bytes

>> C=B+1

C =

     1     2     3     4
     5     6     1     1

>> D=1&A

D =

     0     1     1     1
     1     1     0     0

>> E=D+1

E =

     1     2     2     2
     2     2     1     1

>>
```

Pri pretvorbi iz realne v logično matriko in nazaj, so se stare realne vrednosti ohranile.

⋆ `find` Funkcija `find` poišče indekse matrike, kjer so vrednosti različne od nič. Poglejmo primer.

Matlab

```
>> A=fix(rand(2,5)*3)

A =

     2     2     1     1     2
     0     1     2     1     1

>> B=fix(rand(2,5)*3)

B =

     2     1     0     0     2
     2     2     2     0     2

>> find(A)'

ans =
```

```
     1     3     4     5     6     7     8     9    10

>> find(A>B)'

ans =

     3     5     7     8

>>
```

8.2 Logični operatorji

Logični operatorji	
`&`	logični in
`|`	logični ali
`~`	logični ne
`xor`	logični izključujoči ali
`all`	vsi
`any`	vsaj eden

Logični operatorji določajo enočleno in dvočlene logične operacije nad parom logičnih polj istega reda. Rezultat je polje logičnih vrednosti.

⋆ `&` **Logični in** ali konjunkcijo označimo z znakom `&`, Matlab primerja istoležne elemente v operandih in zapiše v rezultirajoče polje logično vrednost 1, `true` , če sta oba operanda različna od nič in logično vrednost 0, `false`, če je vsaj eden od operandov enak nič.

⋆ `|` **Logični ali** ali disjunkcijo označimo v Matlabu z znakom `|`. Matlab primerja istoležne elemente v operandih in zapiše v rezultat logično vrednost 1, `true` , če je vsaj en operand različen od nič in logično vrednost 0, `false`, če sta oba operanda enaka nič.

⋆ `~` **Logični ne** označimo z znakom `~`. To je enočlena operacija, ki danemu polju realnih ali logičnih vrednosti priredi polje logičnih vrednosti enakih dimenzij. Vrednostim, ki so različne od nič priredi logično vrednost 0, `false`, medtem ko vrednostim, ki so enake nič, priredi logično vrednost 1, `true`.

⋆ `xor` Operacija `xor` je **izključujoči ali**. Ta priredi paru logičnih ali realnih polj polje istega reda. Elementi rezultirajočega polja zavzamejo vrednost 0, `false`, če sta istoležna elementa oba različna od nič ali oba enaka nič, in vrednost 1, `true` sicer.

⋆ `all, any` Funkciji, definirani nad vektorji `all` in `any`, danemu vektorju realnih ali logičnih vrednosti priredita logično vrednost. Funkcija `all` priredi vrednost 0, `false`, če je vsaj en element vektorja enak nič sicer pa priredi vrednost 1, `true`. Funkcija `any` priredi vrednost 0, `false`, če so vsi elementi vektorja enaki 0 in 1, `true` sicer. Če je argument funkcije polje, potem je rezultat vrstica vrednosti funkcije na stolpcih polja. Če želimo uporabiti funkcijo vzdolž vrstic, potem to povemo z drugim argumentom. Če je ta enak 2, potem funkcija vrne stolpec ustreznih rezultatov operacije na vrsticah, če pa je ta enak 1 potem je rezultat enak, kot v primeru enega samega argumenta.

Primer 8.3. *Poglejmo na primerih, kako delujeta funkciji* `all` *in* `any` *na poljih.*

Matlab

```
>> A=[1 0 2 3; 4 0 0 2; 1 2 3 4]

A =

     1     0     2     3
     4     0     0     2
     1     2     3     4

>> all(A)

ans =

     1     0     0     1

>> any(A)

ans =

     1     1     1     1

>> all(any(A))

ans =

     1

>> all(all(A))

ans =

     0

>> any(all(A))

ans =

     1

>> any(any(A))

ans =

     1

>> all(A,2)'

ans =

     0     0     1


>> any(A,2)'
```

```
ans =

     1     1     1

>>
```

Izraz `all(all(A))` *zavzame vrednost* `true`*, če so vsi elementi polja* `A` *različni od nič in* `false` *sicer. Če je vsaj en element polja* `A` *različen os nič potem je izraz* `any(any(A))` *enak* `true`*, sicer pa je enak* `false`*. Če je v vsakem stolpcu polja* `A` *vsaj en element različen od nič je vrednost izraza* `all(any(A))` *enaka* `true` *sicer pa je enaka* `false`*. Podobno je* `any(all(A))` *enako* `true`*, če so v vsaj enem stolpcu polja vsi elementi različni od nič.*

Poglejmo primer uporabe relacijskih in logičnih operatorjev. S pomočjo le-teh izbiramo ali/in spreminjamo elemente matrik, ki izpolnjujejo dane pogoje. Poglejmo, kaj se zgodi, če zapišemo `A(B)`, ko je matrika `B` matrika z naravnimi vrednostmi, ali pa v primeru, ko je matrika `B` matrika logičnih vrednosti.

Primer 8.4. *Elementi matrike* `B` *so cela pozitivna števila, največja vrednost ne presega števila elementov matrike* `A`*.*

Matlab

```
>> A=reshape(1:16,4,4)'

A =

     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16

>> B=[1 1 3 4; 2 3 3 4]

B =

     1     1     3     4
     2     3     3     4

>> A(B)

ans =

     1     1     9    13
     5     9     9    13

>>
```

Rezultat je matrika enakega reda, kot je matrika `B` in vsebuje elemente matrike `A` tistih indeksov (indeks pomeni v tem primeru položaj elementa v matriki, kakor je zapisana v pomnilniku glej stran), ki so zapisani v matriki `B`. Zato vrednosti elementov matrike `B` ne smejo presegati števila elementov matrike `A`.

Primer 8.5. *Poglejmo primer:*

Matlab

```
>> A(:)'

ans =

  Columns 1 through 12

    1     5     9    13     2     6    10    14     3     7    11    15

  Columns 13 through 16

    4     8    12    16

>> B(:)'

ans =

    1     2     1     3     3     3     4     4


>> C=A(B);
>> C(:)'

ans =

    1     5     1     9     9     9    13    13

>>
```

Vrstni red indeksiranja členov matrike `A` je enak vrstnemu redu členov v pomnilniku. Poglejmo, kaj se zgodi, če vsebuje matrika `B` logične vrednosti. V tem primeru mora imeti matrika `B` enako ali manj elementov od matrike `A`. Matlab postavi vektor elementov matrike `A` ob bok vektorju elementov matrike `B`, tako kot so zapisani v pomnilniku, in izbere v rezultirajoči vektor tiste elemente matrike `A`, ki jim ob boku stojijo logično pravilne vrednosti matrike `B`. Če ima matrika `B` manj elementov, se manjkajoči obravnavajo kot logično nepravilni.

Primer 8.6. *Poglejmo primer*

Matlab

```
>> A=reshape(1:16,4,4)'

A =

     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16

>> B=[1 1; 1 1; 0 0; 1 1]

B =

     1     1
```

```
     1     1
     0     0
     1     1

>> A(B)
??? Index into matrix is negative or zero.  See release notes on changes to
logical indices.

>> A(logical(B))

ans =

     1
     5
    13
     2
     6
    14

>>
```

V prvem primeru operacija ni bila dovoljena, ker so elementi matrike `B` *obravnavani kot indeksi elementov matrike* `A`*. Operacija ni možna ker so nekateri elemeni enaki nič. Nič ne more biti indeks elementa matrike. Ko pretvorimo matriko* `B` *v matriko logičnih vrednosti je, operacija dovoljena. Poglejmo natančneje:*

Matlab

```
>> A(:)'

ans =

  Columns 1 through 12

     1     5     9    13     2     6    10    14     3     7    11    15

  Columns 13 through 16

     4     8    12    16

>> B(:)'

ans =

     1     1     0     1     1     1     0     1

>> C=A(logical(B));
>> C(:)'

ans =

     1     5    13     2     6    14

>>
```

Starejše verzije Matlaba niso imele tipa `logical`. Takšno izbiranje elementov matrik, kot smo ga opisali zgoraj, se lahko izvede s pomočjo funkcije `find`.

Primer 8.7. *Kako? Poglejmo naslednji primer:*

Matlab

```
>> A(:)'

ans =

  Columns 1 through 12

    1     5     9    13     2     6    10    14     3     7    11    15

  Columns 13 through 16

    4     8    12    16

>> B(:)'

ans =

    1     1     0     1     1     1     0     1

>> C=find(B~=0)'

C =

    1     2     4     5     6     8

>> A(C)

ans =

    1     5    13     2     6    14

>>
```

Funkcija `find` *vrne vektor indeksov elementov matrike, na katerih je pogoj izpolnjen. S pomočjo vektorja indeksov izberemo ustrezne elemente matrike* `A`*, kot smo to storili v prejšnjem primeru.*

Poglejmo si primere uporabe tega, kar je bilo ravnokar povedano.

Primer 8.8. *Radi bi iz vektorju* `x` *odstranili element z danim indeksom.*

Matlab

```
>> x=[1 5 3 2 4 5 6 7]

x =

    1     5     3     2     4     5     6     7

>> y=x(1:end~=4)

y =
```

```
     1     5     3     4     5     6     7

>> y=x(1:end~=4&1:end~=2)

y =

     1     3     4     5     6     7

>> y=x(~ismember(1:end,[1 2 5]))

y =

     3     2     5     6     7

>>
```

V prvem primeru smo izločili četrti element vektorja, v drugem primeru četrtega in drugega, v zadnjem primeru pa prvega drugega in petega. Pri tem smo uporabili funkcijo `ismember`, ki pregleda ali, se elementi drugega argumenta nahajajo tudi v prvem. Več o tem bomo izvedeli s `help ismember`.

Primer 8.9. *Poglejmo še en primer. Postavimo vse elemente neke matrike* `A`*, ki presegajo dano vrednost, na neko določeno vrednost.*

Matlab

```
>> A=floor(rand(5)*10)

A =

     5     2     2     5     4
     4     5     3     7     3
     5     7     7     0     8
     3     5     6     6     0
     4     6     4     0     7

>> A(A>3)=-1

A =

    -1     2     2    -1    -1
    -1    -1     3    -1     3
    -1    -1    -1     0    -1
     3    -1    -1    -1     0
    -1    -1    -1     0    -1

>>
```

In še nekoliko bolj zapleten primer.

Matlab

```
>> A=floor(rand(5)*10)

A =
```

```
     9     2     4     6     2
     9     6     7     2     6
     7     3     2     8     6
     4     9     4     6     3
     4     7     9     1     5

>> B=floor(rand(5)*10)-5

B =

    -1    -2     1    -5    -5
    -5     1    -5    -2    -1
    -5    -5    -5     1    -1
    -2    -5    -4     2    -2
    -5     1     0     1    -4

>> A(A>6&B<0)=NaN

A =

   NaN     2     4     6     2
   NaN     6   NaN     2     6
   NaN     3     2     8     6
     4   NaN     4     6     3
     4     7     9     1     5

>>
```

8.3 Nekaj pomembnejših logičnih funkcij

V naslednji tabeli imamo opisane nekatere pomembne funkcije, ki vračajo logične vrednosti. Nekatere od njih se sprašujejo po tipu, po posebnih vrednostih, pripadnosti in enakosti spremenljivk.

Logične `is`-funkcije

`isempty`	prazno polje?
`isnan`	`NaN`?
`isinf`	$\pm$`inf`?
`isfinite`	končna vrednost?
`isequal`	polji sta identični?
`islogical`	polje logičnih vrednosti?
`ischar`	črkovni niz?
`isnumeric`	polje števil?
`isreal`	polje realnih števil?

Funkcije `isempty`, `isnan` in `isinf` se sprašujejo po posebnih vrednostih. Funkcije kot so `ischar`, `islogical`, `isnumeric`, `isfinite` in `isreal` se sprašujejo po tipu argumenta.

⋆ `isempty` Če želimo zvedeti, ali ja matrika prazna, uporabimo funkcijo `isempty`. Zapis `A==[]` v novejših verzijah Matlaba odsvetujejo, to pomeni, da bo v bližnji bodočnosti ta raba napačna, zato raje uporabimo funkcijo `isempty`. Lahko pa uporabimo funkcijo `isequal`.

Matlab
```
>> A=[]

A =

     []

>> isequal(A,[])

ans =

     1

>> isempty(A)

ans =

     1

>> A==[]
Warning: X == [] is technically incorrect. Use isempty(X) instead.

ans =

     1

>>
```

⋆ `isnan` Če hočemo vprašati, ali je dana vrednost `NaN`, potem se moramo vprašati z `isnan` in ne z `a==NaN`, ker bomo v drugem primeru vedno dobili logično 0, `false`.

Matlab
```
>> a=NaN; b=a

b =

   NaN

>> a==b

ans =

     0

>> isequal(a,b)

ans =

     0

>> isnan(a)

ans =
```

```
     1

>> all(isnan([a,b]))

ans =

     1

>>
```

Glej primer 8.2.

⋆ `isinf` Se vpraša, ali je vrednst neskočna.

Matlab

```
>> 1/0
Warning: Divide by zero.

ans =

   Inf

>> isinf(1/0)
Warning: Divide by zero.

ans =

     1

>> isinf(-1/0)
Warning: Divide by zero.

ans =

     1

>> 1/0==Inf
Warning: Divide by zero.

ans =

     1

>> -1/0==Inf
Warning: Divide by zero.

ans =

     0

>>
```

Matlab loči pozitivno od negativne neskončnosti.

⋆ `isfinite` Funkcija `isfinite` je negacija od `isinf`. Izraz `isfinite(a)==~isinf(a)`, razen v nekaterih posebnih primerih, dal pravilno vrednost.

⋆ `isequal` Funkcija odgovori na vprašanje, ali sta argumenta identična. Večinoma pomeni `isequal(a,b)` isto kot `a==b`, z nekaj posebnostmi, ki smo jih omenili zgoraj.

⋆ `islogical` Se vpraša ali je argument tipa `logical`.

Matlab

```
>> clear
>> A=1

A =

     1

>> b=islogical(A)

b =

     0

>> C=A|A

C =

     1

>> d=islogical(C)

d =

     1

>> whos
  Name      Size         Bytes  Class

  A         1x1              8  double array
  C         1x1              8  double array (logical)
  b         1x1              8  double array (logical)
  d         1x1              8  double array (logical)

Grand total is 4 elements using 32 bytes

>>
```

⋆ `ischar` Funkcija `ischar` vrne vrednost `true`, če je argument črkovni niz, sicer pa vrne vrednost `false`.

⋆ `isnumeric` Funkcija `isnumeric` se sprašuje, ali je argument numeričen. Funkcija vrne `true`, če je argument realen, kompleksen ali logičen in `false`, če je argument črkovni niz. Poglejmo nakaj primerov.

Matlab

```
>> isnumeric(A)
```

```
ans =

     1

>> isnumeric(C)

ans =

     1

>> B=1+4*i

B =

   1.0000 + 4.0000i

>> isnumeric(B)

ans =

     1

>> isreal(B)

ans =

     0

>> D='abc'

D =

abc

>> ischar(D)

ans =

     1

>> ischar(A)

ans =

     0

>> whos
  Name      Size         Bytes  Class

  A         1x1              8  double array
  B         1x1             16  double array (complex)
  C         1x1              8  double array (logical)
  D         1x3              6  char array
  b         1x1              8  double array (logical)
```

```
  d         1x1              8  double array (logical)

Grand total is 8 elements using 54 bytes

>>
```

`isreal` Vrne vrednost `true`, če je argument realen, oziroma tipa logical in `false`, če je kompleksen, ali če je črkovni niz. Za podrobnejša navodila naj bralec uporabi `help`.

8.4 Operacije nad množicami

Operacije nad množicami	
`ismember`	pripada množici?
`unique`	vrne elemente polja brez ponovitev
`intersect`	presek
`union`	unija
`setdiff`	razlika množic
`setxor`	simetrična razlika množic

⋆ `unique` Funkcija `unique` vrne elemente polja brez ponovitev urejene v naraščajočem vrstnem redu.

Matlab

```
>> x=fix(3*rand(1,10))

x =

     2     0     1     1     2     2     1     0     2     1

>> y=unique(x)

y =

     0     1     2

>>
```

⋆ `intersect` Funkcija `intersect` sprejema dve polji in vrne polje elementov, ki so skupni obema, brez ponavljanja v naraščajočem vrstnem redu.

Matlab

```
>> x=1+fix(3*rand(1,10))

x =

     2     3     3     3     1     2     3     3     2     3

>> y=2+fix(3*rand(1,10))

y =

     2     3     4     2     2     2     2     3     2     2
```

```
>> intersect(x,y)

ans =

     2     3

>>
```

⋆ `union` Funkcija `union` združi elemente dveh polj brez ponavljanja urejene po vrstnem redu.

Matlab

```
>> union(x,y)

ans =

     1     2     3     4

>>
```

⋆ `setdiff` Funkcija `setdiff` *(razlika množic)* sprejme dve polji in vrne elemente prvega polja, ki se ne nahajajo v drugem, elementi v rezultirajočem polju se ne pojavljajo in so v naraščajočem vrstnem redu.

Matlab

```
>> setdiff(x,y)

ans =

     1

>>
```

⋆ `setxor` `setxor` *(simetrična razlika množic)* sprejme dve polji in vrne elemente, ki se nahajajo samo v enem od njih, spet brez ponavljanja, urejene v naraščajočem vrstnem redu.

Matlab

```
>> setxor(x,y)

ans =

     1     4

>>
```

9 Nizi

Nizi so enodimenzionalna polja *(vektorji)* `ASCII` znakov.

Primer 9.1. *Inizializiacija niza:*

Matlab
```
>> a='primer niza'

a =

primer niza

>> a(1)

ans =

p

>> a(1:2:end)

ans =

pie ia

>>
```

Posamezne komponente izbiramo tako, kot pri običajnih poljih. Tudi nekatere funkcije, ki jih poznamo od tam, delujejo nad nizi.

Matlab
```
>> length(a)

ans =

    11

>> size(a)

ans =

     1    11

>> a'

ans =

p
r
i
m
e
r

n
i
z
a

>>
```

Določili smo dolžino niza, pogledali dimenzije polja in transponirali niz. Nize lahko podaljšujemo in tvorimo dvodimenzionalna polja znakov. Pogoj je seveda enak kot pri običajnih poljih, dolžine vrstic morajo biti enake.

Matlab

```
>> b=[a,' in njegov podaljsek']

b =

primer niza in njegov podaljsek


>> A=['abs';'123']

A =

abs
123

>> size(A)

ans =

     2     3

>> B=[A;'AB']
???  All rows in the bracketed expression must have the same
number of columns.

>> B=[A;'ABC']

B =

abs
123
ABC

>>
```

9.1 Funkcije za delo z nizi

Funkcije, ki preverijo vsebino niza

Preverjanje vsebine nizov	
`ischar`	ali je znakovni niz
`isletter`	ali niz vsebuje črkovne zake standarda `ISO Latin-1`
`isspace`	ali je presledek

⋆ `ischar` Funkcijo `ischar` smo že omenili. Preveri, ali gre za zankovni niz.

⋆ `isletter` Preveri ali dan niz vsebuje črkovne `ASCII` znake po standardu `ISO Latin-1`.

Matlab
```
>> isletter('ABab123&^@-+')

ans =

     1     1     1     1     0     0     0     0     0     0     0     0

>>
```

⋆ `isspace` Funkcija `isspace` se vpraša po presledku.

Matlab
```
>> a

a =

primer niza

>> isspace(a)

ans =

     0     0     0     0     0     0     1     0     0     0     0

>>
```

Preoblikovanje nizov

Preoblikovanje nizov	
`lower`	velike črke nadomesti z usreznimi malimi
`upper`	male črke nadomesti z ustreznimi velikimi
`strrep`	zamenjava podniza

⋆ `lower` Nadomesti velike črke z ustreznimi malimi.

Matlab
```
>> a='Minister Gregor pa nic'

a =

Minister Gregor pa nic

>> lower(a)

ans =

minister gregor pa nic

>>
```

⋆ `upper` Nadomesti male črke z ustreznimi velikimi.

Matlab
```
>> upper(a)

ans =

MINISTER GREGOR PA NIC

>>
```

⋆ `strrep` Nadomesti podniz v danem nizu z novim podnizom.

Matlab
```
>> a

a =

Minister Gregor pa nic

>> strrep(a,'Gregor','Grizold')

ans =

Minister Grizold pa nic

>> strrep(a,'Gregorij','Grizold')

ans =

Minister Gregor pa nic

>>
```

Nadomesti se niz `'Gregor'`, medtem ko niz `'Gregorij'` ne obstaja, in ga zaradi tega seveda ni mogoče zamenjati.

Iskanje podnizov

Iskanje podnizov	
`findstr`	iskanje podniza
`strcmp`	primerjanje nizov, loči velike in male črke
`strcmpi`	primerjanje nizov, ne loči velikih in malih črk
`strtok`	prva beseda

⋆ `findstr` Funkcija `findstr` poišče podniz v danem nizu.

Matlab
```
>> a='Za gotovo danes bo srecen dan'

a =

Za gotovo danes bo srecen dan
```

```
>> findstr(a,'da')

ans =

    11    30

>> findstr(a,' ')

ans =

     3    10    16    19    22    29

>> findstr(a,'dan')

ans =

    11    30

>> findstr(a,'danes')

ans =

    11

>>
```

⋆ `strcmp` Funkcija `strcmp` primerja nize. Loči male in velike črke.

Matlab

```
>> strcmp('abc','abd')

ans =

     0

>> strcmp('abc','ABC')

ans =

     0

>> strcmp('abc','abc')

ans =

     1

>>
```

⋆ `strcmpi` Funkcija `strcmpi` podobno kot funkcija `strcmp` primerja nize, le da ne loči malih in velikih črk.

Matlab
```
>> strcmpi('abc','ABC')

ans =

     1

>> strcmpi('abc','aBc')

ans =

     1

>>
```

⋆ `strtok` Funkcija `strtok` poišče prvo besedo niza.

Matlab
```
>> a='Za gotovo danes bo srecen dan'

a =

Za gotovo danes bo srecen dan

>> [T,Z]=strtok(a,' ')

T =

Za


Z =

 gotovo danes za bo srecen dan

>>
```

Drugi argument v funkciji `strtok` je znak, ki loči prvo besedo od ostalega niza. Ta seveda ni nujno presledek, lahko je poljuben ASCII znak. Če drugi argumen ni iz sestavlen iz enega samega znaka ampak jih je več, potem vsak od njih pomeni ločnico. Torej prva beseda se konča na mestu, ker se v nizu prvič pojavi katerekoli znak iz drugega argumenta.

Matlab
```
>> a='123,1231 1234;1234;2';
>> [b,a]=strtok(a,', ;')

b =

123


a =
```

```
,1231 1234;1234;2

>> [b,a]=strtok(a,', ;')

b =

1231


a =

 1234;1234;2

>> [b,a]=strtok(a,', ;')

b =

1234


a =

;1234;2

>> [b,a]=strtok(a,', ;')

b =

1234


a =

;2

>>
```

Pretvorba nizov Funkcije, ki pretvarjajo nize v števila in obratno.

Pretvorba nizov	
`double`	`ASCII` code znakov v nizu
`int2str`	zapiše celo stevilo v niz
`num2str`	zapiše število v niz
`str2double`	predstavi (ASCII) niz z zapisom števila v število tipa `double`
`str2num`	predstavi niz s številom

⋆ `double` Pretvori niz v vektor ASCII vrednosti posameznih znakov v nizu. Vrednosti so seveda naravna števila, zato je ime nekoliko ponesrečeno, ker tip `double` pomeni predstavitev realnega števila v dvojni natančnosti. Sicer pa smo že omenili, da so bila v Matlabu v tej obliki predstavljena vsa števila in logične vrednosti. Iz tega zornega kota je uporaba tega imena bolj razumljiva. Funkcija `double` pretvarja tudi druge tipe zapisa števil v tip `double`. Več o uporabi te funkcije zvemo, če v komandno vrstico Matlaba zapišemo `help double`.

Matlab
```
>> double('abc')

ans =

    97    98    99

>> double('1223.4')

ans =

    49    50    50    51    46    52

>>
```

⋆ `int2str` Pretvori celo število v niz ASCII znakov, ki ga predstavljajo.

Matlab
```
>> a=int2str(123)

a =

123

>> ischar(a)

ans =

     1

>> a=int2str(123.51)

a =

124

>>
```

Če število ni celo, zapiše v niz celi del decimalnega števila.

⋆ `num2str` Številsko matriko (vektor) zapiše v matriko (vektor) *(ASCII)* znakov.

Matlab
```
>> a=num2str(12000.0001)

a =

12000.0001

>> ischar(a)

ans =

     1
```

```
>> a=num2str(120000000000000000.0001)

a =

1.200000e+17

>>
```

Funkcija `num2str` sprejme tudi dva argumenta. V tem primeru drugi argument pove na koliko veljavnih mest zapišemo število, torej koliko bo številk različnih od nič.

Matlab

```
>> a=num2str(randn(2,2),3)

a =

-1.15      1.19
 1.19   -0.0376

>> whos
  Name      Size         Bytes  Class

  a         2x15            60  char array
  ans       2x16            64  char array

Grand total is 62 elements using 124 bytes

>> a=num2str(12345678,3)

a =

1.23e+07

>>
```

⋆ `str2double` Pretvori *(ASCII)* niz z zapisom realnega števila v realno število predstavljeno v obliki `double`.

Matlab

```
>> a=str2double('123.51')

a =

  123.5100

>> a=str2double('120e-2')

a =

    1.2000

>> str2double(12)
```

```
ans =

   NaN

>>
```

Če argument ni črkovni *(ASCII)* niz, potem ta funkcija vrne vrednost `NaN`.

⋆ `str2num` Pretvori matriko črkovnih znakov v številsko matriko, če je to mogoče.

Matlab

```
>> S=['1 3';'3 4']

S =

1 3
3 4

>> s=str2num(S)

s =

     1     3
     3     4

>> whos
  Name      Size         Bytes  Class

  S         2x3             12  char array
  s         2x2             32  double array

Grand total is 10 elements using 44 bytes

>>
```

Če pretvorba ni mogoča funkcija vrne prazno matriko.

Matlab

```
>> clear a
>> S=['1 3';'3 a']

S =

1 3
3 a

>> s=str2num(S)

s =

     []

>>
```

Pretvorba med različnimi številskimi sistemi Sledijo funkcije, ki pretvarjajo številke med različnimi številskimi sistemi.

Številski sistemi

```
base2dec
 bin2dec
dec2base
 dec2bin
 dec2hex
 hex2dec
```

Poglejmo nekaj primerov. Natančnejši pregled funkcij prepuščamo bralcu.

Primer 9.2.

Matlab

```
>> a=1423; dec2hex(a)

ans =

58F

>> hex2dec('AFF')

ans =

        2815

>> base2dec('111',3)

ans =

    13

>> bin2dec('110')

ans =

     6

>> base2dec('110',2)

ans =

     6

>>
```

Ovrednotenje niza Eden izmed pomembnejših ukazov v Matlabu je `eval`. Ukaz `eval` izvrši ASCII niz v argumentu kot, da bi ga zapisali v komandno vrstico Matlaba. To omogoča, med drugim, tudi prenašanje funkcij kot argument v druge funkcije.

Ovrednotenje niza

`eval`

⋆ `eval` Poglejmo si delovanje ukaza `eval`. Vzemimo primer:

Matlab

```
>> a='x=3;';
>> eval(a)
>> x

x =

     3

>>
```

Definirajmo funkcijo $y = x^2 + 3 * x + 3$, tako da jo zapišemo v niz, in izračunajmo vrednost funkcije za različne x.

Matlab

```
>> f='x.^2+3*x+3';
>> x=0:0.4:2;
>> eval(f)

ans =

    3.0000    4.3600    6.0400    8.0400   10.3600   13.0000

>> x=-1:-0.1:-2;
>> eval(f)

ans =

  Columns 1 through 7

    1.0000    0.9100    0.8400    0.7900    0.7600    0.7500    0.7600

  Columns 8 through 11

    0.7900    0.8400    0.9100    1.0000

>>
```

10 Programiranje v Matlabu

10.1 Kontrolni stavki

Kontrolni stavki	
`if`	
`elseif`	
`else`	
`end`	stavek if
`switch`	
`case`	
`otherwise`	
`end`	stavek case
`while`	
`end`	while zanka
`for`	
`end`	for zanka
`continue`	nadaljevanje zanke
`break`	prekinitev zanke
`return`	povratek iz funkcije
`error`	prekinitev ob napaki
`lasterr`	vzrok napake
`try`	
`carch`	preusmeritev ob napaki

Matlab pozna osem kontrolnih stavkov:

⋆ `if` Kontrolni stavek `if`, skupaj z `elseif`, `else` in `and` izvrši skupino ali blok stavkov, le kadar so izpolnjeni določeni pogoji. S stavkom `if`...`end` v programu naredimo razvejišče. Argument je poljuben veljaven izraz imenujmo ga *pogoj*. Če zavzame funkcija `all(all(`*pogoj*`))` vrednost `true` pravimo, da je pogoj izpolnjen, če pa zavzame vrednost `false`, pa pogoj ni izpolnjen. Najpreprostejša oblika stavka je

`if ... end`

`if` *pogoj*

stavki$_1$

`end;`

Če dopuščamo alternativno, skupino stavkov, ki se izvrši kadar pogoj ni izpolnjen potem uporabimo stavek `if`...`else`...`end`.

Stavek `if ... else ... end`

`if` *pogoj*

stavki$_1$**`;`**

`else`

stavki$_2$**`;`**

`end;`

Če želimo početi več različnih reči vsako od njih kadar je izpolnjen nek določen pogoj, uporabimo verigo `if...elseif...else...end`.

if ... elsif ... else ... end

```
if   pogoj_1

        stavki_1;

elseif   pogoj_2

        stavki_2;

...

elseif   pogoj_n

        stavki_n;

else

        stavki_{n+1};

end;
```

Primer 10.1. *Poglejmo primera*

Program

```
if rem(a,2) == 0
    disp('a je sodo stevilo')
    b = a/2;
end

if n < 0
    disp('n mora biti pozitiven');
elseif  rem(n,2) == 0
        a = n/2;
else
        a = (n+1)/2;
end
```

Če je število `a` *v prvem primeru sodo število, dobimo izpis* `a je sodo stevilo`*, sicer pa ne izpiše ničesar. V drugem primeru dobimo sporočilo, da mora biti število* `n` *pozitivno, če temu ni tako, sicer pa nam program vrne v spremenljivki* `a` *polovično vrednost spremenlivke* `n`*, če je ta soda in polovično vrednost, zaokroženo navzgor, če je liha.*

⋆ `switch` Stavek `switch`, skupaj s `case`, `otherwise` in `end`, izvrši različne bloke stavkov, glede na vrednost določenega parametra. Njegova osnovna oblika je naslednja:

switch ... case ... otherwise ... end

```
switch parameter

case vrednost_1

        stavki_1;
```

```
...

case vrednost_n

        stavki_n;

otherwise

        stavki_{n+1};
end
```

Če vrednost parametra ni enaka nobeni vrednosti iz seznama `case`, potem se izvrši blok, ki sledi stavku `otherwise`. Blokov `case` je lahko poljubno število.

Primer 10.2. *Primer stavka* `switch`.

Program

```
switch input_num
case -1
        disp('minus ena');
case 0
        disp('nic');
case 1
        disp('plus ena');
otherwise
        disp('neka druga vrednost');
end
```

Enemu bloku `case` lahko pripada več različnih vrednosti.

Primer 10.3. *Primer takšnega stavka:*

Program

```
switch var
case 1
        disp('1')
case {2,3,4}
        disp('2 ali 3 ali 4')
case 5
        disp('5')
otherwise
        disp('nekaj drugega')
end
```

⋆ `while` Zanka `while` skupaj z `end` ponavlja skupino stavkov dokler je izpolnjen dan pogoj. Sintaksa je naslednja:

while ... end

```
while pogoj

        stavki

end
```

Primer **10.4.** *Primer uporabe stavka* `while`.

Program

```
k = 5;
n = 1;
x = 1;
while n <= k
        x=x*n;
        n = n+1;
end
```

Vrednost spremenljivke `x` *je na koncu enaka* $5! = 120$.

Zanko lahko predčasno zapustimo z ukazom `break`.

Primer **10.5.** *Poglejmo primer prekinitve zanke z uporabo ukaza* `break`. *Naslednji program nam vrne enak rezultat kot prejšnji.*

Program

```
k = 5;
n = 1;
x = 1;
while 1
        x=x*n
        if n==k,
                break,
        end
        n = n+1;
end
```

Ker je pogoj vedno izpolnjen, $1 \neq 0$ *pomeni logično pravilen izraz, imamo opravka z neskončno zanko.*

Zanke lahko tudi gnezdimo. Sledi primer gnezdene zanke.

Primer **10.6.** *Inicializirajmo matriko, tako da bo vrednost elementov matrike enaka vsoti indeksov vrstice in stolpca v katerem se nahaja,* $a_{ij} = i + j$.

Program

```
A=zeros(4,5);
m=4; n=5;
while m
        while n
                A(m,n)=m+n;
                n=n-1;
        end;
        n=5;
        m=m-1;
end
```

⋆ `for` Zanka `for` skupaj z `end` ponavlja skupino ukazov določeno število krat. Sintaksa je naslednja:

for ... end

```
for indeks=zacetek:prirastek:konec

        stavki

end
```

Če prirastka ne imenujemo je ta enak 1. Za pozitivne prirastke se izvajanje zanke konča ko indeks preseže končno vrednost. Za negativne prirastke pa se izvajanje zanke konča, ko indeks postane manjši od končne vrednosti.

Primer 10.7. *Naslednja zanka se izvede petkrat.*

Program

```
x(1)=3;
for i = 2:6
        x(i) = 2*x(i-1);
end
```

Primer 10.8. *Primer gnezdene zanke* `for`*. Inicializitajmo matriko:*

Program

```
m=4; n=5;
A=zeros(m,n)
for i = 1:m
    for j = 1:n
            A(i,j) = 1/(i + j - 1);
    end
end
```

Opomba 1. Poglejmo pomembno razliko med stavkoma `for` in `while`. Ko zapišemo na primer `for i=1:n`, se v pomnilniku inicializira spremenljivka, vektor, ki vsebuje števila od 1 do `n`. To je lahko zelo potratno, če je `n` veliko število. Če je `n`, na primer, `100000`, potem zasedemo samo s števcem skoraj 1 MB (mega bajtov) pomnilnika. Medtem ko v stavku `while` lahko vodimo števec, ki zaseda samo pomnilniku 8 bajtov, kolikor je potrebno za eno skalarno spremenljivko, torej `n` krat manj. Zato bomo v primerih, ko je `n` velik raje uporabljali stavek `while`.

⋆ `continue` Stavek `continue` je nov. Implementiran je bil šele v verziji 6 Matlaba. Stavek `continue` prepusti kontrolo naslednjemu prehodu `for` ali `while` zanke. To pomeni, da preskoči vse ukaze danega prehoda, ki mu sledijo.

Primer 10.9. *Poglejmo primer uporabe stavka* `continue`.

Program

```
a=floor(rand(1,100)*10);
count = 0;
for i=1:length(a)
        if a(i)~=1
                continue
            end
            count = count + 1;
end
```

Zgornji program prešteje število komponent vektorja a, ki so enake 1.

⋆ `break` Prekinitveni stavek `break` prekine izvajanje tekoče `for` ali `while` zanke. Program se nadaljuje z izvajatnjem stavkov, ki sledijo ločnici `end` na koncu zanke. Pri gnezdenih zankah se prekine izvajanje najbolj notranje zanke in kontrola programa se prestavi na en nivo višje.

Primer 10.10. *Poglejmo primer:*

Program
```
A=floor(rand(10)*10);
produkt=ones(10,1);
for i=1:10
            for j=1:10
                if A(i,j)==0
                        produkt(i)=0;
                    break
                end;
                produkt(i)=produkt(i)*A(i,j);
            end
end
count
```

Program zmoži elemente vrstic matrike `A`*. Če je komponenta matrike enaka nič, potem nadalnje množenje členov v tej vrstici nima smisla, ker vemo da bo rezultat enak nič. To smo realizirali s pomočjo ukaza* `break`*.*

⋆ `return` Stavek `return` konča izvajanje dane funkcije in kontrola se vrne v program na mesto, ki sledi klicu te funkcije. Ko se konča izvajanje funkcije se vedno izvrši ukaz `return`, četudi ga ne zapišemo. Lahko pa z njim predčasno končamo izvajanje funkcije. Več o funkcijah v matlabu bomo izvedeli v naslednjem razdelku.

⋆ `error` S prekinitvenim stavkom `error` sami sprožimo napako. Program se prekine in izpiše sporočilo, niz, ki smo ga zapisali v argument ukaza `error`.

Matlab
```
>> i=2; if(i==1), error('i ne sme biti enako ena'), end;
>> i=1; if(i==1), error('i ne sme biti enako ena'), end;
??? i ne sme biti enako ena

>>
```

⋆ `lasterr` Z ukazom `lasterr` lahko preberemo sporočilo zadnje napake. Če nadaljujem s stanjem, ki ga je zapustil za sabo niz savkov gornjega primera, bomo dobili

Matlab
```
>> lasterr

ans =

i ne sme biti enako ena

>>
```

Še enkrat se je izpisalo sporočilo zadnje napake.

⋆ `try` S stavkom `try` in `catch` spremenimo tok programa v primeru, ko je med izvajanjem prišlo do napake. Splošna oblika stavka `try`...`catch` je naslednja.

try ... catch ... end

```
try

        stavki_1

catch

        stavki_2

end
```

Izvajajo se stavki med ukazoma `try` in `catch` eden za drugim. Če pride do napake se kontrola programa prenese na blok `catch`. Če se pri izvajanju bloka `catch` zgodi napaka, se izvajanje programa ustavi razen, če ni znotraj bloka gnezden še kakšen stavek `try`...`catch`.

Matlab

```
>> a=2:10

a =

     2     3     4     5     6     7     8     9    10

>> i=100; try, a(i), catch, -1, end;

ans =

    -1

>>
```

Ko smo hoteli izbrati vrednost `a(100)`, je prišmo do napake, ker ima vektor `a` samo 10 komponent. Ko je prišlo do napake je kontrolo prevzel blok `catch`, kjer se izpiše vrednost -1. Z ukazom `lasterr` lahko preberemo vzrok napake v bloku `try`. Vzemimo isti vektor `a` kot v gornjem primeru.

Matlab

```
>> lasterr('')
>> i=8; try, a(i), catch, -1, end; lasterr

ans =

     9


ans =

     ''


>> i=80; try, a(i), catch, -1, end; lasterr

ans =
```

```
    -1


ans =

 Index exceeds matrix dimensions.

>>
```

Z uakzom `try` in `catch` lahko ulovimo tudi napake, ki jih sami sprožimo.

Matlab

```
> i=1; try, if(i==1), error('i ne sme biti enako ena'), end, catch, -1, end; lasterr

ans =

    -1


ans =

i ne sme biti enako ena

>>
```

Ulovili smo napako, ki smo jo sami prožili. Izvedel se je stavek `catch`. Z ukazom `lasterr` se je izpisalo naše spotočilo, ki smo ga zapisali v argument stavka `errorr`.

Poglejmo nekaj primeov preprostih programov v Matlabu.

Primer 10.11. *Izračunajmo vrednost polinoma*

$$a_1 x^n + a_2 x^{n-1} + \cdots a_n x + a_{n+1}$$

v neki točki `x`*. Koeficienti nahajajo v vektorju* `a`*; Vrednost polinoma vrnemo v spremenljivki* `px`*.*

Program

```
% inicializacija
        n=5;
        a=fix(rand(1,5)*10);
        x=1/2;
% racun
        px = a(1);
        for k=2:n+1
                px=a(k) + px*x;
        end;
```

Izračunajmo kvadratni koren s pomočjo iteracije

$$x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$$

Izberimo začetno vrenost in x_0 *in parameter a ter sploži mo iteracijo. če je izačetna vrednost ptavilo izbrana in, če je vrednost parametra* $a > 0$*, potem ima zaporedje* x_n *limito* $\sqrt{a}$*.*

Program

```
% inicializacija
        x0=1;
        a=5;
        epsilon=10^(-5);
% racun
        x1=1/2*(x0+a/x0);
        while abs(x1-x0)>epsilon
                x=1/2*(x1+a/x1);
                x0=x1;
                x1=x
        end;
```

Ko zapišemo gornje vrstice v vrstico za vrstico v Matlabovo komandno vrstico, se izvede program in dobimo naslednji izpis:

Matlab

```
x0 =

     3


x0 =

   2.33333333333333


x0 =

   2.23809523809524


x0 =

   2.23606889564336

>> sqrt(5)

ans =

   2.23606797749979

>>
```

Ko sta dve iteraciji dovolj blizu skupaj, absolutna vrednost njune razlike je pod `epsilon`*, končamo izvajanje programa. Rezultat se pojavi v spremenljivki* `x`*. Ker so lihi členi zaporedja približkov manjši od limitne vrednosti, sodi pa večji, potem razliko dveh sosednjih približkov lahko uporabimo za oceno napake. V našem primeru smo dobili vrednost* $\sqrt{5}$ *izračunano na 5 decimalnih mest natačno. Gornji račun korena lahko realiziramo tudi z neskončno zanko.*

Program

```
% inicializacija
        x0=1;
        a=5;
        epsilon=10^(-5);
```

```
% racun
        while 1
                x=1/2*(x0+a/x0);
                if abs(x0-x)<epsilon
                        break
                end;
                x0=x
        end;
```

11 Programske in funkcijske datoteke

Programske in funkcijske datoteke morajo imeti koničnico imena `.m` in se morajo nahajati v drevesu poti, kjer Matlab išče svoje datoteke, rekli bomo, da mora biti datoteka dostopna Matlabu. Programskim in funkcijskim datotekam bomo rekli kar datoteke tipa `.m`. Na datoteke tipa `.m` zapisujemo definicije funkcij in programe, redkeje podatke. S pomočjo teh datotek dodajamo Matlabovem sistemu programskih knjižnic našo kodo.

11.1 Nastavitve

Pred vsem drugim si poglejmo, kako prilagodimo nastavitve v Matlabu našim potrebam. Kako dodamo poti, kjer Matlab išče svoje datoteke, in kako aktiviramo naše nastavitve pri inicializaciji sistema.

Drevo poti	
`path`	drevo aktivnih poti
`addpath`	doda vejo k drevesu poti
`rmpath`	odstrani vejo v drevesu poti
`startup`	začetne nastavitve
`what`	poišče Matlabove datoteke
`why`	odgovori na poljubno vprašanje

⋆ `path` Funkcija `path` vrne seznam aktivnih poti, to so poti, kjer se nahajajo datoteke tipa `.m`, ki so dostopne v Matlabu. Dostopamo jih tako, da v komandno vrstico zapišemo njeno ime brez končnice `.m`.

⋆ `addpath` S funkcijo `addpath` dodajamo veje drevesu aktivnih poti.

⋆ `rmpath` S tem ukazom odvzemamo veje iz drevesa aktivnih poti.

⋆ `startup` Datoteka `startup.m`, ki se nehaja v mapi kjer zaženemo Matlab se izvede med zaganjanem. V njo zapišemo naše osebne nastavitve, kot na primer nove poti aktivnega drevesa itd.

⋆ `what` Ukaz `what` izpiše seznam datotek tipa `.m` in drugih Matlabovih datotek, ki se nahajajo v trenutni mapi.

11.2 Programi

Najprej poglejmo, kako pišemo programske datoteke. Če je datoteka tipa `.m` dostopna Matlabu, potem se njena vsebina izvede, ko vpišemo v Matlabovo ukazno vrstico ime datoteke brez končnice `.m`. Njena vsebina se izvede kot, da bi vrstico za vrstico datoteke vpisovali v ukazno vrstico Matlaba. Vse uporabljene spremenljivke so globalne, zato moramo računati na to, da se bodo vrednosti spremenljivk, ki jih uporablja program, ustrezno spremenile.

Primer 11.1. *Zapišimo na datoteko z imenom* `vrstice.m` *naslednjo vsebino:*

`vrstice.m`

```
% primer programske datoteke
 % ta vrstica se z ukazom help ne izpise vec
    clear;
    n=10;
    x=1;
    while n
        x=x+(x+1);
```

```
        n=n-1;
        x=x+1;    % pojasnilo
    end;
```

Poglejmo, kakšna je vsebina pomnilnika po tem, ko smo poklicali program `vrstice`.

Matlab

```
>> vrstice
>> whos
  Name      Size         Bytes  Class

  n         1x1              8  double array
  x         1x1              8  double array

Grand total is 2 elements using 16 bytes

>> n

n =

     0

>> x

x =

  3070

>> help vrstice

  primer programske datoteke
```

Vrstice s pojasnilom se začnejo z znakom `%` *in končajo s prelomom vrstice. Te vrstice Matlab pri izvajanju preskoči. Ko vpišemo v ukazno vrstico* `help` *in ime datoteke, bomo dobili izpisane vrstice s pojasnilom, ki imajo v prvem stolpcu znak* `%` *in se nahajajo pred prvim stavkom, ki se izvede.*

11.3 Funkcije

Funkcije	
`function`	definicija funkcije
`nargin`	število vhodnih argumentov
`nargout`	število izhodnih argumentov
`return`	povratek iz funkcije

Funkcijske `.m` datoteke sprejemajo vhodne spremenljivke *(argumente)* in vračajo izhodne spremenljivke. Prvi stavek v funkcijskih datotekah mora biti stavek `function`. V tem stavku se določijo imena vhodnih in izhodnih spremenljivk funkcije. Za razliko od navadnega programa, so notranje spremenljivke lokalne, nastanejo v pomnilniku pri klicu funkcije in se pri povratku iz funkcije izbrišejo tako, da se ne spremenijo vrednosti obstoječim spremenljivkam v delovnem pomnilniku, ki nosijo enako ime.

⋆ `function` Sintaksa funkcijskega stavka je naslednja

function
```
function [a, b, c, ...]=ImeFunkcije(x, y, z, ...)
```

kjer so `x`, `y`, `z` ... vhodne spremenljivke to so argumenti funkcije, medtem ko so `a`, `b`, `c` ... izhodne spremenljivke, to so spremenljivke, ki jih funkcija vrača. Vhodne in izhodne spremenljivke so poljubnega tipa. Oglati oklepaj na levi strani enačaja je znak za seznam izhodnih spremenljivk. Vhodne in izhodne spremenljivke so lahko poljubnih tipov.

⋆ `nargin` Število vhodnih spremenljivk.

⋆ `nargout` Število izhodnih spremenljivk.

⋆ `return` Povratek iz funkcije.

Primer 11.2. *Zapišimo preprost primer funkcijske datoteke. Definirajmo funkcijo, ki sprejema eno spremenljivko in vrača vrednost v drugi spremenljivki. Funkcija naj vrača največjo absolutno vrednost elementov matrike. Imenujmo jo* `maxentry.m`.

maxentry.m
```
% MAXENTRY
%              MAXENTRY(A) najvecji element matrike A
%              po absolutni vrednosti.
function y=maxentry(x)
    y=max(abs(x(:)));
```

Primer klica funkcije. Poglejmo najprej kaj vrača ukaz `help` *z imenom funkcije.*

Matlab
```
>> help matxentry

  MAXENTRY
              MAXENTRY(A) najvecji element matrike A
              po absolutni vrednosti.

>>
```

Poiščimo največji člen matrike `A` *po absolutni vrednosti. Matriko dobimo takole:*

Matlab
```
>> clear
>> A=rand(5)-0.5

A =

    0.4501    0.2621    0.1154   -0.0943   -0.4421
   -0.2689   -0.0435    0.2919    0.4355   -0.1471
    0.1068   -0.4815    0.4218    0.4169    0.3132
   -0.0140    0.3214    0.2382   -0.0897   -0.4901
    0.3913   -0.0553   -0.3237    0.3936   -0.3611

>> a=maxentry(A)
a =

    0.4901
```

```
>> whos
  Name      Size         Bytes  Class

  A         5x5            200  double array
  a         1x1              8  double array

Grand total is 26 elements using 208 bytes
```

Ko smo preverili vsebino delovnega pomnilnika smo se prepričali, da ne vsebuje spremenljivk `x` *in* `y`*, ki so bile uporabljene v definiciji funkcije.*

Ime funkcije naj bo enako imenu datoteke, ne glede na to, da je za Matlab pomembno samo ime datoteke. Ime, ki je zapisano v funkcijskem stavku, se ne upošteva. V nadaljevanju bomo pogledali, kako pri definiciji funkcije lahko izvajamo različno kodo, glede na to koliko vhodnih spremenljivk ji ponudimo.

Primer 11.3. *Definirajmo funkcijo* `maxabs`*.*

maxabs.m

```
% MAXABS    Najvecja absolutna vrednost.
%    Za matrike MAXABS(X) je the najvecja absolutna vrednost v stolpcih X.
%
%    [Y,I] = MAXABS(X) vrne indekse najvecjih absolutnih vrednsoti v I.
%
%    MAXABS(X,Y) vrne polje enake velikosti kot sta  X in Y z vecjimi
%    absolutnimi vrednostmi vzetih iz X ali Y.
%    Ce sta razlicnih dimenzij, mora biti eden od njih skalar.
%
function [z,i]=maxabs(x,y)
    if nargin == 1 & nargout <= 2
            [z,i]=max(abs(x));
        [z,j]=max(z);
        i=[i,j];
    elseif nargin == 2 & nargout <= 1
            z=max(abs(x),abs(y))
    else
            error('napacno stevilo argumentov')
    end;
```

Če funkciji ponudimo en argument, vrača lahko dva. Če pa ji ponudimo dva argumenta, potem vrača samo enega.

Naj na tem mestu opozorimo na poseben primer klica funkcije v Matlabu. Če neka funkcija sprejema argumente tipa `char` in je ne kličemo v prireditvenem stavku, potem jo lahko pokličemo tako, da za imenom funkcije naštejemo argumente po vrsti, brez oklepajev in navednic, ločene s presledki.

Primer 11.4. *Definirajmo funkcijo, ki sprejema tri argumente tipa* `char`*. Imenujmo jo* `trifun`

trifun.m

```
function trifun(x,y,z)
% trifun primer funkcije, ki sprejema tri argumente tipa char
        disp(x), disp(y), disp(z)
```

Funkcija sprejema tri nize in jih izpiše na zaslon. Lahko jo pokličemo takole

Matlab
```
>> trifun ena dve tri
ena
dve
tri
```

ali pa, kot je običajno:

Matlab
```
>> trifun('ena','dve','tri')
ena
dve
tri
```

Kaj dobimo, če zapišemo

Matlab
```
>> sin 1

ans =

   -0.9538

>> x=sin 1
??? x=sin 1
        |
Missing operator, comma, or semi-colon.

>> x=sin('1')

x =

   -0.9538

>>
```

Ko smo funkcijo `sin` *na ta način smo dobili neko vrednost, čeprav funkcija* `sin` *sprejema samo numerični argument. Kaj smo torej dobili? Naj še enkrat opozorimo, da na ta način ne smemo klicati funkcije v priredtvenem stavku. Poglejmo kaj pomeni vrednost, ki smo jo dobili.*

Matlab
```
>> double('1')

ans =

    49

>> sin(49)

ans =

   -0.9538
```

Vidimo, da smo dobili enak rezultat, kot da bi `ASCII` *vrednost znaka vstavili v funkcijo* `sin`*. Torej* `sin 1` *da enak rezultat kot* `sin(double('1'))` *Poglejmo še en primer*

Matlab

```
>> sin to_je_pa_res_smesno

ans =

  Columns 1 through 7

    0.2367   -0.8646    0.6833   -0.7271    0.4520    0.6833   -0.8900

  Columns 8 through 14

    0.3796    0.6833    0.7850    0.4520    0.9454    0.6833    0.9454

  Columns 15 through 19

    0.8167    0.4520    0.9454   -0.0442   -0.8646
```

Kaj smo dobili?

11.4 Funkcija kot spremenljivka

Funkcije kot spremenljivke	
`eval`	ovrednotenje niza
`feval`	klic funkcije po imenu
`inline`	inline funkcijski objekt
`vectorize`	vektorizacija
`fcnchk`	preveri funkcijski argument

Pri pisanju programov se kmalu srečamo s problemom, kako prenesti funkcijo kot parameter v drugo funkcijo. V Matlabu lahko to storimo na več načinov. Naštejmo po vrsti:

⋆ `eval` Zapišemo funkcijo v spremenljivko tipa `char` kot črkovni niz z njeno definicijo. Na primer

Matlab

```
>> f='sin(x)+2*cos(x)^2'
```

Spremenljivka `f` se prenese v drugo funkcijo kot argument. Vrednost se izračuna tako, da se najprej inicializira spremenljivka `x` in kliče `eval(f)`. Funkcija `eval` izvrši niz `f`, kot da bi ga zapisali v ukazno vrstico Matlaba. Neugodno v tem primeru je, da moramo vedeti kako se imenuje neodvisna spremenljivka.

Matlab

```
>> f='sin(x)+2*cos(x)^2';
>> x=5;
>> eval(f)

ans =

   -0.7980

>> sin(5)+2*cos(5)^2
```

```
ans =

   -0.7980

>>
```

⋆ `feval` Spremenljivka je niz z imenom funkcije, ki je definirana Matlabu ali pa v datoteki tipa `.m`. V funkcijo, ki sprejema drugo funkcijo kot argument, prenesemo funkcijo preko argumenta tipa `char`, ki nosi ime prenesene funkcije. Funkcijske vrednosti računamo s pomočjo `feval`. Naj spregovori primer

Matlab

```
>> f='sin';
>> y=feval(f,3)

y =

    0.1411

>> z=2;
>> y=feval(f,z)

y =

    0.9093

>>
```

⋆ `inline` V novejših verzijah Matlaba lahko definiramo funkcije s pomočjo ukaza `inline`. S stavkom `inline` zgradimo funkcijski objekt, spremenljivko, ki se lahko kot argument prenaša v druge funkcije, tako kot vsaka druga spremenljivka. Prvi argument je niz z funkcijskim predpiso, ki mu sledijo imena spremenljivk. Sintaksa klica, funkcije definirane na ta način, je `f(x)`, če je `f` njeno ime in `x` neka inicializirana spremenljivka. Poglejmo:

Matlab

```
>> clear
>> ff=inline('sin(x)+2*cos(x)^2','x')

ff =

     Inline function:
     ff(x) = sin(x)+2*cos(x)^2

>> whos
  Name      Size         Bytes  Class

  ff        1x1            850  inline object

Grand total is 50 elements using 850 bytes

>> y=ff(3)

y =
```

```
    2.1013

>> g = inline('x^P1', 'x','P1')

g =

     Inline function:
     g(x,P1) = x^P1

>> g(2,3)

ans =

     8

>>
```

⋆ `vectorize` Funkcija `vektorize` sprejema spremenljivko tipa `char` in vrača spremenljivko tipa `char`. Računske operacije v nizu *vektorizira,* pretvori v operacije nad polji. Operacijo `*` nadomesti z `.*` in tako naprej. Poglejmo primer.

Matlab

```
>> x=1:10; y=1:10; f='x*sin(x)+x^2/y'

f =

x*sin(x)+x^2/y

>> eval(f)
??? Error using ==> *
Inner matrix dimensions must agree.

>> f=vectorize(f)

f =

x.*sin(x)+x.^2./y

>> eval(f)

ans =

  Columns 1 through 7

    1.8415    3.8186    3.4234    0.9728    0.2054    4.3235   11.5989

  Columns 8 through 10

   15.9149   12.7091    4.5598

>>
```

Ko smo operacije vektorizirali, smo lahko izračunali funkcijsko vrednost na vektrjih. Podobno pretvori `vectorize` `inline` funkcijski objekt.

Matlab
```
>> f=inline('x*sin(x)+x^2/y','x','y')

f =

     Inline function:
     f(x,y) = x*sin(x)+x^2/y

>> vectorize(f)

ans =

     Inline function:
     ans(x,y) = x.*sin(x)+x.^2./y
```

⋆ `fcnchk` Ko definiramo funkcijo, ki sprejema kot argument drugo funkcijo, bi želeli, da program dela pravilno, ne glede nato kako je prenešena funkcija definirana. V primeru, ko je funkcijski predpis zapleten, in ga ne moremo zapisati v eni sami vrstici, bomo funkcijo definirali v funkcijski datoteki tipa `.m`. Če pa je funkcijski predpis bolj preprost, običajno definiramo `inline` funkcijski objekt. Včasih bi radi prenesli niz z funkcijskim predpisom neposredno, ne da bi morali prej definirati `inline` funkcijski objekt. Stavek `fcnchk` preveri kako je funkcija, ki jo prenesemo preko spremenljivke, v resnici definirana. Definicijo pretvori v tako obliko, da jo lahko sprejme funkcija `feval`.

Prvi argument funkcije `feval` je ime funkcije, v novejših verzijah Matlaba je lahko tudi `inline` funkcijski objekt. Argumenti ki sledijo, pa so vrednosti neodvisnih spremenljivk, ki jih želimo vstaviti v funkcijski predpis. Definirajmo `inline` funkcijski objekt in poglejmo, kako s pomočjo `feval` izračunamo funkcijske vrednosti.

Matlab
```
>> f=inline('x*sin(x)+x^2/y','x','y')

f =

     Inline function:
     f(x,y) = x*sin(x)+x^2/y

>> feval(f,3,4)

ans =

    2.6734
```

Sedaj pa se vrnimo k `fcnchk`. Rekli smo, da spremeni funkcijski argument v obliko, ki jo sprejema `feval`. Če je argument tipa `char`, ki predstavlja izraz z definiranim funkcijskim pravilom, ga spremeni v `inline` funkcijski objekt, če pa argument tipa `char`, ki nosi samo ime funkcije, ga pusti nedotaknjenega in nazadnje, če je argument `inline` funkcijski objekt, takega tudi vrne. Preverimo delovanje ukaza `fcnchk`.

Matlab
```
>> f=fcnchk('x*sin(x)+x^2/y','x','y')

f =

    Inline function:
    f(x,y) = x*sin(x)+x^2/y

>> g=fcnchk('sin')

g =

sin

>> h=fcnchk(f)

h =

    Inline function:
    h(x,y) = x*sin(x)+x^2/y

>> whos
  Name      Size         Bytes  Class

  f         1x1            892  inline object
  g         1x3              6  char array
  h         1x1            892  inline object

Grand total is 145 elements using 1790 bytes
```

Funkcija `inline`, ko ji ne podemo spremenljivk, le-te izbere po določenem pravilu med znaki v nizu, ki določajo funkcijski predpis. Ta pravila tu ne bi razlalagali. Če bralca zanima kaj vec o tem si naj pogleda v `doc inline`.

Matlab
```
>> f=inline('sin(alfa)*pi')

f =

    Inline function:
    f(alfa) = sin(alfa)*pi

>> f=inline('sin(alfa)*x')

f =

    Inline function:
    f(alfa,x) = sin(alfa)*x

>> f=inline('sin(alfa)*pi','x','pi')

f =

    Inline function:
```

```
     f(x,pi) = sin(alfa)*pi

>>
```

Primer 11.5. *Zapišimo primer programa, kjer bomo prenašali funkcije kot argument. Vzeli bomo končno razliko, diferenčni kvocient, kot približek za odvod. Imenujmo funkcijo* `fd`*. Zapišimo njeno definicijo na datoteko* `fd.m`*.*

df.m

```
% FD APROKSIMACIJA ODVODA S KONCNO RAZLIKO
%        FD(F,X,H) je aproksimacija odvoda z
%        diferencnim kvocientom funkcije F v tocki X s korakom H.
%        ce korak H ni podan, potem se vzame vrednost SQRT(EPS)
%
function y=fd(f,x,h)
        if nargin < 3, h=sqrt(eps); end
        f=fcnchk(f);
        y=(feval(f,x+h)-feval(f,x-h))/(2*h);
```

1. *Najprej zapišimo definicijo funkcije, ki jo želimo odvajati, v datoteko* `fn.m`*. Naj bo to funkcija* $f(x) = x * \sin(x) - x^2$*. Vsebina datoteke je naslednja:*

fn.m

```
function y=fn(x)
        y=x.*sin(x)-x.^2;
```

2. *V drugem primeru definiramo niz* `f` *s funkcijskim predpisom.*

Matlab

```
>> f= vectorize('x*sin(x)+x^2')

f =

x.*sin(x)+x.^2
```

3. *in nazadnje še* `inline` *funkcijski objekt* `g`*, ki določa isto funkcijo.*

Matlab

```
>> g= inline(vectorize('x*sin(x)+x^2'))

g =

     Inline function:
     g(x) = x.*sin(x)+x.^2
```

Prepričajmo se, da lahko v kličemo funkcijo `fd` *z argumentom, ki nosi kateregakoli od zgoraj omenjenih načinov definicije funkcije.*

Matlab

```
>> x=linspace(-3,3,10)

x =
```

```
  Columns 1 through 7

   -3.0000   -2.3333   -1.6667   -1.0000   -0.3333    0.3333    1.0000

  Columns 8 through 10

    1.6667    2.3333    3.0000

>> y=fd('fn',x)

y =

  Columns 1 through 7

   -3.1711   -3.7780   -4.1692   -3.3818   -1.3088    1.3088    3.3818

  Columns 8 through 10

    4.1692    3.7780    3.1711

>> y=fd(f,x);

y =

  Columns 1 through 7

   -3.1711   -3.7780   -4.1692   -3.3818   -1.3088    1.3088    3.3818

  Columns 8 through 10

    4.1692    3.7780    3.1711

>> y=fd(g,x)

y =

  Columns 1 through 7

   -3.1711   -3.7780   -4.1692   -3.3818   -1.3088    1.3088    3.3818

  Columns 8 through 10

    4.1692    3.7780    3.1711

>> whos
  Name      Size         Bytes  Class

  f         1x14            28  char array
  g         1x1            844  inline object
  x         1x10            80  double array
  y         1x10            80  double array
```

```
Grand total is 81 elements using 1032 bytes
```

Stvarno kazalo