# 1 Uvod

Matlab je interaktivni sistem za numerične izračune. Numerični 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 knjižnicami.

- Omogoča hitro in enostavno kodiranje programov.
- Podatkovne strukture ne zahtevajo posebne pozornosti. Na primer, polja ni potrebno deklarirati pred inicializacijo.
- Na razpolago so kakovostna orodja za vizualizacijo.
- Program teče na več operacijskih sistemih oziroma platformah. Njegove `m`-datoteke so neodvisne od operacijskega sistema.
- V zadnjem času so dodali tudi Mapleov kernel in s tem omogočili, da lahko, podobno kot v Mathematici, v Matlabu računamo simbolično.
- Na razpolago so specialni 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.

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 pospeši 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.

## 1.1 UNIX

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

| Ukazi sistema UNIX | |
|---|---|
| `username` | *uporabniško ime* |
| `passwd` | *geslo* |
| `$` | *začetek komandne vrstice* |
| `.` | *trenutno vozlišče* |
| `..` | *eno vozlišče višje* |
| `cd` | *menjava vozlišč* |
| `/` | *ime najvišjega vozlišča* |
| `pwd` | *pot do trenutnega vozlišča* |
| `mkdir` | *kreiranje direktorija* |
| `rmdir` | *brisanje direktorija* |
| `ls` | *vsebina direktorija* |
| `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. Med ikonami na namizju izberemo tisto, ki odpre terminalsko okno v sistem. Ko kliknemo na ikono se odpre terminalsko okno. Na začetku komandne vrstice je izpisan znak `$`.

Ko smo odprli terminalsko okno se znajdemo na korenu domačega drevesa direktorijev. Od tod moramo priti na delovni direktorij. Na delovnem direktoriju bomo shranjevali naše programske in podatkovne datoteke, da jih bomo pozneje lažje našli. Po drevesu direktorijev se pomikamo z ukazom `cd` *(change directory)*. Da ne pišemo celotne poti, uporabljamo relativna imena. Ime direktorija v vozlišču kjer se trenutno se nahajamo je `.`, ime direktorija v vozlišču neposredno nad njim je `..`, ime direktorija najvišjega domačega vozlišča je $\sim$ medtem ko je ime korena sistema `/`. Če želimo priti iz direktorija
`/home/users/studenti/vaje`
na direktorij
`/home/users/studenti/vaje/nummet/MojaMapa`,
potem zadošča, da zapišemo samo relativno pot,
`$cd nummet/MojaMapa`
Šele potem, ko se prepričamo, da se nahajamo delovnem direktoriju začnemo delati. Prepričamo se z ukazom `pwd`. Z njim dobimo izpis celotne poti od korena sistema do direktorija na katerem se trenutno nahajamo. Z ukazom
`$cd ..`
se bomo vrnili na direktorij neposredno nad trenutnim. Zapišimo zaporedje ukazov
```
$pwd
$cd ..
$pwd
```
Če delovnega direktorija nimamo, ga naredimo z ukazom `mkdir` *(make directory)*. Postavimo se na vozlišče od koder bomo nadaljevati drevo z novim direktorijem. Na primer
`$cd /home/users/studenti/vaje/nummet`
S
`$pwd`
se prepričamo, da se nahajamo na pravem mestu. Zapišemo v komandno vrstico
`$mkdir MojaMapa`
Ukaz `ls` *(list)* izpiše vsebino trenutnega direktorija. Tako se prepričajmo, da je bil direktorij uspešno narejen. Če je drugih direktorijev preveč, lahko zahtevamo bolj selektivni izpis, na primer z

`$ls M*`
se bodo izpisali samo direktoriji, katerih ime se začne z veliko črko `M`. Lahko pa zapišemo tudi
`$ls MojaMapa`
Če želimo odstraniti direktorij, storimo to z ukazom `rmdir` *(remove directory)*. Postavimo se v vozlišče na katerem se nahaja in zapišemo
`$rmdir MojaMapa`
Z ukazom
`$ls MojaMapa`
se prepričamo ali je bil ukaz uspešno izvršen. Če želimo samo spremeniti njegovo ime, naredimo to z ukazom `mv` *(move)*. Postavimo se v vozlišče kjer se nahaja in zapišemo
`$mv MojaMapa TvojaMapa`
Če zapišemo
`$ls TvojaMapa`
ugotovimo ali je bilo preimenovanje uspešno izpeljano. Ukaz `cp` služi za kopiranje direktorijev in datotek. Na primer s
`$cp TvojaDatoteka MojaDatoteka`
dobimo kopijo datoteke `TvojaDatoteka` z novim imenom `MojaDatoteka`. Z `ls` se prepričamo, da se na tekočem direktoriju nahajata obe datoteki. Če želimo izvedeti kaj več o uporabi posameznega ukaza, na primer `cp`, zapišemo
`$ man cp`
Na zaslon se nam izpiše izčrpna razlaga kako se dan ukaz uporablja.

**Kako pišemo kopiramo in brišemo datoteke na disketi?** Sistem UNIX ne loči med disketno enoto in ostalimi diskovnimi enotami ter cdromi. Vse te enote so v sistemu veje v drevesu direktorijev. Običajno se nahaja direktorij, na katerega je pripeta vsebina diskete v disketni enoti na `/mnt/floppy`. To je odvisno od instalacije sistema. Seveda pa moramo disketno enoto prilepiti na drevo vsakokrat ko zamenjamo disketo. To počne ukaz `mount`. Da bi se tej komplikaciji izognili, so dodani programi, 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 direktorija na disketi* |
| `mcopy` | *kopiranje z/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 kopiramo z ukazom `mcopy` z opcijo `-t`.
`$ mcopy -t MojaDatoteka a:MojaDatoteka`
Datoteko `MojaDatoteka` smo napisali na disketo. Če želimo prepisati datoteko iz diskete na trenutni direktorij potem zapišemo:
`$ mcopy -t a:MojaDatoteka .`
Več o uporabi ukazov izvemo z `man`, ki mu sledi ime ukaza. Zapišite na primer
`$ man mcopy`
in poglejte, kaj je treba storiti, da se celotna vsebina diskete z drevesom direktorijev vred prepiše na trenutni direktorij. Priporočam uporabo teh ukazov nasproti originalnega UNIX-ovega ukaza `mount` in `umount` tudi zato, ker ti ohranijo dolga imena datotek operacijskega sistema WINDOWS.

## 1.2 Matlab

| začetek | |
|---|---|
| `matlab` | *vstop v Matlab* |
| `help` | *pomoč* |
| `lookfor` | *ključna beseda* |
| `format` | *formatiranje 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.

Najprej si poglejmo nekaj najosnovnejših reči za delo z Matlabom.

- Za vstop v Matlab 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 $\langle\hookleftarrow\rangle$. Matlab se bo odzval z izpisom rezultatov v naslednjih vrsticah terminala.

- Matlab loči male in velike črke. Pri imenih spremenljivk in ukazov moramo paziti, da jih pišemo z malimi in velikimi črkami natanko tako kot so jih definirani.

- 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.

- Puščica navzgor in puščica 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 zapustite tako, da vpišete ukaz `quit`.

Poglejmo tabelo vseh osnovnih ključev, 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* |
| ⟨Esc⟩ | *brisanje vrstice* |
| ⟨End⟩ | *na konec 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 v spremenljivko `ans`.

*Primer 1.1.* Ukazno vrstico lahko uporabljamo kot kalkulator.

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

ans =

  109.4560

>>
```

*Primer 1.2.* Če želimo izračunano vrednost shraniti, jo priredimo spremenljivki.

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

x =

    3

>> y=3*x+7

y =

    16

>>
```

Zberimo nekaj najpomembnejših pravil ki smo jih omenili.

**Pravilo 1.1.**

- *Ime spremenljivke je lahko sestavljeno iz velikih in malih črk, številskih znakov in podčrtajem. Ne sme se začeti 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 rezultat ne priredimo nobeni spremenljivki, se bo ta shranil v spremenljivko* `ans`*. Če ga želimo ohraniti, moramo njeno vsebino shraniti, ker se bo vrednost le-te slej ko prej prepisala.*

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

*Primer 1.3.* Poglejmo vsebino pomnilnika potem ko smo izvedli ukaze iz primerov 1.1 in 1.2.

```
>> 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

>>
```

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, ki se trenutno nahajajo v pomnilniku.

*Primer 1.4.* Nadaljujemo s stanjem pomnilnika, kot je bilo v prejšnjem primeru. Zapišimo:

```
>> clear a x
>> who

Your variables are:

ans  y

>> clear
>> who
>>
```

**Konstante** so v naprej inicializirane spremenljivke.

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

Primer take konstante je `pi`, katere vrednost je približno enaka $\pi$, kolikor dopušča aritmertika. Poleg `pi` najdemo v matlabu še konstanti `i` in `j`, katerih vrednost je imaginarna enota. Če konstanti priredimo kakšno drugo vrednost, se bo originalna prepisala z novo. Če želimo vspostaviti prvotno stanje preprosto zbrišemo spremenljivko, ki se skriva za imenom konstante.

```
>> 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 Matlab imajo večinoma vgrajeno aritmetiko s plavajočo piko, ki se podreja *IEEE* standardom. Logična funkcija `isieee` vrne rezultat 1, če imamo opraviti z aritmetiko tipa *IEEE* in vrednost 0, če ne. Od verzije 6 Matlaba naprej ta funkcija vedno vrne vrednost 1.

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

```
>> isieee

ans =

     1

>> computer

ans =

LNX86

>>
```

Matlabova aritmetika dvojne natančnosti je aritmetika s plavajočo piko in 64 bitov dolgo besedo. Območje aritmetike je od $-10^{-308}$ do $10^{308}$. V tej aritmetiki ne moremo predstaviti vsa realna števila, ker je dolžina besede končna. Tista realna števila, ki jih lahko predstavimo, bomo imenovali *predstavljiva števila*.

- Število, ki ga vrne funkcija `realmax` je največje predstavljivo realno število v dani aritmetiki.
- Če je rezultat računske operacije večje 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`.
- Konstanta `realmin` vrne najmanjše predstavljivo pozitivno število.
  *Primer 1.5.* Poglejmo primere

```
>> realmax

ans =

  1.7977e+308

>> -1.2*realmax

ans =

  -Inf

>> 2*realmax

ans =

   Inf

>>
```

- Nedoločenem izrazu Matlab priredi konstanto `NaN`, kratica za *not a number*, kar pomeni 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.

  *Primer 1.6.* Poglejmo si nekaj primerov

```
>> 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
>>
```

- Konstanata `eps` je razdalja med številom 1 in naslednjim predstavljivim številom na realni premici.

Če prištejemo številu 1 `eps`, dobimo predstavljivo število in `1+eps` bo različno od 1.

```
>> eps

ans =

   2.2204e-16

>> a=1+eps

a =

    1.0000

>> a-1

ans =

   2.2204e-16

>>
```

Če pa prištejemo številu 1 `eps/2`, potem to število ni predstavljivo, in rezultat bo postal natanko enak 1.

```
>> a=1+eps/2

a =

     1

>> a-1

ans =

     0

>>
```

Vidimo, da v aritmetiki s plavajočo piko končne natančnosti ne velja zakon asociativnosti in komutativnosti. Napako pri računanju lahko radikalno zmanjšamo, če izberemo primerno zaporedje operacij.

*Primer 1.7.* Poglejmo naslednja dva primera

```
>> 1+eps/2-1

ans =

     0
```

```
>> 1-1+eps/2

ans =

   1.1102e-16

>>
```

Še en primer, ki nas bo opozoril, kako moramo paziti na vrstni red operacij. 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.

```
>> 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.

*Primer 1.8.* Ko zapišemo v ukazno vrstico `help`, dobimo naslednji izpis

```
>> 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.
toolbox/local        -  Preferences.
...                  -  ...

>>
```

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

```
>> 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:

```
>> 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.

>>
```

**Pravilo 1.2.** *Sledil je kratek opis funkcije in spisek argumentov, ki jih sprejema in argumentov, 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!*

```
>> help ATAN2

ATAN2.m not found.

>>
```

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.

*Primer 1.9.* Poglejmo si primer, kako poiščemo vse funkcije, ki so povezane s ključno besedo `integral`.

```
>> 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)
>>
```

**Za formatiranje izpisa na zaslon** skrbi ukaz `format`. Na primer `format long` nastavi izpis na več decimalnih mest. Vpišimo v ukazno vrstico `help format`, če nas zanimajo še druge možnosti izpisa, ki jih ponuja `format`.

*Primer 1.10.* Ukaz `format` nastavi v naprej določen način izpisa na štiri decimalna mesta, ki ustreza načinu `format short`.

```
>> 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 vrednosti spremenljivk v obliki okrajšanega ulomka. Ostale možnosti, lahko pregleda bralec sam.

# 2 Matrike

Najpomembnejša podatkovna struktura v Matlabu je matrika oziroma polje. Dolgo časa je bila matrika tako rekoč edini podatkovni tip Matlaba. Šele novejše verzije imajo vgrajene druge podatkovne tipe. Matrike so dvodimenzionalna polja realnih števil dvojne natančnosti, `double`. Nove verzije Matlaba poznajo tudi večdimenzionalna polja - tenzorje, vendar o tem ob drugi priložnosti. Matrika reda $m \times n$ je polje števil razvrščenih v $m$ vrstic in $n$ stolpcev. Skalar je matrika reda $1 \times 1$.

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

## 2.1 Inicializacija matrik

Matrike lahko inicializiramo na več načinov. Najpreprostejši način je vnos matrike direktno preko ukazne vrstice, vendar je to za matrike z mnogo elementov 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 prenašamo tudi podatke med različnimi programi.

Poglejmo vse možne načine vnosa matrik v Matlab. Kot smo že omenili lahko matrike vnesemo

1. neposredno preko ukazne vrstice.
2. s funkcijami za tvorbo nekaterih posebnih matrik, glej funkcijo `gallery`.
3. preberemo datoteko, ki hrani matriko.

Poglejmo vse te načine vnosa vsakega posebej.

**Neposreden način vnosa matrik.**

*Primer 2.1.* Primera neposrednega vnosa matrik preko ukazne vrstice

```
>> 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 2.1.** *Elelemente matrike vpisujemo po vrsticah. Znotraj vrstice jih ločujemo z vejico ali presledkom, medtem ko vrstice ločujemo s podpičjem.*

Ukaz `size` vrne število vrstic in stolpcev matrike.

*Primer 2.2.* Preberimo število stolpcev in vrstic matrik iz primera 2.1.

```
>> 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 ena vrne število vrstic, če je enak dve pa število stopcev. Vzemimo matriko `A` iz primera 2.1.

```
>> 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`.

*Primer 2.3.* Naj spregovorita primera

```
>> 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 2.2.** *Dve matriki (lahko tudi več), na primer* `A` *in* `B`*, z enakim številom vrstic lahko združimo v novo matriko, ki ima vrstice matrike* `A` *podaljšane z vrsticami matrike* `B`*,* `C=[A,B]`*. 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* `C=[A;B]`*, ki ima stolpce matrike* `A` *podaljšane s stolpci matrike* `B`*. V tem primeru ločimo matrike s podpičjem.*

*Primer 2.4.* Poglejmo še bolj zapleten primer takšnega sestavljanja matrik.

```
>> 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* |

**Pravilo 2.3.** *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.*

1. 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.

   *Primer 2.5.* Poglejmo primere

   ```
   >> 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

>>
```

2. Funkcija `zeros` tvori matriko ničel.
   *Primer 2.6.* Poglejmo primera:

```
>> zeros(2)

ans =

     0     0
     0     0

>> zeros(3,2)

ans =

     0     0
     0     0
     0     0

>>
```

3. Funkcija `ones` tvori matriko enic ustreznega reda.

4. Funkcija `rand` tvori 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`.
   *Primer 2.7.* Poglejmo primer. Tvorimo matriko slučajnih vrednosti enakega reda kot matrika `A`.

```
>> 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

>>
```

5. 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.

   *Primer 2.8.* Naj spregovori primer

```
>> 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

>>
```

   *Primer 2.9.* Če je `A` skalarna matrika, dobimo matriko enakih vrednosti danega reda. Ta ukaz je mnogo hitrejši kot pa množenje z matriko `ones`.

```
>> 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

>>
```

6. Z `blkdiag` dobimo bločno diagonalne matrike.

   *Primer 2.10.* Naj spregovori primer

```
>> 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

>>
```

7. 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.

   *Primer 2.11.* Če želimo sestaviti vektor 5 ekvidistantnih vrednosti, ki se začne z vrednostjo 2 in konča z 11 potem bomo rekli

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

ans =

    2.0000    4.2500    6.5000    8.7500   11.0000

>>
```

8. Dvopičje `:` uporabljamo za sestavljanje ekvidistantnih vektorjev, kjer zahtevamo točno določeno vrednost razlike 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 $d$. Meje in prirastek so realna števila.

   *Primer 2.12.* Naj spregovorijo primeri

```
>> 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

>>
```

**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 2.13.* Sestavimo matriko slučajnih vrednosti reda $3 \times 5$. Element `A(2,3)` priredimo spremenljivki `a` in ga na koncu nadomestimo s 3;

```
>> 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 dvopičje uporabljamo tudi za izbiranje podmatrik.

*Primer 2.14.* Spet prepustimo razlago posebnim primerom

```
>> 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 velikost matrike, uporabimo besedico `end`, ki predstavlja število vrstic

oziroma stolpcev, glede na to kje se nahaja.

Elemente matrike lahko izbiramo tudi posredno.

```
>> 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 je $C(i,j) = A(a(i), b(j))$.

**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 2.15.* Poglejmo naslednje primere.

```
>> 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

```
>> A(:)'

ans =

     1     3     2     4

>>
```

S `A(:)`, smo zahtevali vektor vseh elementov matrike v vrstnem redu, kakor so zapisani v pomnilniku. `A(:)` je vektor stolpec. Transponirali smo ga, 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`. Naj spregovori naslednji primer.

```
>> 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 bi bil zunaj območja indeksov vrstic in stolpcev matrike, potem bo Matlab razširil prvotno matriko z ničelnimi elementi minimalno tako, da bo dan 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 2.16.* Poglejmo primera

```
>> 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

>>
```

Še en primer

```
>> 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

>>
```

**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 naprej v Matlabu, potem te podatke shranimo na datoteko, ki jo bo Matlab razumel, in jo preberemo v Matlabu.

Datoteka, ki hrani vrednosti matrike je lahko binarna ali tekstovna ASCII datoteka. V tekstovni datoteki hrani Matlab člene matrike, zapisane v obliki ASCII, po vrsticah. Vrstica besedila je vrstica matrike. Posamezni členi so ločeni s presledkom ali vejico. Vsaka vrstica mora imeti enako število elementov.

*Primer 2.17.* Sestavimo s pomočjo urejevalnika besedil datoteko z naslednjo vsebino:

```
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:

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

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

```
>> 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 se ne sme začeti s posebnim ali numeričnim znakom, in od posebnih znakov sme vsebovati samo podčrtaj.

S `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 tt 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. Tako 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.

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 delo19022002`. Stanje se bo shranilo na datoteko z imenom `delo19022002.mat`. Ko želimo z delom nadaljevati, zaženemo Matlab in zapišemo v ukazno vrstico `load delo19022002` 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`. Če pa želimo vrednost spremenljivke `a` shraniti na tekstovno datoteko z imenom `ax.txt`, potem to zapišemo takole

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

Z urejevalnikom besedil, se lahko prepričamo, kako je Matlab shranil vrednosti spremenljivke `a` na datoteko `ax.txt`.

*Primer 2.18.* Poglejmo si primere shranjevanja podatkov v Matlabu.

```
>> 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.

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

## 2.2 Algebrske operacije nad matrikami in polji

| 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* |
| `'` | *transponiranje matrik* |
| `( )` | *določanje vrstnega reda operacij* |

Za aritmetične operacije veljajo enaki zakoni prednosti, kot smo jih navajeni od drugod in kot 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 2.19.* Zapišimo nekaj primerov aritmetičnih izrazov in opazujmo prednost *(prioriteto)* operacij.

```
>> 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

>>
```

Opozoriti moramo, da potenciranje potenc Matlab računa od leve proti desni, medtem, ko v nekaterih drugih implementacijah velja, da je 3^3^3= $3^{3^3}$.

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

ans =

  19683

>> (3^3)^3

ans =

  19683

>> 3^(3^3)

ans =

7625597484987

>>
```

1. Vsota matrik je vsota po komponentah. Dimenzije sumandov se morajo ujemati.
   `C=A+B`, $c_{ij} = a_{ij} + b_{ij}$.
   *Primer 2.20.* Primer vsote matrik

```
>> 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

>>
```

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

   *Primer 2.21.* Poglejmo kakšna je razlika gornjih matrik

```
>> A-B

ans =

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

>>
```

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`, $c_{ik} = \sum_j a_{i,j} b_{j,k}$

   *Primer 2.22.* `>> 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

>>
```

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

   *Primer 2.23.* Poglejmo primer produkta polj

```
>> 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

>>
```

5. Desno deljenje matrik je množenje z inverzno matriko z desne. Matrika pod črto mora biti kvadratna.
   `C=A/B`, $A/B = A \cdot B^{-1}$

   *Primer 2.24.* Poglejmo primer desnega deljenja matrik.

```
>> 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

>>
```

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

7. Levo deljenje matrik je množenje z leve z inverzno matriko. Matrika pod črto mora biti kvadratna.
   `C=A\B`, $C = A^{-1} \cdot B$

*Primer 2.25.* 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`.

```
>> 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

```
>> A*x-b

ans =

   1.0e-15 *

    0.0139
   -0.1110
    0.0069
         0
    0.1110
```

```
>>
```

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

9. 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.

10. Potenciranje polja je potenciranje po komponentah.

11. Transponiranje matrike zamenja vrstice matrike s stolpci.

    *Primer 2.26.*
    ```
    >> 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

    >>
    ```

**Pravilo 2.4.** *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* $1 \times 1$ *ni dovoljeno.*

*Primer 2.27.* Naj morebitne nejasnosti razjasnijo naslednji primeri

```
>> 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 možno.

## 2.3 Elementarne funkcije

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* |

Morda bi 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. Kot merimo od $-\pi <$ atan2$(x, y) \leq \pi$.

| Eksponentna funkcija in logaritem | |
|---|---|
| `exp` | *eksponentna funkcija* $e^x$, *kjer je e osnova naravnih logaritmov* |
| `log` | *naravni logaritem* |
| `log10` | *desetiški logaritem* |
| `log2` | *logaritem z osnovo 2* |
| `pow2` | *eksponentna funkcija* $2^x$ |
| `sqrt` | *kvadratni koren* |

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

| 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* |

*Primer 2.28.* Poglejmo si nekaj primerov

```
>> 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`.

```
>> mod(10,3)

ans =

     1

>> rem(10,3)

ans =

     1
```

```
>> rem(-10,3)

ans =

    -1

>> mod(-10,3)

ans =

     2

>> mod(2+4,7)

ans =

     6

>> mod(2*4,7)

ans =

     1

>>
```

**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` | *maksimalna komponenta* |
| `max` | *minimalna 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.

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

*Primer 2.29.* Poglejmo primere

```
>> 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
```

```
>>
```

2. Funkciji `min` in `max` vračata vrednost najmanjše oziroma največje komponente vektorja. Funkciji vračata lahko tudi dve vrednosti, pri tem je druga indeks ustreznega elementa.

   *Primer 2.30.* Poglejmo si to na primeru funkcije `max`.

```
>> 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

>>
```

   Naj bo argument funkcije splošnejša matrika.

```
>> 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 členov v stolpcih in vrstica njihovih indeksov. Če želimo izračunati maksimalne vrednosti v vrsticah, potem je klic funkcije naslednji

```
>> 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 operacije `max(A,B)`, kjer iščemo večjega od istoležnih elementov dveh matrik, ki imata enako število stolpcev in vrstic. Poglej razliko
`>> max(1,3)`
in
`>> max(1,[],2)`

3. 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 2.31.* Natančno prouči naslednji primer.

```
>> 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 s `>> help sort`. Sorodna funkcija funkciji `sort` je `sortrows`. Bralec naj sam ugotovi kako deluje, za začetek si naj ogleda `>> help sortrows`.

4. Funkcija `sum` vrne vsoto komponent 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.

   *Primer 2.32.* Poglejmo primere uporabe

```
>> 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.

```
>> prod(1:5)

ans =

   120

>>
```

Če je argument splošnejša matrika,

```
>> 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`.

5. Funkciji `cumsum` in `cumprod` izračunata kumulativno vsoto in kumulativni produkt. Kot pri drugih funkcijah, lahko povemo ali želimo operacijo izvršiti po stolpcih ali po vrsticah.
   *Primer 2.33.* 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.

```
>> 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. Vpiši
`>> help vpa`

**Matrične funkcije**

| 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* |

1. Funkcija `triu` izloči zgornjetrikotno matriko. Lahko sprejema en argument, to je matrika, 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.

   *Primer 2.34.*

```
>> 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

>>
```

2. Podobno kot `triu` deluje funkcija `tril`, le da ta izbira člene matrike pod ustrezno diagonalo, torej

   *Primer 2.35.*

```
>>  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

>>
```

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

   (a) 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.
   *Primer 2.36.* `>> 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

>>
```

(b) Če je argument funkcije `diag` matrika, potem je rezultat vektor stolpec s komponentami iz glavne diagonale te matrike.
*Primer 2.37.* `>> 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

    >>
```

4. Funkcija `reshape` spremeni obliko matrike. Pri tem se ne spremeni število niti vrstni red elementov matrike v pomnilniku.

   *Primer 2.38.* Poglejmo naslednji primer

```
>> 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

>>
```

5. Funkcija `flipr` izbere stolpce matrike v obratnem vrstnem redu, medtem ko `flipud` izbere vrstice v obratnem redu, medtem ko `transpose` zamenja vrstice s stolpci.

   *Primer 2.39.* Poglejmo si nekaj primerov. Transponiranje matrike kratko označimo z `A'`.

```
>> 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

>>
```

| Nekaj pomembnih matričnih funkcij | |
|---|---|
| `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* |

1. Funkcija `null` vrne ortogonalno bazo ničelnega podprostora matrike, ali z drugimi besedami, lastnega podprostora, ki pripada lastni vrednosti 0.

2. Funkcija `orth` vrne ortogonalno bazo prostora, katerega generatorji so stolpci matrike.
3. Funkcija `rank` pove kolikšen je rang matrike ali z drugimi besedami, koliko je maksimalno število linearno neodvisnih vrstic oziroma stolpcev.
4. Funkcija `det` izračuna determinanto dane kvadratne matrike.
5. Funkcija `rref` naredi Gauss-Jordanovo eliminacijo na vrsticah matrike.
6. Funkcija `eig` vrne lastne vrednosti in lastne vektorje matrike.
7. Funkcija `cond` izračuna pogojenost matrike, to je $\|A\| \cdot \|A^{-1}\|$.
8. Funkcija `norm` poišče različne norme matrike.
9. Funkcija `inv` vrne inverzno matriko.
10. Funkcija `pinv` vrne psevdoinverzno matriko. Če je determinanta matrike različna od nič je psevdoinverzna matrika enaka inverzni matriki.
11. Z `svd` dobimo razcep matrike $A = U\Sigma V^{-1}$, kjer sta matriki $U$ in $V$ ortogonalni matriki, medtem ko diagonalna matrika $\Sigma$ nosi singularne vrednosti matrike $A$.

Poglejmo te funkcije v konkretnih primerih.

- Ničelni prostor matrike $A$ je množica vektorjev $x$ za katere velja $A\,x = 0$

```
>> 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-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.

- Poglejmo bazo prostora stolpcev matrike `A`

```
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

>>
```

Če bralec zna razložiti naslednji račun potem je razumel zgornji primer.

```
>> 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

>>
```

- Ukaz `rank` vrne dimenzijo prostora stolpcev, ki je enaka dimenziji prostora vrstic. Baza prostora stolpcev ima dva vektorja, torej dimenzija prostora stolpcev je 2. Rang matrike je 2.
- Funkcija `det` vrne vrednost determinante matrike. Matrika mora biti kvadratna. Če je determinanta matrike enaka 0 potem je rang matrike manjši od števila stolpcev oziroma vrstic.

```
>> 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

>> det(1./C)

ans =

     2.819050438097980e-08

>>
```

- Ukaz `rref` prevede matriko, s pomočjo elementarnh transformacija na vrsticah, na zgornjetrikotno matriko. Če je matrika kvadratna in 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, enako rangu prvotne matrike.

  **Pravilo 2.5.** *Elementarne transformacije na vrsticah so:*

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

```
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

>> rref(1./C)

ans =
```

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

>>
```

Z `rref` lahko poiščemo inverzno matriko ali rešujemo sisteme enačb.

```
>> 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 2.6.** *Ukaz* `rref` *pretvori s pomočjo elementarnih transformacij razširjeno matriko* $[A, B]$*, kjer je matrika* $A$ *kvadratna matrika v matriko* $[I, A^{-1}B]$*, kjer je* $I$ *enotna matrika enakega reda kot matrika* $A$*.*

```
>> D

D =

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

>> b=[2;3;4;7]

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

>>
```

- Ukaz `eig` vrne lastne vrednosti in lastne vektorje matrike.

```
>> [a,b]=eig(C)

a =

   -0.4140    0.8229   -0.2488    0.2960
```

```
   -0.4688    0.4219    0.6954   -0.0511
   -0.5236    0.0210   -0.6445   -0.7857
   -0.5785   -0.3800    0.1979    0.5408


b =

   36.2094         0         0         0
         0   -2.2094         0         0
         0         0    0.0000         0
         0         0         0   -0.0000

>> aa=C*a(:,1)

aa =

  -14.9908
  -16.9757
  -18.9606
  -20.9455

>> aa./a(:,1)

ans =

   36.2094
   36.2094
   36.2094
   36.2094

>>
```

- Psevdoinverzna matrika k matriki $A$ je matrika $P$, ki ima naslednje lastnosti

  1. matrika $P$ je enakega reda kot matrika $A'$.
  2. Velja $A * P * A = A$, $P * A * P = P$.
  3. Matriki $A * P$ in $P * A$ sta hermitski.
  4. Če je $\texttt{det}A \neq 0$, potem je $P = A^{-1}$.

  Psevdoinverzno matriko dobimo s pomočjo funkcije `svd`.

```
>> 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
```

Če obstaja inverzna matrika sta potem obe enaki.

```
>>D =

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

>> D^(-1)

ans =

    0.7000   -0.8000   -0.2000   -0.2000
   -0.0750    0.3000    0.2000   -0.3000
   -0.6500    0.6000    0.4000    0.4000
    0.2750   -0.1000   -0.4000    0.1000

>> pinv(D)
```

```
ans =

    0.7000   -0.8000   -0.2000   -0.2000
   -0.0750    0.3000    0.2000   -0.3000
   -0.6500    0.6000    0.4000    0.4000
    0.2750   -0.1000   -0.4000    0.1000

>>
```

- Funkcija `svd` (singular value decomposition) vrne razcep matrike po singularnih vrednostih. 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.

```
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 psev/-do/-in/-verz/-na matrika matrike `A`. Psovdoinverzna matrika `pinv(A)` je enaka matriki `V'*T*U`, kjer je matrika `T` izračunamo takole:

```
>> 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. V našem primeru je prag `10*eps`.

```
>> 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)

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 matrike `A` so kvadratni koreni lastnih vrednosti simetričnih matrik. `A'*A` in `A*A'`.

```
>> 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

>>
```

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

**Pravilo 2.7.** *Rešitev enačbe $A'A\,x = A'b$ minimizira zgornji izraz. $x = (A'A)^{-1}b$.*

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

```
>> 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

>>
```

# 3 Relacijski, logični operatorji in množice

## 3.1 Relacijski operatorji

| Relacijski operatorji | |
|---|---|
| `==` | *enako* |
| `~=` | *ni enako* |
| `<` | *manjše* |
| `>` | *večje* |
| `<=` | *manjše ali enako* |
| `>=` | *večje ali enako* |
| `logical` | *določi logični tip* |

Relacijski operatorji določajo dvočlene operacije nad parom realnih polj istega 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 označeno posebej, tako da se loči od polja realnih vrednosti 0 in 1.

  *Primer 3.1.* Poglejmo primer

```
>> 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 se Matlab zapomni polje logičnih vrednosti. Če postane takšno polje operand aritmetične operacije, potem ga Matlab obravnava kot polje realnih vrednosti 0 in 1.

```
>> 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`.

Če se vprašamo, ali je `NaN` enako `NaN` dobimo odgovor:

```
>> NaN==NaN

ans =

     0

>>
```

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

- Operator ∼= določa operacijo neenakosti. Če sta istoležna elementa v operandih različna je rezultat 1, `true`, če pa sta enaka potem je rezultat operacije 0, `false`.

  *Primer 3.2.* Vzemimo matriki `A` in `B` iz zgornjega primera.

```
>> 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.

  *Primer 3.3.* Vzemimo matriki `A` in `B` iz zgornjega primera.

```
>> 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

>>
```

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

  *Primer 3.4.* Poglejmo primer

```
>> 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 stare realne vrednosti ohranile.

## 3.2 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 operand 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 $\sim$. 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`.
- 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.
- Vektorski funkciji `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 kakšne druge dimenzije, 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 3.5.* Poglejmo na primerih, kako delujeta funkciji `all` in `any` na poljih.

```
>> 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.

## 3.3 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 | |
|---|---|
| `isequal` | *sta polji identični?* |
| `isempty` | *prazno polje?* |
| `isnan` | `NaN`*?* |
| `isfinite` | `inf`*?* |
| `ischar` | *črkovni niz?* |
| `islogical` | *polje logičnih vrednosti?* |
| `isnumeric` | *polje števil?* |
| `isreal` | *polje realnih števil?* |

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

*Primer 3.6.* Imena so dovolj zgovorna, zato si poglejmo samo nekaj primerov.

```
>> 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

>>
```

Funkcija `isnumeric` se sprašuje ali je argument numeričen. Funkcija vrne `true`, če je argument realen, kompleksen ali logičen.

```
>> 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

>>
```

Ostale funkcije naj bralec razišče sam. Za podrobnejša navodila naj uporabi `help`. Če hočemo vprašati, ali je dana vrednost `NaN`, potem moramo to storiti s funkcijo `isnan` ne pa z `a==NaN`, ker bomo v drugem primeru vedno dobili vrednost 0 `false`. Podobno moramo postopati, če hočemo izvedeti, da je dana matrika prazna. Zapis `A==[]` v novejših verzijah Matlaba odsvetujejo, zato uporabimo funkcijo `isempty`.

## 3.4 Operacije nad množicami

| Operacije nad množicami | |
|---|---|
| `find` | *indeksi elementov, ki ustrezajo pogoju* |
| `ismember` | *pripada množici?* |
| `unique` | *vrne elemente polja brez ponovitev* |
| `intersect` | *presek* |
| `union` | *unija* |
| `setdiff` | *razlika množic* |
| `setxor` | `xor` *množic* |

1. 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 3.7.* Elementi matrike `B` so cela pozitivna števila, največja vrednost ne presega števila elementov matrike `A`.

     ```
     >> 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`, ki vsebuje elemente matrike `A` tistih indeksov, ki so zapisani v matriki `B`. Zato vrednosti elementov matrike `B` ne smejo presegati števila elementov matrike `A`. Poglejmo kako so izbrani elementi:

```
>> 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.

- *Primer 3.8.* Poglejmo kaj se zgodi, če nosi 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 rezultirajoč 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. Poglejmo primer

```
>> 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 mora biti indeks elementa matrike. Ko pretvorimo matriko `B` v matriko logičnih vrednosti je operacija dovoljena. Poglejmo natančneje:

```
>> 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

>>
```

- V starejše verzije Matlaba imele tipa `logical`. Takšno izbiranje elementov matrik, kot smo ga opisali zgoraj, se lahko izvede s pomočjo funkcije `find`.
  *Primer 3.9.* Kako? Poglejmo naslednji primer.

```
>> 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 3.10.* Radi bi iz vektorju `x` odstranili element z danim indeksom.

```
>> 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. V zadnjem primeru smo uporabili funkcijo

`ismember`, ki pregleda ali se elementi drugega argumenta nahajajo tudi v prvem. Več o tem bomo izvedeli s `>> help ismember`. Postavimo vse elemente neka matrike `A`, ki presegajo dano vrednost na neko določeno vrednost.

```
>> 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

>>
```

Poglejmo še nekoliko bolj zapleten primer.

```
>> 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

>>
```

2. `unique` vrne elemente polja brez ponovitev, ki so urejeni v naraščajočem vrstnem redu.
3. `intersect` sprejema dve polji in vrne polje elementov, ki so skupni obema, brez ponavljanja v naraščajočem vrstnem redu.
4. `union` združi elemente dveh polj brez ponavljanja urejene po vrstnem redu.
5. `setdiff` 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.
6. `setxor` sprejme dve polji in vrne elemente, ki se nahajajo samo v enem od njih, spet brez ponavljanja, urejeni v naraščajočem vrstnem redu.

# 4 Programiranje v Matlabu

**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` | *poratek iz funkcije* |
| `try` | |
| `carch` | *preusmeritev ob napaki* |
| `error` | *prekinitev ob napaki* |
| `lasterr` | *vzrok napake* |

Matlab pozna osem kontrolnih stavkov:

- Kretnica `if`, skupaj z `elseif`, `else` in `and` izvrši določeno skupino ali blok stavkov, le kadar so izpolnjeni določeni pogoji.
- Kretnica `switch`, skupaj s `case`, `otherwise` in `end`, izvrši različne skupine stavkov, glede na vrednost določenega parametra.
- Zanka `while` skupaj z `end` ponavlja skupino stavkov dokler je izpolnjen dan pogoj.
- Zanka `for` skupaj z `end` ponavlja skupino ukazov določeno število krat.
- Skok `continue` prepusti kontrolo naslednjemu prehodu `for` ali `while` zanke to pomeni, da preskoči vse ukaze danega prehoda, ki mu sledijo.
- Prekinitveni stavek `break` prekine izvajanje `for` ali `while` zanke.
- S stavkom `try` in `catch` spremenimo tok programa, če je med izvajanjem prišlo do napake.
- Stavek `return` konča izvajanje dane funkcije in kontrola se vrne v program na točko, ker je bila klicana ta funkcija.
- S prekinitvenim stavkom `error` prekinemo program, ko je prišlo do napake.

Kontrolni stavki `if`, `while`, `for` in `switch` se zaključijo s stavkom `end`, ki določa konec dane skupine ukazov, oziroma kontrolnega bloka.

`if` S stavkom `if ... end` naredimo kretnico oziroma razvejišče v programu. Argument je logični izraz *pogoj*, izraz na katerem zavzame funkcija `all(all(`*pogoj*`))` vrednost `true` oziroma `false`. Najpreprostejša oblika je

```
if    pogoj
      stavki
end
```

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

```
if      pogoj_1
        stavki_1
else    pogoj_2
        stavki_2
end
```

Če želimo imeti več testov, uporabimo verigo `if ... elseif ... else ... end`.

```
if       pogoj_1
         stavki_1
elseif   pogoj_2
         stavki_2
...
else     pogoj_n
         stavki_n
end
```

Izvrši se samo blok stavkov, ki sledijo prvemu izpolnjenemu pogoju, oziroma blok, ki sledi `else` v primeru, da ni bil noben pogoj izpolnjen.

*Primer 4.1.* Poglejmo primera

```
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 ... case ... othervise ... end` izvršuje določen blok stavkov, glede na vrednost parametra. Njegova osnovna oblika je naslednja:

```
switch      parameter
case        vrednost_1
            stavki_1
case        vrednost_2
            stavki_2
otherwise
            stavki_3
end
```

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

*Primer 4.2.* Primer stavka `switch`.

```
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 4.3.* Primer takšnega stavka:

```
switch var
    case 1
        disp('1')
    case {2,3,4}
        disp('2 or 3 or 4')
    case 5
        disp('5')
    otherwise
        disp('something else')
end
```

`while` Zanka `while` izvaja skupino stavkov med `while` in `end`, dokler je izpolnjen pogoj. Sintaksa je naslednja:

```
while   pogoj
        stavki
end
```

*Primer 4.4.* Primer uporabe stavka `while ... end`.

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

Vrednost spremenljivke `x` je na koncu enaka $x = k! = 5! = 120$. Zanko lahko predčasno zapustimo z ukazom `break`. Poglejmo primer uporabe `break`. Naslednji program nam vrne enak rezultat kot prejšnji.

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

Zanka `while 1 ... end` je primer neskončne zanke. Zanke lahko tudi gnezdimo.

`for` Zanka `for` izvaja blok stavkov med `for` in `end` določeno število krat. Sintaksa je naslednja:

```
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 4.5.* Naslednja zanka se izvede petkrat.

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

*Primer 4.6.* Primer gnezdene zanke `for`.

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

Z ukazom `break` predčasno zapustimo zanko `for`.

*Opomba 1.* Zapišimo še eno pripombo v zvezi s stavkoma `for` in `while`. Ko zapišemo na primer `for i=1:10000`, se v pomnilniku inicializira spremenljivka, ki vsebuje števila od 1 do 10000, kar je zelo potratno. Zato bomo v takih primerih bomo raje uporabljali stavek `while`.

`continue` Stavek `continue` preda kontrolo naslednji iteraciji zanke `for` ali `while` tako, da preskoči stavke tekoče prehoda, ki mu sledijo. V gnezdenih zankah ukaz `continue` preda kontrolo naslednji iteraciji najbolj notranje zanke, ki ga vsebuje.

*Primer 4.7.* Poglejmo primer uporabe stavka `continue`.

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

*Primer 4.8.* Poglejmo primer:

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

Program prešteje elemente matrike `A`, ki so enaki nič.

Paragraph`try ... catch`

Splošna oblika stavka `try ... catch` je naslednja.

```
try
        stavki_1
catch
        stavki_2
end
```

Izvajajo se stavki med ukazoma `try` in `catch` eden za drugim, dokler se ne pride do napake. Ob napaki se kontrola programa prenese na blok `catch`. Z ukazom `lasterr` lahko preberemo vzrok napake. Če se pri izvajanju bloka `catch` zgodi napaka, se izvajanje programa ustavi razen, če ni znotraj bloka gnezden še kakšen stavek `try ... catch`.

`return` Ukaz `return` konča izvajanje funkcije in vrne kontrolo na točko programa takoj za klicem le-te. 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.

# 5 Datoteke tipa *.m*

## 5.1 Programi

| drevo poti | |
|---|---|
| `path` | *drevo poti* |
| `addpath` | *doda vejo k drevesu poti* |
| `rmpath` | *odstrani vejo v drevesu poti* |
| `strtup` | *ime datoteke tipa .m, kjer so zapisane začetne nastavitve* |
| `what` | *poišče Matlabove datoteke* |
| `why` | *odgovori na poljubno vprašanje* |

Zaporedje stavkov lahko zapišemo na datoteko tipa `.m`. Končnica imena datoteke more biti `.m` in se mora nahajati na direktoriju v drevesu poti. Ko pokličemo ime datoteke, brez končnice `.m` se bo začela njena vsebina izvajati, kot da bi vsebino datoteke vrstico za vrstico vpisovali v ukazno vrstico Matlaba. Vse njene spremenljivke so spremenljivke delovnega pomnilnika. Njihove vrednosti lahko preberemo, ko se program izvede. Pri uporabi takih datotek, moramo biti pazljivi. V delovnem pomnilniku se bodo vrednosti vseh spremenljivk, ki jih uporablja program ustrezno spremenile.

*Primer 5.1.* Zapišimo na datoteko z imenom *skript.m* naslednjo vsebino:

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

Zapišimo v ukazno vrstico Matlaba naslednje zaporedje ukazov

```
>> skript
>> 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 script

  primer programske datoteke
```

Vrstice s komentarjem se začnejo z znakom % in končajo s prelomom vrstice. Te vrstice Matlab pri izvajanju preskoči. Ko zapišemo `help` in ime datoteke, bomo dobili izpisane vrstice s komentarjem, ki imajo v prvi koloni znak % in se nahajajo pred prvim stavkom, ki se izvede.

## 5.2 Funkcije

| funkcije | |
|---|---|
| `function` | *deklaracija funkcije* |
| `nargin` | *število vhodnih argumentov* |
| `nargout` | *število izhodnih argumentov* |
| `return` | *povratek iz funkcije* |

Funkcijske `.m` datoteke sprejemajo vhodne argumente in vračajo izhodne argumente. Prvi stavek v funkcijskih datotekah more biti stavek `function`, ki definira kateri so vhodni argumenti in kateri so izhodni argumenti funkcije.

Za razliko od navadnega programa, so v tem primeru vse spremenljivke, ki jih inicializira program lokalne, nastanejo v pomnilniku pri klicu funkcije in pri povratku iz funkcije izbrišejo, tako da te spremenljivke navzven niso vidne niti nam ne pokvarijo obstoječih spremenljivk z enakim imenom v delovnem pomnilniku.

Tipični funkcijski stavek je

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

kjer so `x`, `y`, `z` itd. vhodni argumenti, medtem ko so `a`, `b`, `c` itd... argumenti, ki jih funkcija vrača. Vhodni argumenti so poljubnega tipa, prav tako tudi izhodni argumenti.

Oglati oklepaj na levi strani enačaja je znak za seznam izhodnih argumentov, ki so lahko različnih tipov.

*Primer 5.2.* Poglejmo kako sestavimo tipično funkcijo, ki sprejema en in vrača en argument. Zapišimo naslednjo vsebino na datoteko z imenom `maxentry.m`.

Običajno se pišejo imena funkcij in komentarji v angleškem jeziku ne glede na to kakšne narodnosti je avtor.

```
% MAXENTRY  Largest absolute value of matrix entries.
%     MAXENTRY(A) is maximum of absolute values
%               of the entries of A
function y=maxentry(x)
y=max(abs(x(:)));
```

Datoteko shranimo, in v komandno vrstico Matlaba vpišemo:

```
>> help maxentry
```

```
 MAXENTRY     Largest absolute value of matrix entries.
              MAXENTRY(A) is maximum of absolute values
              of the entries of A

>> 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 lokalnih spremenljivk, ki smo jih uporabljali pri izvajanju funkcije.

Ime funkcije naj bo enako imenu datoteke, neglede na to, da Matlab upošteva samo ime datoteke.

*Primer 5.3.* Poglejmo še nekaj primerov funkcij. Zapišimo datoteko `maxabs.m` takole

```
% MAXABS    Largest absolute value.
%    For matrices, MAXABS(X) is the largest absolute value in X.
%
%    [Y,I] = MAXABS(X) returns the index of the maximum absolute value in vector I.
%
%    MAXABS(X,Y) returns an array the same size as X and Y with the
%    largest absolute value taken from X or Y.  Either one can be a scalar.
%
function [z,i]=maxabs(x,y)
    if nargin < 2 & nargout < 3
[z,i]=max(abs(x(:)));
    elseif nargin > 0 & nargout < 2
z=max(abs(x),abs(y))
    else
error('wrong number of arguments')
    end;
```

Funkcija vrača dva argumenta, če ji ponudimo en argument. Če pa ji ponudimo dva argumenta, potem vrača samo en argument.

Če so argumenti funkcije nizi, potem Matlab dopušča alternativni način klica takšne funkcije. Funkcijo lahko kličemo tako da povemo njeno ime in naštejemo argumente po vrsti, brez oklepajev in navednic, ločene s presledki.

*Primer 5.4.* Definirajmo funkcijo, ki sprejema tri argumente tipa `char`. Imenujmo jo `trifun.m`

```
function trifun(x,y,z)
% trifun Illustrative function with three string arguments
disp(x), disp(y), disp(z)
```

Funkcija sprejema tri nize in jih izpiše na zaslon. Funkcijo lahko kličemo takole

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

Ali pa kot je običajno takole

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

Ta sproščenost pa nas lahko privede v past. Kaj dobimo, če zapišemo

```
>> sin 1

ans =

   -0.9538
```

Tudi v tem primeru smo dobili neko vrednost, čeprav funkcija `sin` sprejema samo numerični argument. Kaj smo dobili? Zapišimo

```
>> double('1')

ans =

    49

>> sin(49)

ans =

   -0.9538
```

Vidimo, da smo dobili enak rezultat, kot da bi `ascii` vrednost argumanta vstavili v funkcijo `sin`. Torej `sin 1` je isto kot `sin(double('1'))` Poglej mo še naslednji primer

```
>> 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 v tem primeru?

## 5.3 Funkcija kot parameter

| funkcije | |
|---|---|
| `eval` | *ovrednotenje niza* |
| `feval` | *klic funkcije po imenu* |
| `inline` | *inline funkcijski objekt* |
| `vectorize` | *vektorizacija* |
| `fcnchk` | *preveri funkcijski argument* |

Pri uporabi Matlaba kmalu naletimo na problem, kako prenesti funkcijo kot parameter v drugo funkcijo. Matlab ima za to na razpolago več načinov. Pri vsem tem, se odraža nekoliko tudi zagodovina razvoja Matlaba.

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

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

   Nato se spremenljivka `f` prenese v drugo funkcijo kot vhodni parameter. Vrednost izračunamo tako, da najprej inicializiramo spremenljivko `x` in nato pokličemo `eval(f)`. Funkcija `eval` izvrši niz `f`, kot da bi ga zapisali v ukazno vrstico Matlaba. Neugodno je v tem primeru je, da moramo vedeti, kako se imenuje neodvisna spremenljivka v nizu `f`.

   *Primer 5.5.* Poglejmo primer

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

   ans =

      -0.7980

   >>
   ```

2. Funkcijo lahko prenesemo tako, da prenesemo niz z imenom funkcije, ki je definirana v Matlabu ali po jo definiramo sami neki datoteki tipa `.m`. Podobno kot prej prenesemo v drugo funkcijo spremenljivko tipa `char`, ki nosi ime funkcije. Funkcijske vrednosti izračunamo s pomočjo funkcije `feval`. Naj spregovori primer:

   *Primer 5.6.* Vzemimo Matlabovo funkcijo `sin`.

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

y =

    0.1411

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

y =

    0.9093

>>
```

3. V novejših verzijah Matlaba lahko definiramo funkcije s pomočjo ukaza `inline`.

   *Primer 5.7.* Definicija funkcije s pomočjo `inline`

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

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
```

   S stavkom `inline` zgradimo inline funkcijski objekt, spremenljivko, ki se kot argument prenaša v drugo funkcijo, podobno kot vsaka druga spremenljivka. Sintaksa klica funkcije običajna `f(x)`.

Funkcija `vektorize` sprejema argument tipa `char` in vrača argument enakega tipa. Računske operacije aritmetičnega izraza, zapisanega v nizu pretvori v operacije nad polji. Operacijo `*` nadomesti z `.*` in tako naprej.

*Primer 5.8.* Poglejmo primer.

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

f =

x*sin(x)+x^2/y
>> vectorize(f)

ans =

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

Podobno pretvori operacije v inline funkcijskem objektu.

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

Ko definiramo funkcijo, ki sprejema kot argumente druge funkcije bi radi, da bi lahko prenesli funkcije, ki so definirane na katerikoli od opisanih načinov. Če je funkcijski predpis zapleten, da ga ne moremo zapisati v eni sami vrstici, potem moramo funkcijo nujno zapisati v datoteko tipa `.m`. Če pa je funkcijski predpis preprost lahko definiramo inline funkcijski objekt. Včasih bi radi prenesli niz z funkcijskim predpisom neposredno, ne da bi morali prej definirati inline funkcijski objekt. To je možno s pomočjo ukaza `fcnchk`, ki preveri, kako je funkcija definirana in jo pripravi v obliko, da jo lahko sprejme funkcija `feval`, kot argument. Funkcijski argument funkcije `feval` je lahko, kot smo že omenili, ime funkcije definirane v Matlabu ali na datoteki tipa `.m`, ali inline funkcijski objekt.

*Primer 5.9.* Definirajmo inline funkcijski objekt in poglejmo, kako s pomočjo `feval` izračunamo funkcijske vrednosti.

```
>> f=inline('x*sin(x)+x^2/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 ukazu `fcnchk`. Rekli smo, da spremeni funkcijski argument v obliko, ki jo sprejme `feval`. Če je argument tipa `char`, ki predstavlja izraz z definiranim funkcijskim pravilom, ga spremeni v inline funkcijski objekt, če pa argument tipa `char` nosi samo ime funkcije, ga pusti nedotaknjenega in nazadnje, če je argument inline funkcijski objekt, takega tudi vrne.

*Primer 5.10.* Preverimo delovanje ukaza `fcnchk`.

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

f =

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

>> f=fcnchk('sin')

f =

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
```

Natančno proučimo naslednji primer.

*Primer 5.11.* Vzeli bomo končno razliko, kot približek za odvod funkcije. Zapišimo v datoteko `fd.m` naslednje

```
% FD Finite difference approximation to derivative
% FD(F,X,H) is finite difference approximation to the
% derivative of function F at X with difference parameter H.
% H defaults to 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);
```

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

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

- V drugem primeru definiramo niz `f` s funkcijskim predpisom.

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

  f =

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

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

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

  g =

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

Sedaj pa prepričajmo, da lahko v funkcijo `fd` vstavimo katerokoli definicijo naše funkcije in nam bo vrnila vedno pravilen rezultat.

```
>> 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('x.*sin(x)+x.^2',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
```