Pozdravljen, Matlab

Osnove Matlaba za študente numerične matematike

Jan Grošelj

jan.groselj@fmf.uni-lj.si

Povzetek. Gradivo je namenjeno spoznavanju osnovnih funkcionalnosti računalniškega programa Matlab (in GNU Octave). Po kratki predstavitvi programa si ogledamo osnovne ukaze s števili in matrikami. Nadaljujemo s pregledom nekaterih vgrajenih funkcij in predstavimo, kako kot uporabniki implementiramo svoje. Zadnji del je posvečen funkcijam za ustvarjanje slik.

1 Kaj je Matlab?

Matlab je računalniški program za numerično računanje, ki ga razvija podjetje MathWorks. Zametki programa segajo v pozna sedemdeseta leta prejšnjega stoletja, tržni produkt pa je postal v sredini osemdesetih let.

Ime Matlab je skovanka, ki izhaja iz besedne zveze 'matrični laboratorij'. Poimenovanje razkriva, da je program izvirno in v prvi vrsti namenjen računanju z matrikami. Podprte metode iz numerične linearne algebre pretežno temeljijo na postopkih, implementiranih v programski knjižnici LAPACK.

Poleg osnovnih orodij za delo z matrikami dandanes Matlab nudi široko podporo za risanje funkcij in predstavitev podatkov, implementacijo algoritmov in oblikovanje uporabniških vmesnikov. Jedro programa je razširljivo s številnimi dodatnimi paketi, namenjenimi simboličnemu in vzporednemu računanju, strojnemu učenju, reševanju optimizacijskih problemov, modeliranju s krivuljami, analizi finančnih podatkov, simulaciji kontrolnih sistemov, ... Krog uporabnikov programa je izredno širok, med drugim je odlično orodje za učenje in preizkušanje numeričnih metod.

Ker je licenca za Matlab plačljiva, lahko nadomestilo zanj poiščemo med številnimi odprtokodnimi alternativami. Omenimo le program GNU Octave, ki velja za najbolj zvestega posnemovalca Matlabovih funkcionalnosti. Ta lahko v celoti nadomesti Matlab pri izvajanju ukazov, zaobjetih v tem besedilu.

2 Osnovni ukazi s števili

2.1 Računanje

V Matlabu ukaze izvajamo v ukaznem oknu (ang. Command Window). Ukazno okno nam lahko služi kot kalkulator z računskimi operacijami seštevanja (`+`), odštevanja (`-`), množenja (`*`), deljenja (`/`) in potenciranja (`^`).

```
2+3
5-4
```

```
3*2
5/4
3^2
```

Kot decimalno ločilo pri izpisu in vnosu števil se v Matlabu uporablja pika. Pri vnosu števil si lahko pomagamo z eksponentno notacijo.

```
1.234
1e5
1e-3
```

Na voljo so tudi elementarne funkcije za korenjenje (`sqrt`), eksponenciranje (`exp`), logaritmiranje (`log`, `log2`, `log10`) in trigonometrijo (`sin`, `cos`, `tan`, `cot`). Ne manjkajo niti specialne funkcije, na primer gama (`gamma`) in beta (`beta`).

```
sqrt(3)
exp(2)
log(exp(2))
log10(1e-5)
sin(0)
gamma(0.5)
beta(2,3)-beta(3,2)
```

Za izpisovanje števil se privzeto uporablja kratki zapis (`format short`), pri katerem so za decimalno piko navedene štiri števke. Kadar jih želimo videti več (na primer pri izpisu približka za π, ki je definiran v konstanti `pi`), preklopimo na dolgi zapis (`format long`) s petnajstimi števkami za decimalno piko.

```
format short
pi
format long
pi
```

Računamo lahko tudi v kompleksni aritmetiki. Imaginarno enoto predstavlja znak `1i`.

```
(2+1i)+(3-2i)
(1+1i)*(5+2i)
```

Pri računanju lahko naletimo na neskončno (`Inf`) in ne-število (`NaN`).

```
1/0
0/0
1/Inf
Inf-Inf
1+NaN
```

2.2 Primerjanje

Števila lahko med seboj primerjamo z operatorji manjše (`<`), večje (`>`), manjše ali enako (`<=`) in večje ali enako (`>=`) ter je enako (`==`) in ni enako (`~=`). Rezultat ukaza je logična vrednost res (`1`) ali ni res (`0`). Logično vrednost negiramo z `~`, za logični in ter ali pa uporabimo operatorja `&&` ter `||`.

```
3 < 2
4 >= 4
1 == 1
2 ~= 2
~1
(0 || 1) && ~0
```

2.3 Spremenljivke

Izračunane vrednosti lahko shranjujemo v spremenljivke. Spremenljivk običajno ne deklariramo, saj se njihov tip določi avtomatsko. Ko spremenljivki predpišemo število, se privzeto uporablja tip `double`. Če ga želimo spremeniti v celo število, uporabimo na primer ukaz `int64`, če želimo logično vrednost, pa ukaz `logical`. Podatke o spremenljivkah, ki smo jih definirali, lahko pridobimo z ukazom `whos`. Spremljamo jih lahko tudi v delovnem oknu (ang. Workspace), v katerem vedno najdemo tudi spremenljivko `ans`, ki predstavlja zadnjo nedefinirano vrednost, izračunano v ukaznem oknu.

```
c = 2
n = int64(2)
d = 2.34
l = logical(0)
s = 'niz'
C = c + d
N = n + d
```

Poseben tip spremenljivk so funkcije, ki jih lahko podamo v anonimni (brezimenski) obliki. V ta namen uporabimo znak `@`, za katerim v okroglih oklepajih (`(`, `)`) definiramo enega ali več argumentov funkcije. Argumentom sledi predpis funkcije. Vrednost funkcije, ki smo jo shranili v spremenljivko, izračunamo tako, da poleg imena spremenljivke v okroglih oklepajih navedemo vrednosti argumentov.

```
f = @(x)sqrt(x)*exp(-x)
f(2)
g = @(x,y)sin(x)*cos(y)
g(2,3)
```

3 Osnovni ukazi z matrikami

3.1 Vnos matrik

Matrike v Matlab vnašamo s pomočjo oglatih oklepajev (`[`, `]`). Koeficiente matrike vnašamo po vrsticah; pri tem podpičje (`;`) napoveduje novo vrstico, koeficienti v isti vrstici pa so ločeni s presledkom ali (bolj eksplicitno) z vejico (`,`). Število koeficientov mora biti v vseh vrsticah enako, sicer program vrne napako. Posebni primeri matrik so prazna matrika `[]` (matrika velikosti 0×0), stolpec (matrika z enim stolpcem) in vrstica (matrika z eno vrstico).

```
A = [2 7 9; 3 1 5; 8 1 2; 3 6 1]
B = [8, 5, 7; 0, 2, 4; 2, 4, 1]
E = []
a = [2; 4; 3]
b = [3 2 1 7]
```

Matrike lahko gradimo tudi bločno, to je z združevanjem več manjših matrik, ki predstavljajo bloke končne matrike. Pri tem moramo paziti na skladnost blokov, da je rezultat konstrukcije res matrika.

```
[1 2 3; A]
[A [4; 2; 1; 5]]
[A [1 2 3; B]]
[A A; A A]
```

3.2 Operacije z matrikami

Podobno kot pri številih so tudi za matrike na voljo računske operacije seštevanja (`+`), odštevanja (`-`), množenja (`*`) in potenciranja (`^`). Pri izvajanju teh operacij je treba paziti na ustrezne velikosti matrik, s katerimi računamo: seštevamo in odštevamo le matrike enakih velikosti, pri množenju se mora število stolpcev prve matrike ujemati s številom vrstic druge matrike, potenciramo pa le kvadratne matrike. Izjeme so prištevanje in odštevanje števila ter množenje s številom.

```
A+A
B-B
A*B
A*a
b*A
b*A*a
B^2
A+3
3*B
```

Matriko transponiramo z ukazom `transpose` ali (priročneje) z operatorjem `'`. Če imamo opravka z matrikami s kompleksnimi koeficienti, za transponiranje namesto `'` uporabimo

`.'`, saj `'` ustreza konjugiranemu transponiranju.

```
A'
A*A'
(A+1i).'
(A+1i)'
```

Kadar matriko obravnavamo kot tabelo in želimo množiti, deliti ali potencirati istoležne koeficiente, operacijo predznačimo s piko (`.`). Na matrikah lahko uporabimo tudi elementarno funkcijo; pri tem se funkcija aplicira na vsak koeficient matrike posebej. Enako velja za primerjalne operatorje.

```
A.*A
A./A
A.^2
exp(A)
B == B'
A > 2
```

3.3 Naslavljanje koeficientov matrike

Do koeficientov matrike dostopamo z okroglimi oklepaji (`(`, `)`), znotraj katerih najprej podamo indeks vrstice i, nato pa še indeks stolpca j. Indeksa ločimo z vejico (`,`). Par `(i,j)` je naslov koeficienta v matriki. Koeficienti v Matlabu so vedno indeksirani od 1 dalje, to je $i \geq 1$ in $j \geq 1$. Vrednost koeficienta matrike lahko spremenimo, če njegovemu naslovu nastavimo novo vrednost.

```
A(2,3)
A(2,3) = 0
```

Naslavljamo ali spreminjamo lahko tudi več koeficientov matrike hkrati. Pri tem nam je pogosto v pomoč dvopičje (`:`), s katerim nadomestimo enega izmed indeksov, če želimo nasloviti vse koeficiente v stolpcu ali vrstici. Namesto indeksa lahko uporabimo tudi simbol `end`, ki predstavlja zadnji indeks v vrsticah ali stolpcih.

```
A(:,1)
A(:,2) = b'
A(2,:)
A(end,:)
```

Natančneje lahko koeficiente matrik naslavljamo s pomočjo seznamov. Če na prvo oziroma drugo mesto v naslovu postavimo seznam, s tem naslavljamo vse vrstice oziroma stolpce, katerih indeksi se pojavijo v seznamu. Tudi pri navajanju seznamov si lahko pomagamo z dvopičjem: `i:j` je zaporedje indeksov $i, i+1, \ldots, j$, `i:k:j` pa zaporedje indeksov $i, i+k, \ldots, j$.

```
A([2 4],[3 1])
A(1:3,1:3) = B
A(1:2:end,:)
```

Naslavljanje koeficientov matrike z enim indeksom navadno uporabljamo le v primerih, kadar je matrika stolpec ali vrstica. V splošnem enojni indeks predstavlja koeficient na mestu, ki ga dobimo, če koeficiente v matriki številčimo glede na vrstni red $(1,1),(2,1),(3,1),\ldots,(1,2),(2,2),\ldots$. Če uporabimo enojno indeksiranje in za indeks vstavimo `:`, je rezultat vektorizirana matrika: to je vektor, dobljen z zlaganjem stolpcev matrike enega pod drugim.

```
a(2)
b(3)
A(7)
A(:)
```

Elemente seznamov je včasih priročno naslavljati z logičnimi vrednostmi. Če na primer želimo podseznam z elementi, ki zadoščajo določenemu kriteriju, najprej pripravimo seznam logičnih vrednosti, ki je enake dolžine kot prvotni seznam in za vsak element posebej pove, ali izpolnjuje kriterij (`1`) ali ne (`0`). Nato ta seznam logičnih vrednosti uporabimo namesto indeksov.

```
a>2
a(a>2)
```

3.4 Posplošitve matrik

Tako kot matrike posplošujejo sezname, večdimenzionalne tabele posplošujejo matrike. Tridimenzionalno tabelo si lahko predstavljamo kot skupek več slojev matrik. Koeficiente naslavljamo na enak način kot pri matrikah, le da uporabljamo tri (ali več) indeksov.

```
A3 = A;
A3(:,:,2) = A.^2;
A3(2,2,2)
```

Namesto (večdimenzionalnih) seznamov lahko uporabljamo objekte, ki jim pravimo celični seznami. Podajamo jih z zavitimi (`{`, `}`) namesto z oglatimi oklepaji. Elemente celičnih seznamov imenujemo celice in jih naslavljamo na enak način kot elemente seznamov, le da namesto okroglih uporabljamo zavite oklepaje. Celični seznami se od običajnih razlikujejo predvsem po tem, da lahko celica hrani objekte poljubnega tipa. Tako lahko na primer v eno izmed celic celičnega seznama shranimo matriko, v drugega število, v tretjega niz, v četrtega funkcijo in podobno.

```
C = {A, 3; 'niz', B}
C{2,2} = @(x)x.^2
C{2,2}(C{1,1})
```

4 Vgrajene funkcije

4.1 Osnovna oblika

Matlab se ponaša s številnimi vgrajenimi funkcijami. Funkcijo z imenom `funkcija` pokličemo z ukazom `funkcija(v1,v2,...)`, kjer so `v1`, `v2`, ... vhodni podatki. Nekatere funkcije imajo le en vhodni podatek, druge več. Funkcije imajo lahko tudi spremenljivo število vhodnih podatkov, kar pomeni, da jih v nekaterih primerih kličemo z več, v drugih pa z manj vhodnimi podatki. Rezultat funkcije je posredovan preko enega ali več izhodnih podatkov. Če ima funkcija en izhodni podatek ali nas zanima le prvi izhodni podatek in želimo tega shraniti v spremenljivko `i1`, uporabimo klic `i1 = funkcija(v1,v2,...)`. Sicer izhodne podatke `i1`, `i2`, ... naštejemo v seznamu in jih pridobimo z ukazom `[i1,i2,...] = funkcija(v1,v2,...)`. V kolikor nas zanima le specifičen izhodni podatek, na primer drugi, uporabimo klic `[~,i2] = funkcija(v1,v2,...)`. Z ukazom `help funkcija` v ukaznem oknu lahko dostopamo do podrobne dokumentacije funkcije, ki vključuje opis njenega delovanja ter pomen njenih vhodnih in izhodnih podatkov.

4.2 Funkcije za analizo matrik

Za ugotavljanje velikosti matrike je na voljo funkcija `size`, ki vrne dva izhodna podatka: prvi predstavlja število vrstic, drugi pa število stolpcev. Če želimo pridobiti le število vrstic ali stolpcev, lahko poleg matrike kot vhodni podatek navedemo dodaten parameter: s parametrom `1` dobimo število vrstic, s parametrom `2` pa število stolpcev. Pri obravnavi dolžine seznama (oziroma vrstice ali stolpca) je priročnejša funkcija `length`, ki ustreza večjemu od izhodnih podatkov funkcije `size`.

```
[m,n] = size(A)
m = size(A,1)
n = size(A,2)
length(a)
length(b)
```

Pri iskanju najmanjšega in največjega elementa v seznamu si lahko pomagamo s funkcijama `min` in `max`. Funkciji imata dva izhodna podatka: prvi predstavlja najmanjši oziroma največji element, drugi pa indeks tega elementa v seznamu. Če eno ali drugo funkcijo uporabimo na matriki, sta izhodna podatka vrstici, v katerih j-ti element predstavlja najmanjši oziroma največji element ter pripadajoči indeks v j-tem stolpcu matrike. Na analogen način delujeta tudi funkciji `sum` in `prod` za seštevanje in množenje elementov v seznamu.

```
[m,i] = min(a)
max(b)
[M,I] = min(A)
min(min(A))
sum(a)
prod(a)
```

Determinanto kvadratne matrike lahko izračunamo s funkcijo `det`. Glavno diagonalo matrike dobimo s funkcijo `diag`. Priročni sta tudi funkciji `tril` in `triu`, ki vrneta matriki, dobljeni iz vhodne matrike tako, da elemente nad oziroma pod glavno diagonalo nadomestita z ničlami.

```
det(B)
diag(B)
diag(A)
tril(B)
triu(A)
```

4.3 Funkcije za generiranje matrik

Matriko velikosti $m \times n$, ki ima vse koeficiente enake 0, lahko konstruiramo z ukazom `zeros(m,n)`. Podobno z ukazom `ones(m,n)` dobimo matriko, v kateri so vsi koeficienti enaki 1, z ukazom `rand(m,n)` pa matriko z (enakomerno porazdeljenimi psevdo) naključnimi koeficienti. Kadar želimo kvadratno matriko, je dovolj funkcijam podati le en vhodni podatek, ki določa število vrstic in stolpcev.

```
zeros(4,2)
ones(2,3)
rand(3,2)
rand(4)
```

Identiteto velikosti n dobimo z ukazom `eye(n)`, diagonalno matriko z različnimi elementi na diagonali pa lahko definiramo s pomočjo funkcije `diag`, ki ji podamo seznam koeficientov na diagonali. Pri tej funkciji lahko kot dodatni vhodni podatek navedemo še celo število k: rezultat je kvadratna matrika, v kateri se elementi seznama nahajajo na k-ti diagonali.

```
eye(5)
diag(1:4)
diag(1:4,1)
diag(1:4,-2)
```

5 Uporabniške datoteke

5.1 Skriptne datoteke

Zaporedja ukazov lahko v Matlabu shranjujemo v obliki skriptnih datotek (ang. Script) s končnico `.m`. Prazno datoteko ustvarimo v glavnem oknu programa z bližnjico CTRL+N. Odpre se urejevalnik (ang. Editor), v katerega vnašamo ukaze vrstico za vrstico na enak način kot v ukaznem oknu. Razlika je, da se ukazi ne izvedejo takoj, temveč šele, ko skriptno datoteko zaženemo. Pred tem je treba datoteko shraniti, kar napravimo z bližnjico CTRL+S. Nato datoteko zaženemo z bližnjico F5. Ukazi se izvedejo v enakem vrstnem redu, kot so zapisani v datoteki, definirane spremenljivke pa se prikažejo v delovnem oknu, kot če bi jih definirali v ukaznem oknu. V ukaznem oknu lahko do teh spremenljivk dostopamo in jih uporabljamo v nadaljnjih izračunih.

Skriptno datoteko lahko zaženemo tudi iz ukaznega okna tako, da jo preprosto pokličemo po njenem imenu. Pri tem mora biti trenutna mapa (ang. Current Folder) v programu nastavljena na mapo, v kateri je shranjena datoteka.

Naloga 1. Pripravite skriptno datoteko, ki s pomočjo vgrajene funkcije `isprime` poišče in izpiše vsa praštevila manjša od 100.

Rešitev. Zgeneriramo seznam vse števil med 1 in 99 ter s funkcijo `isprime` testiramo, katera so praštevila. Nato sestavimo in izpišemo seznam vseh praštevil. Pravilnost rezultata lahko preverimo z vgrajeno funkcijo `primes`.

```
% skriptna datoteka, ki izpiše vsa praštevila manjša od 100
stevila = 1:99;
jePrastevilo = isprime(stevila); % help isprime
prastevila = stevila(jePrastevilo);
disp(prastevila);
```

V datoteki lahko vnašamo komentarje, ki jih napovemo z znakom `%`. Pri pripravi skriptnih datotek na koncu vsakega ukaza navadno uporabimo podpičje (`;`), s čimer preprečimo izpis vrednosti pri zaganjanju datoteke. Če želimo vrednost ob zagonu izpisati v ukaznem oknu, pa lahko posežemo po funkciji `disp` ali `fprintf`. S pomočjo skriptnih datotek lahko sestavljamo tudi ukazne programe, ki uporabniku omogočajo dinamičen vnos podatkov.

Naloga 2. S pomočjo skriptne datoteke sestavite ukazni program, ki prešteje vsa praštevila manjša od števila, ki ga vnese uporabnik.

Rešitev. Vnos števila uporabniku omogočimo s pomočjo funkcije `input`. Nato s funkcijo `sum` preštejemo, koliko je praštevil manjših od podanega števila.

```
% skriptna datoteka, ki prešteje vsa praštevila manjša od
% števila, ki ga vnese uporabnik
n = input('Vnesite število: '); % vnos se shrani v 'n'
stevila = 1:n-1;
jePrastevilo = isprime(stevila);
p = sum(jePrastevilo);
fprintf('Število praštevil manjših od %d je %d.\n',n,p);
```

5.2 Funkcijske datoteke

Funkcijsko datoteko (ang. Function) ustvarimo na enak način kot skriptno, le da v njej definiramo funkcijo v standardni obliki `[i1,i2,...] = funkcija(v1,v2,...)`, kjer je `funkcija` ime funkcije, `v1`, `v2`, ... in `i1`, `i2`, ... pa so vhodni in izhodni podatki. Če je izhodni podatek le en, oglate oklepaje opuščamo. V telesu funkcije moramo definirati vse izhodne podatke. Funkcija ob klicu v odgovor posreduje vrednosti izhodnih podatkov, ki jih ima ob izteku vseh ukazov. Eksplicitno vračanje rezultatov ni potrebno.

Naloga 3. Težišče matrike $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ je točka (x, y), ki jo določata koordinati

$$x = \frac{1}{M}\sum_{i=1}^{m}\sum_{j=1}^{n} j\, \boldsymbol{A}(i,j), \quad y = \frac{1}{M}\sum_{i=1}^{m}\sum_{j=1}^{n} i\, \boldsymbol{A}(i,j); \quad M = \sum_{i=1}^{m}\sum_{j=1}^{n} \boldsymbol{A}(i,j).$$

Sestavite funkcijo, ki izračuna težišče podane matrike.

Rešitev. Pripravimo funkcijo `tezisce`, ki ima en vhodni podatek, matriko $\boldsymbol{A}$, in dva izhodna podatka, koordinati x in y. V telesu funkcije najprej izračunamo vrednost M, nato pa z nekaj spretnosti še koordinati x in y.

```
function [x,y] = tezisce(A)
% funkcija
%   [x,y] = tezisce(A)
% izračuna težišče (x,y) matrike A

M = sum(sum(A));
[m,n] = size(A);
x = (1:n)*sum(A,1)'/M;
y = (1:m)*sum(A,2)/M;

end
```

Implementirano funkcijo shranimo v datoteko z enakim imenom, kot ga ima funkcija. Pokličemo jo iz ukaznega okna na enak način kot vgrajene funkcije. Če funkcijo dokumentiramo, se z ukazom `help` prikaže tudi njen opis.

Naloga 4. Magični kvadrat velikosti $n \in \mathbb{N}$ je kvadratna matrika velikosti $n \times n$ s koeficienti $1, 2, \ldots, n^2$, za katero velja, da je vsota koeficientov v vseh stolpcih in vrsticah ter na obeh diagonalah enaka. Primer magičnega kvadrata izbrane velikosti dobimo z vgrajeno funkcijo `magic`. Izračunajte težišča magičnih kvadratov za $n = 3, 4, 5$.

Rešitev. Najprej preverimo, da `help tezisce` izpiše opis funkcije, ki smo ga navedli v njeni implementaciji. Za težišče magičnega kvadrata velikosti n v splošnem velja, da ustreza središču mreže $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$, v točke katere postavimo mase, ki ustrezajo koeficientom matrike na teh naslovih. Klici implementirane funkcije za velikosti $n = 3, 4, 5$ to potrdijo.

```
help tezisce
[x3,y3] = tezisce(magic(3)) % (2,2)
[x4,y4] = tezisce(magic(4)) % (2.5,2.5)
[x5,y5] = tezisce(magic(5)) % (3,3)
```

Vhodni podatki se v funkcijo posredujejo po vrednosti. To pomeni, da funkcija v primeru, da spreminjamo vhodne podatke, operira na njihovih kopijah in spremembe nimajo vpliva na stanje spremenljivk izven funkcije.

6 Programiranje

6.1 Pogojni stavki

Osnovni pogojni stavek v Matlabu zapišemo z besedo `if` (če), ki ji sledi pogoj. Če je pogoj izpolnjen, se izvedejo ukazi v telesu pogojnega stavka. Stavek zaključuje beseda `end`. Stavek lahko razširimo z besedo `else` (sicer), ki se izvede, če ni izpolnjen pogoj, ki sledi besedi `if`. Kot je običajno pri programskih jezikih, je na voljo tudi beseda `elseif` (sicer če), s pomočjo katere lahko izrazimo zaporedje pogojev.

Naloga 5. Sestavite funkcijo, ki ugotovi, ali je podana matrika magični kvadrat.

Rešitev. Pripravimo funkcijo `jemagicnikvadrat`, ki sprejme matriko in zanjo po vrsti preveri vse pogoje, ki morajo biti izpolnjeni, da ustreza definiciji magičnega kvadrata: biti mora kvadratna matrika z elementi $1, 2, \ldots, n^2$, vrednosti vsot koeficientov v vseh stolpcih in vrsticah ter v obeh diagonalah pa morajo biti enake. Pri tem si pomagamo z vgrajenimi funkcijami `isequal`, `unique`, `sum` in `diag`. Funkcija vrne logično vrednost res (`true`) ali ni res (`false`).

```
function je = jemagicnikvadrat(A)
% funkcija
%   je = jemagicnikvadrat(A)
% preveri, ali je matrika A magični kvadrat

[m,n] = size(A);
if m ~= n
    je = false;
elseif ~isequal(unique(A(:))',1:n^2)
    je = false;
else
    s = sum(A,1);
    if any(s ~= s(1))
        je = false;
    else
        if any(sum(A,2) ~= s(1))
            je = false;
        elseif sum(diag(A)) ~= s(1)
            je = false;
        elseif sum(diag(A(:,n:-1:1))) ~= s(1)
            je = false;
        else
            je = true;
        end
    end
end

end
```

Funkcijo testiramo z naslednjimi ukazi.

```
jemagicnikvadrat(magic(7))   % true
jemagicnikvadrat(magic(2))   % false !?
jemagicnikvadrat(ones(7))    % false
```

6.2 Zanke

Za pregledovanje sta v Matlabu na voljo zanki `for` in `while`. Pri zanki `for` definiramo indeks `i` in z izrazom `i = s` povemo, da `i` preteče vse elemente seznama `s`. Če uporabljamo zanko `while` navedemo pogoj, ki mora biti izpolnjen, da se ta nadaljuje. Telesi obeh zank zaključimo z besedo `end`.

Naloga 6. Z zanko `for` poiščite največje praštevilo, ki je manjše od danega števila.

Rešitev. S `for` zanko se lotimo pregledovanja števil manjših od vnešenega števila n v vrstnem redu od $n-1$ proti 2. Ko s funkcijo `isprime` prepoznamo praštevilo, prekinemo izvajanje zanke z ukazom `break`.

```
n = input('Vnesite število: ');
for p = n-1:-1:2
    if isprime(p)
        break
    end
end
fprintf('Največje praštevilo manjše od %d je %d.\n',n,p);
```

Naloga 7. Z zanko `while` poiščite najmanjše praštevilo, ki je večje od danega števila.

Rešitev. Za dano število n pregledovanje začnemo z $n+1$ in tega v `while` zanki v vsakem koraku povečamo za ena, dokler ne naletimo na praštevilo.

```
n = input('Vnesite število: ');
p = n+1;
while ~isprime(p)
    p = p+1;
end
fprintf('Najmanjše praštevilo večje od %d je %d.\n',n,p);
```

6.3 Preverjanje podatkov

Pri implementaciji funkcije imamo na voljo nekaj spremenljivk in vgrajenih funkcij, s katerimi lahko nadziramo uporabnikove klice. Spremenljivki `nargin` in `nargout` povesta, s kakšnim številom vhodnih in izhodnih podatkov je bil klic opravljen, s funkcijama `narginchk` in `nargoutchk` pa lahko nadziramo število obveznih vhodnih in izhodnih podatkov. Pri implementaciji funkcije včasih prideta prav spremenljivki `varargin` in `varargout`, ki označujeta spremenljivo število vhodnih in izhodnih podatkov ter predstavljata celična seznama, ki ju funkcija sprejme oziroma vrne. V primeru nepravilnih podatkov lahko posežemo tudi po funkciji `error`, ki prekine izvedbo funkcije z napako. Napake, ki jih vrnejo druge funkcije, pa lahko v naši funkciji prestrežemo z blokom, ki ga definirata besedi `try` in `catch`.

Naloga 8. Sestavite funkcijo, ki obvezno sprejme dve matriki in poljubno število neobveznih podatkov, vendar mora biti skupno število vseh vhodnih podatkov sodo. Funkcija naj vrne vsoto matrik ter vsak drugi neobvezni vhodni podatek, začenši s prvim. Klic funkcije mora biti obvezno opravljen tako, da je število izhodnih podatkov za polovico manjše od števila izhodnih podatkov. Če podanih matrik ni mogoče sešteti, naj funkcija prestreže napako in jo izpiše, namesto vsote matrik pa vrne prazno matriko.

Rešitev. Po navodilih naloge pripravimo funkcijo `preveri`.

```
function [C,varargout] = preveri(A,B,varargin)

narginchk(2,Inf);
if rem(nargin,2) ~= 0
    error('Število vhodnih podatkov mora biti sodo');
end
nargoutchk(nargin/2,nargin/2);

try
    C = A+B;
catch me
    C = [];
    disp(me);
end
varargout = varargin(1:2:end);

end
```

Funkcijo testiramo z naslednjimi klici.

```
preveri([])                              % napaka
preveri([1 2],[3; 4])                    % napaka
i1 = preveri([1 2],[3; 4],'v1')          % napaka
i1 = preveri([1 2],[3; 4],'v1','v2')     % napaka
i1 = preveri([1 2],[3; 4])               % [] in izpis napake
[i1,i2] = preveri([1 2],3,'v1','v2')     % [4 5], 'v1'
```

6.4 Analiza izvedbe ukazov

Matlab pri izvajanju kode nastopa kot tolmač in uporabnikove kode ne prevaja v strojni jezik. Posledično trpi hitrost izvedbe, kar pride še posebej do izraza pri uporabi zank. Kadar torej želimo v Matlabu pripraviti časovno učinkovito funkcijo, se zankam izogibamo in poskušamo iterativne postopke implementirati s pomočjo vgrajenih funkcij za delo z matrikami. Pri veliki količini podatkov so lahko razlike v časovni izvedbi občutne, v kar se prepričamo z merjenjem časa izvajanja funkcije s pomočjo ukazov `tic` in `toc`.

Naloga 9. Implementirajte funkcijo, ki izračuna težišče matrike z uporabo dvojne zanke `for`. Primerjajte čas izvajanja funkcije na naključni matriki velikosti $10^4 \times 10^4$ s časom izvajanja funkcije, opisane v rešitvi naloge 3.

Rešitev. Težišče matrike lahko izračunamo z naslednjo funkcijo.

```
function [x,y] = teziscePocasi(A)
% funkcija
%   [x,y] = tezisce(A)
% izračuna težišče (x,y) matrike A (na neučinkovit način)

x = 0;
y = 0;
M = 0;
[m,n] = size(A);
for i = 1:m
    for j = 1:n
        x = x + j*A(i,j);
        y = y + i*A(i,j);
        M = M + A(i,j);
    end
end
x = x/M;
y = y/M;

end
```

Če testiramo na veliki matriki, ugotovimo, da se funkcija `teziscePocasi` izvaja skoraj desetkrat več časa kot funkcija `tezisce`, ki je opisana v rešitvi naloge 3.

```
A = rand(1e4);
tic; teziscePocasi(A); toc;
tic; tezisce(A); toc;
```

Na težave s časovno učinkovitostjo lahko naletimo tudi pri nerodni izbiri vrstnega reda operacij.

Naloga 10. Primerjajte razmerje med časom računanja $(\boldsymbol{x}\boldsymbol{x}^T)\boldsymbol{x}$ in $\boldsymbol{x}(\boldsymbol{x}^T\boldsymbol{x})$ za naključno izbran vektor (stolpec) $\boldsymbol{x}$ velikosti 1000.

Rešitev. V prvem primeru (ki je tudi privzeti način računanja, če ne pišemo oklepajev) najprej izračunamo matriko $\boldsymbol{x}\boldsymbol{x}^T$, ki je velikosti 1000×1000, nato pa to matriko množimo z vektorjem $\boldsymbol{x}$. V drugem primeru izračunamo skalar $\boldsymbol{x}^T\boldsymbol{x}$, s katerim pomnožimo vektor $\boldsymbol{x}$. Pričakovati je torej, da bo drugi način hitrejši; primerjava časov pokaže, da je za več kot stokrat.

```
x = rand(1000,1);
tic; (x*x')*x; T = toc;
tic; x*(x'*x); t = toc;
T/t
```

7 Risanje

7.1 Grafi

Osnovni ukaz za risanje grafa funkcije v Matlabu je `plot`. Kot prvi vhodni podatek mu podamo seznam abscis, ki ga dobimo s fino delitvijo intervala $[a, b]$, na katerem je funkcija definirana. To običajno napravimo z ukazom `a:k:b`, kjer je `k` razmik med točkami na intervalu $[a, b]$, ali z ukazom `linspace(a,b,n)`, kjer je `n` število delilnih točk, s katerimi razdelimo interval. Drugi vhodni podatek za funkcijo `plot` je seznam ordinat, to je vrednosti funkcije v pripadajočem seznamu abscis. Ukaz nariše sliko grafa funkcije oziroma sliko lomljene črte, ki povezuje točke v ravnini, določene s pari istoležnih elementov v seznamih abscis in ordinat.

Naloga 11. Narišite graf funkcije $x \mapsto e^x \sin(100x)$ na intervalu $[0, 1]$. Preizkusite različno fine delitve intervala.

Rešitev. V seznamu `x` definiramo abscise, v seznamu `y` pa ordinate. Ker funkcija močno oscilira, delitev intervala s 100 delilnimi točkami ne zadošča za vizualno gladek graf.

```
x = linspace(0,1,100);
y = exp(x).*sin(100*x);
plot(x,y);
```

Funkcijo `plot` lahko uporabimo tudi pri risanju parametrično podanih krivulj. Če parameter tretiramo kot čas, seznama abscis in ordinat določata pozicije v odvisnosti od časa na abscisni in ordinatni osi.

Naloga 12. Narišite parametrično krivuljo $t \mapsto (e^t \sin(100t), e^t \cos(100t))$ za parametre $t \in [0, 1]$.

Rešitev. Interval, po katerem teče parameter, razdelimo z delilnimi točkami in te zabeležimo v seznam `t`. Nato v seznamih `x` in `y` izračunamo vrednosti x in y koordinat krivulje pri delilnih točkah iz seznama `t`.

```
t = linspace(0,1,1000);
x = exp(t).*sin(100*t);
y = exp(t).*cos(100*t);
plot(x,y);
```

Rišemo lahko tudi krivulje v prostoru. Za ta namen uporabimo funkcijo `plot3`, ki poleg seznama abscis in ordinat sprejme še seznam aplikat.

Naloga 13. Narišite parametrično krivuljo $t \mapsto (e^t \sin(100t), e^t \cos(100t), -t^3)$ za parametre $t \in [0, 1]$.

Rešitev. Kodo iz prejšnje naloge razširimo z definicijo seznama `z`, ki določa vrednosti aplikat. Namesto ukaza `plot` uporabimo ukaz `plot3`.

```
t = linspace(0,1,1000);
x = exp(t).*sin(100*t);
y = exp(t).*cos(100*t);
z = -t.^3;
plot3(x,y,z);
```

Za risanje grafa funkcije dveh spremenljivk je na voljo ukaz `surf`. Uporabimo ga na podoben način kot ukaz `plot3`, le da za podajanje abscis, ordinat in aplikat namesto seznamov uporabimo matrike oziroma tabele vrednosti. Tabeli abscis in ordinat določata mrežo, s katero je razdeljena pravokotna domena funkcije. Navadno ju definiramo s pomočjo ukaza `meshgrid`. Aplikate nato določimo z izračunom vrednosti funkcije nad točkami mreže.

Naloga 14. Narišite graf funkcije $(x, y) \mapsto \cos(x) \sin(y)$ na pravokotniku $[-4, 4] \times [-2, 2]$.

Rešitev. Najprej razdelimo intervala $[-2, 2]$ in $[-1, 1]$ ter delilne točke shranimo v seznama `x` in `y`. Nato z ukazom `meshgrid` definiramo tabeli `X` in `Y`, ki določata abscise in ordinate točk mreže, s katero razdelimo definicijski pravokotnik. Nazadnje izračunamo vrednosti funkcije v točkah mreže in podatke shranimo v tabelo `Z`.

```
x = linspace(-4,4,200);
y = linspace(-2,2,100);
[X,Y] = meshgrid(x,y);
Z = cos(X).*sin(Y);
surf(X,Y,Z);
```

Namesto ukaza `surf` lahko uporabimo ukaz `contour`, ki izriše nivojnice grafa funkcije dveh spremenljivk.

7.2 Oblikovanje slik

Pri risanju grafov so na voljo številne oblikovne možnosti. Funkcijam, ki poskrbijo za risanje, jih posredujemo z dodatnimi vhodnimi podatki. Pri funkciji `plot` lahko z dodatnim nizom opišemo barvo in stil grafa. Niz `'rx--'` na primer pomeni, da graf narišemo v rdeči barvi (`r`), s točkami, označenimi z `x`, in črtkano črto (`--`). O različnih opcijah, ki so na voljo, se lahko poučimo v dokumentaciji funkcije (`help plot`). Spreminjamo lahko tudi debelino črte, tako da navedemo dva dodatna podatka: prvi, `'LineWidth'`, napoveduje lastnost, ki jo spreminjamo, drugi, številski podatek, pa vrednost, na katero želimo nastaviti lastnost črte.

Slike lahko opremimo z naslovom (funkcija `title`) in legendo (funkcija `legend`). Poimenujemo lahko tudi osi grafa (funkciji `xlabel` in `ylabel`). Z ukazom `axis off` odstranimo osi, z ukazom `axis equal` pa izenačimo dolžino enote na obeh oseh. S pomočjo funkcije `text` lahko na sliko dodamo tekstovni opis.

Obstoječo sliko pobrišemo z ukazom `clf`. Okno z novo sliko odpremo z ukazom `figure`. Na isto sliko lahko narišemo tudi več grafov, kar dosežemo tako, da ukaze za izris umestimo med ukaza `hold on` in `hold off`. Sliko razdelimo v tabelo več slik z dodatnimi ukazi `subplot`. Pri tem navedemo tri parametre, od katerih prva dva podajata število vrstic in stolpcev tabele, tretji pa določa indeks slike, narisane z ukazi, ki sledijo funkciji `subplot`.

Naloga 15. Narišite sliko, sestavljeno iz dveh slik. Na levi sliki naj bosta grafa funkcij $x \mapsto e^x \sin(100x)$ in $x \mapsto e^x \cos(100x)$. Graf prve funkcije naj bo narisan z rdečo barvo, črta naj bo črtkana. Graf druge funkcije naj bo narisan s črno barvo, črta naj bo pikčasta. Ob koordinatnih oseh naj bo napis 'abscisa' oziroma 'ordinata'. Slika naj ima legendo, v kateri naj bo prva funkcija naslovljena kot 'x koordinata', druga pa kot 'y koordinata'. Na desni sliki naj bo prikazana parametrična krivulja $t \mapsto (e^t \sin(100t), e^t \cos(100t))$, črta naj bo debeline 2. Slika naj ima naslov 'Krivulja' in naj bo brez osi.

Rešitev. Pri pripravi slike si pomagamo z ukazi, navedenimi zgoraj, ter njihovo dokumentacijo.

```
% leva slika
subplot(1,2,1);
x = linspace(0,1,1000);
hold on
plot(x,exp(x).*sin(100*x),'r--');
plot(x,exp(x).*cos(100*x),'k:');
hold off
xlabel('abscisa');
ylabel('ordinata');
legend('x koordinata','y koordinata');

% desna slika
subplot(1,2,2);
t = linspace(0,1,1000);
plot(exp(t).*sin(100*t),exp(t).*cos(100*t),'LineWidth',2);
title('Krivulja');
axis off
```

7.3 Grafikoni

Za prikaz podatkov lahko uporabimo vgrajene funkcije za risanje grafikonov: s funkcijo `bar` dobimo stolpčni grafikon, s funkcijo `barh` paličnega, s funkcijo `scatter` raztresenega, s funkcijo `plot` pa črtnega. Vsem funkcijam kot vhodna podatka podamo seznam abscis in ordinat. Za tortni diagram uporabimo funkcijo `pie`.

Naloga 16. Spodnja tekstovna datoteka `udelezba.txt` vsebuje podatke o volilni udeležbi na parlamentarnih volitvah v Sloveniji. V prvem stolpcu so leta volitev, v drugem pa volilna udeležba v odstotkih.

```
1992, 86.5
1996, 73.7
2000, 70.1
2004, 60.4
2008, 63.1
2011, 65.6
2014, 51.7
2018, 51.5
```

Uvozite datoteko v Matlab in na podlagi podatkov narišite stolpčni, palični, raztreseni, točkovni in tortni diagram.

Rešitev. Datoteko s podatki uvozimo z ukazom `load`, ki vrne tabelo z osmimi vrsticami in dvema stolpcema. Nato narišemo različne grafikone. Raztresenega in črtnega upodobimo na isti sliki. Pri tortnem diagramu s kosi torte ponazorimo deleže, z oznakami ob kosih pa leta. Funkciji `pie` za prvi vhodni podatek podamo deleže, drugi vhodni podatek pa so leta, pretvorjena v celični seznam s tekstovnimi elementi.

```
P = load('udelezba.txt');

subplot(2,2,1);
bar(P(:,1),P(:,2));

subplot(2,2,2);
barh(P(:,1),P(:,2));

subplot(2,2,3);
hold on
plot(P(:,1),P(:,2));
scatter(P(:,1),P(:,2));
hold off

subplot(2,2,4);
pie(P(:,2),cellstr(num2str(P(:,1))));
```