Uvod v Matlab

Osnovne karakteristike

- MATrix LABoratory, razvije ga Cleve Moler po letu 1970,
- namenjen je numeričnemu računanju,
- osnovna podatkovna struktura je matrika,
- računa v realni in kompleksni aritmetiki,
- ima zelo močno grafično podporo.

Podatkovne strukture

- realna in kompleksna števila,
- logične vrednosti,
- znaki,
- matrike in tabele,
- večdimenzionalne tabele,

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

- celice (objekti, ki lahko vsebujejo tabele različnih dimenzij)

```
M = cell(3,1);
for n = 1:3
  M{n} = magic(n);
end
```

Izvajanje ukazov

- neposredno v ukazni vrstici

```
>>magic(4)
```

- preko programske datoteke,
- preko funkcijske datoteke.

Programska datoteka

```
%komentarji
%ime datoteke je *.m
stavek1;
stavek2
klic_vgrajene_funkcije;
klic_lastne_funkcije;
.
.
.
```

Bodite pozorni na ;, ki prepreči izpis na zaslon.

Funkcijska datoteka

```
function [r1,r2,...,ri]=ime_funkcije(a1,a2,...,aj)
%komentarji za zunanjo pomoc
stavek1;
stavek2;
klici_funkcij;
%lahko tudi rekurzivni klici
%dovoljene pomozne funkcije znotraj osnovne
%pomozne funkcije niso vidne navzven
%ime_funkcije mora biti enako kot ime datoteke
%brez podaljska 'm
```

Funkcijo pokličemo z

```
[r1,r2,...,ri]=ime_funkcije(a1,a2,...,aj)
```

Poskusite tudi

```
>>help ime_funkcije
```

Ko pokličemo funkcijo, Matlab najprej preveri, ali je klicano “ime”:

- spremenljivka,
- pomožna funkcija,
- naša funkcija,
- razredni konstruktor,
- prenaložena metoda,
- funkcija v trenutnem direktoriju,
- funkcija na definirani “poti”.

Primer

Primer funkcije, ki v matriki poišče največji element in njegova indeksa:

```
function [M,imax,jmax]=najvecji(A)
%Poisce najvecji element in njegovo pozicijo v matriki
%[M,imax,jmax]=NAJVECJI(A)
%M je najvecji element
%imax in jmax sta indeksa vrstice in stolpca
%A je matrika

[M1,I]=max(A);
[M,jmax]=max(M1);
imax=I(jmax);
```

Kontrolne strukture

Stavek `if`

```
if logicni_izraz
  stavki
end
```

```
if logicni_izraz
  stavki
else
  stavki
end
```

```
if logicni_izraz1
  stavki
elseif logicni_izraz2
  stavki
else
  stavki
end
```

Primer

```
n = input('Vnesi stevilo: ');
if (rem(n,2)==0)
  fprintf('Stevilo %d je sodo.\n', n)
end
```

Primer

```
n = input('Vnesi stevilo: ');
if (rem(n,2)==0)
  fprintf('Stevilo %d je sodo.\n', n)
else
  fprintf('Stevilo %d je liho.\n', n)
end
```

Stavek `while`

```
while logicni_izraz
  stavki
end
```

Primer

Kaj izračuna naslednji program?

```
x=1;
s=1;
i=1;
clen=1;
while clen>=eps*s
  clen=x*clen/i;
  s=s+clen;
  i=i+1;
end
```

Stavek `for`

```
for indeks=zacetek:korak:konec
  stavki
end
```

Primer

```
for k=5:2:13
  fprintf('Kvadrat stevila %d je %d.\n', k, k^2)
end
```

Poseben primer stavka `for`

```
for indeks=A
  stavki
end
```

Pri tem je A matrika.

Primer

Izračunajmo norme stolpcev matrike A:

```
for stolpec=A
  fprintf('Norma je %6.4f.\n', norm(stolpec,2))
end
```

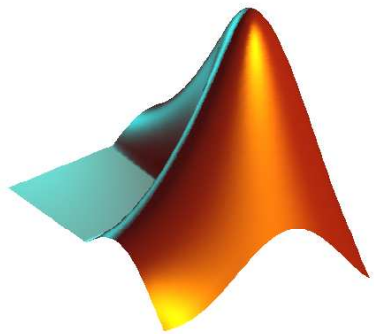
Stavki switch, case in otherwise

```
switch izraz
  case vrednost1
    stavki
  case vrednost2
    stavki
  ...
  otherwise
    stavki
end
```

Osnove grafike in napotki za hitro in ekonomično računanje

Grafika

Zelo dober priročnik za osnove grafike dobite na naslovu
http://www.mathworks.com/access/helpdesk/help/techdoc/ref/f16-6011.html



Slika : Primer slike v Matlabu (logotip).

Nekaj osnovnih ukazov

- `plot`, `plot3d`, `plotyy`, `polar`, `contour...`, risanje dvodimenzionalnih objektov, krivulj v treh dimenzijah in polarnih grafov ter kontur
- `surf`, `mesh`, `contour3d`
- `colormap`, `axis`, `xlabel`, `ylabel`,...nadzor nad grafiko.

Primer

Primer 3D grafa

```
[X,Y,Z] = peaks(30);
surfc(X,Y,Z)
colormap hsv
axis([-3 3 -3 3 -10 5])
```

Splošni napotki

Pri programiranju v Matlabu se moramo zavedati nekaj splošnih navodil:

- Matlab je interpreterski jezik, zato je preudarno programiranje še posebej pomembno.
- Če je le mogoče uporabljajmo Matlabove vgrajene ukaze.
- Uporabljajmo vektorsko/matrični zapis in ustrezne matrične operacije.
- Pri računanju s polji uporabljajmo operator . (pika).

Nekaj poučnih primerov

Oglejmo si nekaj primerov, ki pokažejo, kako velike so lahko razlike v času izvajanja programa za različne implementacije.

- Naj bosta $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. Izračunati želimo vsoto

$$s = \sum_{i=1}^{n} x_i \, y_i.$$

Najpogostejša rešitev je

```
s=0;
for i=1:n
  s=s+x(i)*y(i);
end
```

Izkaže se, da ni najboljša, zato poskusimo z izboljšavo

```
z=x.*y
s=sum(z);
```

Najboljša rešitev pa je kar skalarni produkt

```
s=x*y'
```

- Poiskati želimo največji element v matriki $A \in \mathbb{R}^{m \times n}$. Ponovno izvedba

```
M=-inf;
for i=1:m
  for j=1:n
    if A(i,j)>M
      M=A(i,j);
    end
  end
end
```

ni najboljša. Bistveno krajša in hitrejša je

```
M=max(max(A));
```

- Izračunati želimo prvih n členov Fibonaccijevega zaporedja $f_1 = 1, f_2 = 1, f_{n+1} = f_n + f_{n-1}$. Napisali bomo dve verziji, eno brez inicializacije vektorja `f`, drugo pa z inicializacijo. Torej

```
f(1:2)=1;
for i=3:n
  f(i)=f(i-1)+f(i-2);
end
```

in

```
f=ones(1,n);
for i=3:n
  f(i)=f(i-1)+f(i-2);
end
```

- Čeprav prejšnji primeri večinoma kažejo, da se moramo izogibati nepotrebnim stavkom `for`, to včasih ne drži. Ponovno se vrnimo k Fibonaccijevemu zaporedju. Izračunati želimo n-ti člen. Verzija s stavkom `for` je očitna

```
f1=1;
f2=1;
for i=3:n
  f3=f2+f1;
  f1=f2;
  f2=f3;
end
```

Rekurzivna izvedba pa neuporabna

```
function f=fib(n)
%FIB racuna n-ti clen Fibonaccijevega zaporedja
%rekurzivno (NEUPORABNO!)
%f=FIB(n)
%n je indks clena (zacnemo z 1)
%f je rezultat

if n==1
  f=1;
elseif n==2
  f=1;
else
  f=fib(n-1)+fib(n-2);
end
```

- Koliko elementov v matriki $A \in \mathbb{R}^{m \times n}$ je večjih od c?

```
stevec=0;
for i=1:m
  for j=1:n
    if A(i,j)>c
      stevec=stevec+1;
    end
  end
end
```

Elegantna izvedba v enem stavku je

```
stevec=sum(sum(A>c));
```

Oglejmo si klasično izvedbo LU razcepa z delnim pivotiranjem

```
function [L,U,P]=lu_delno(A)
%LU_DELNO je LU razcep z celnim pivotiranjem
%[L,U,P]=LU_DELNO(A)
%A je vhodna matrika
%L je spodnja trikotna z enkami na diagonali
%U je zgornja trikotna
%P je permutacijska

n=size(A,1);
P=eye(n);
L=zeros(n);
U=zeros(n);
```

```
for i=1:n-1
  [M,maxi]=max(abs(A(i:n,i)));
  maxi=maxi+i-1;%poravimo na globalni indeks
  P([i,maxi],:)=P([maxi,i],:);
  A([i,maxi],:)=A([maxi,i],:);
  for k=i+1:n
    A(k,i)=A(k,i)/A(i,i);
  end
  for v=i+1:n
    for s=i+1:n
      A(v,s)=A(v,s)-A(v,i)*A(i,s)
    end
  end
end
L=tril(A,-1)+eye(n);
U=triu(A);
```

Glavni del funkcije lahko zamenjamo z

```
for i=1:n-1
  [M,maxi]=max(abs(A(i:n,i)));
  maxi=maxi+i-1;%poravimo na globalni indeks
  P([i,maxi],:)=P([maxi,i],:);
  A([i,maxi],:)=A([maxi,i],:);
  A(i+1:n,i)=A(i+1:n,i)/A(i,i);
  A(i+1:n,i+1:n)=A(i+1:n,i+1:n)-A(i+1:n,i)*A(i,i+1:n);
end
L=tril(A,-1)+eye(n);
U=triu(A);
```

Numerične metode z Matlabom

Reševanje sistemov linearnih enačb in iskanje lastnih vrednosti

- Matrični sistem $A\boldsymbol{x} = \boldsymbol{b}$, kjer je $A \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^{m\times 1}$ in $\boldsymbol{x} \in \mathbb{R}^{n\times 1}$, v splošnem rešimo z ukazom \ (backslash):

  `x=A\b`

  Če je matrika nesingularna, je to algebraično (ne pa numerično!!!) ekvivalentno $\boldsymbol{x} = A^{-1}\boldsymbol{b}$.
  Če je $m > n$ in je A polnega ranga, potem \ vrne rešitev po metodi najmanjših kvadratov, torej tisti $\boldsymbol{x}$ za katerega je dosežen minimum

  $$\min_{x\in\mathbb{R}^n} \|A\boldsymbol{x} - \boldsymbol{b}\|_2.$$

- Pomemben ukaz je tudi `lu`, ki izračuna LU razcep z delnim pivotiranjem. Za nesingularno matriko $A \in \mathbb{R}^{n \times n}$ je

$$PA = LU,$$

kjer je P permutacijska matrika, L spodnja trikotna matrika z enkami na diagonali in U zgornja trikotna matrika. Klic je oblike

```
[L,U,P]=lu(A)
```

- Za simetrično pozitivno definitno matriko $A \in \mathbb{R}^{n \times n}$ je ekonomičneje uporabiti razcep Choleskega

$$A = V^\top V,$$

kjer je V zgornja trikotna matrika. Klic je oblike

```
V=chol(A)
```

- Za matriko $A \in \mathbb{R}^{m \times n}$ lahko izračunamo tudi t.i. QR razcep:

$$A = QR,$$

kjer je $Q \in \mathbb{R}^{m \times m}$ ortogonalma matrika ($Q^\top Q = QQ^\top = I$) in $R \in \mathbb{R}^{m \times n}$ "zgornja trikotna" matrika. Klic je oblike

```
[Q,R]=qr(A)
```

- Med najpomembnejšimi je singularni razcep. Za matriko $A \in \mathbb{R}^{m \times n}$ je

$$A = U\Sigma V^\top,$$

kjer sta $U \in \mathbb{R}^{m \times n}$ in $V \in \mathbb{R}^{n \times n}$ ortogonalni matriki ter $\Sigma \in \mathbb{R}^{m \times n}$ "diagonalna matrika" s singularnimi vrednostmi na diagonali. Klic je oblike

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

- Veliko pomembnih pojmov iz linearne algebre temelji na prejšnjih razcepih. Denimo `det` (determinanta), `rank` (rang), `null` (jedro), `orth` (ortogonalna baza), `det` (determinanta), ...
- Za kvadratno matriko $A \in \mathbb{R}^{n \times n}$ lahko poiščemo lastne vrednosti in pripadajoče lastne vektorje. Klic je oblike

```
[V,D]=eig(A)
```

Pri tem so stolpci matrike `V` lastni vektorji, diagonala matrike `D` pa lastne vrednosti.

Reševanje (sistemov) nelinearnih enačb

- Za reševanje ene enačbe uporabimo lahko ukaz `fzero`.
- Za reševanje sistema pa (če imamo "Optimization toolbox") `fsolve`.

Primer

Rešujemo enačbo

$$e^{-x} - x = 0$$

in sistem

$$x^2 + y^2 - 1 = 0$$
$$x^2 - y = 0.$$

- V prvem primeru uporabimo

```
f=inline('exp(-x)-x','x');
xz=fzero(f,0.5);
```

v drugem pa

```
fsis=inline('[X(1)^2+X(2)^2-1;X(1)^2-X(2)]','X');
xz=fsolve(fsis,[1;1]);
```

Seveda moramo poznati dobre začetne približke, da metoda res konvergira k željeni rešitvi.

Iskanje ekstremov skalarnih polj

- Lokalni minimum funkcije ene spremenljivke, torej

$$\min_{x\in[a,b]} f(x),$$

poiščemo z ukazom `fminbnd`. Klic je oblike

```
[xmin,fmin]=fminbnd(f,a,b)
```

Primer

Poiščimo minimum funkcije $f(x) = \sin x - \cos x$ *na intervalu* $[-\pi, \pi]$*:*

```
f=inline('sin(x)-cos(x)','x');
[xmin,fmin]=fminbnd(f,-pi,pi)
```

- Lokalne ekstreme funkcij več spremenljivk lahko poiščemo s funkcijo `fminsearch`. Za funkcijo $F : \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}$ želimo torej poiskati

$$\min_{\boldsymbol{x} \in \mathcal{D}} F(\boldsymbol{x}).$$

  Uporabimo klic oblike

```
[xmin,fmin]=fminsearch(F,x0)
```

Primer

Poiščimo minimum skalarnega polja $f(x,y) = x^2 + y^2 - x\,y$ *na domeni* $[-1,1] \times [-1,1]$*:*

```
F=inline('X(1)^2+X(2)^2-X(1)*X(2)','X');
[Xmin,Fmin]=fminbnd(f,[1;1])
```

- Podobna ukaza sta še `fminunc` in `fmincon`.

Interpolacija in aproksimacija s polinomi

- Tabelo podatkov (x_i, y_i), $i = 0, 1, \ldots m$, interpoliramo s polinomom p_n stopnje $\leq n \leq m$ z ukazom `polyfit`. Klic je oblike

```
p=polyfit(x,y,n)
```

  Če je $n < m$, se uporabi metoda najmanjših kvadratov, torej poišče tak polinom p, za katerega je dosežen minimum

$$\min_{p \in \mathcal{P}_n} \sum_{i=0}^{m} (p(x_i) - y_i)^2,$$

  kjer je $\mathcal{P}_n$ prostor polinomov stopnje ne več kot n.

- Matlab podpira tudi računanje z zlepki. Za tabelo podatkov iz prejšnje točke in predpisane abscise t_j, $j = 1, 2, \ldots, N$, lahko interpolacijski zlepek dobimo z ukazom

```
ft=interp1(x,y,t,metoda)
```

Numerično integriranje

- Integralov

$$\int_a^b f(x)\,dx$$

  pogosto ne moremo analitično izračunati.
- V Matlabu imamo na voljo dve standardni metodi `trapz` (sestavljeno trapezno pravilo) in `quad` (adaptivno Simpsonovo pravilo): Klica sta

```
I=trapz(x,y)
```

```
I=quad(f,a,b)
```

- Podobni ukazi so še `quadgk`, `quadl`, `dblquad`, `triplequad`,...

Reševanje navadnih diferencialnih enačb

- Rešujemo začetni problem

$$y'(x) = f(x), \quad y(x_0) = y_0.$$

Uporabimo lahko eno od vgrajenih Matlabovih funkcij, denimo `ode45`:

```
[x,y]=ode45(f,[x0,b],y0)
```

- Če je diferencialana enačba višjega reda,

$$y^{(k)}(x) = f(x, y(x), y'(x), \ldots, y^{(k-1)}(x)),$$
$$y(x_0) = y_0\, y'(x_0) = y'_0, \ldots, y(x_0)^{(k-1)}(x_0) = y_0^{(k-1)}.$$

jo prevedemo na sistem enačb prvega reda:

Uvedemo nove spremenljivke $y_1 = y, y_2 = y', \ldots, y_k = y^{(k-1)}$.
Zapišemo sistem

$$\begin{aligned} y_1'(x) &= y_2(x) \\ y_2'(x) &= y_3(x) \\ &\vdots \\ y_k'(x) &= f(x, y_1(x), \ldots, y_{k-1}(x)), \end{aligned}$$

z začetnimi pogoji $y_1(x_0) = y_0, y_2(x_0) = y_0', \ldots, y_k(x_0) = y_0^{(k-1)}$.

Primer

Numerično izračunajte višino in hitrost pri navpičnem padanju telesa, na katerega delujeta sili teže in upora. Upoštevajte kvadratični zakon upora. Podani sta začetna višina in začetna hitrost.
Odgovorite na nekaj naslednjih vprašanj:

a) *Kakšna je hitrost telesa po dolgem času?*

b) *Kdaj telo pade na tla?*

c) *Kdaj je telo na polovici začetne višine?*

Namig: Izpeljite diferencialno enačbo drugega reda, jo prepišite v sistem prvega in ga rešite z Matlabovo funkcijo `ode45`*. Odgovore na zgornja vprašanja poičite na podlagi rezultatov. Morda boste morali uporabiti še kakšne metode iz reševanja nelinearnih enačb, . . .*