El?z? gyakorlat - Fel - Következ? gyakorlat

Tartalomjegyzék

- 1 Octave
  - 1.1 Kezdeti lépések
    - 1.1.1 Hozzáférés a programhoz
    - 1.1.2 Számológép
  - 1.2 Adattípusok
    - 1.2.1 A számábrázolások
    - 1.2.2 Mátrixok
    - 1.2.3 Tartományok
    - 1.2.4 typeinfo
  - 1.3 M?veletek mátrixokkal
    - 1.3.1 Transzponált
    - 1.3.2 Összeadás
    - 1.3.3 Szorzás
    - 1.3.4 Tagonként vagy mátrixként
  - 1.4 Változók
  - 1.5 Indexelés
  - 1.6 Vektorizáció
  - 1.7 Függvények
  - 1.8 Feladatok
    - 1.8.1 Mi ez?
    - 1.8.2 LER
    - 1.8.3 Még LER
    - 1.8.4 Nagy mátrix okosan
    - 1.8.5 Függvény mátrixokon
    - 1.8.6 Részmátrix
    - 1.8.7 Részmátrixon függvény
    - 1.8.8 Minden második oszlop
    - 1.8.9 Függvény alkalmazás csak adott elemeken
    - 1.8.10 Segíts magadon
    - 1.8.11 Numerikus deriválás

Octave

Az Octave program alkalmas különböz? matematikai számításokat numerikusan elvégzésére, a nagytestvérének a MatLab-nak az ingyenes (opensource) változata.

Kezdeti lépések

Hozzáférés a programhoz

Ha otthonról dolgozunk, akkor a következ? lehet?ségek legalább egyikével éljünk:

- telepítsünk Octave-ot, ez minden platformra ingyenes
- Putty-al lépjünk be a `leibniz`-re és írjuk be a terminálba, hogy `octave`

A géptermekb?l Linux-ról futtassuk az Octave-ot

Számológép

Az Octave egy fejlettebb számológépként is használható. Írjuk be az `octave` parancssorába az alábbiakat:

```
2+3
```

majd üssünk `Enter`-t. Ennek hatására:

```
> 2+3
ans =  5
> _
```

Próbáljuk ki ezeket is:

```
2-3
2*3
2/3
floor(2/3)
mod(2,3)
2^3
sqrt(2)
log(2)
log(3)
log(8)/log(2)
exp(1)
pi
cos(pi/2)
(180/pi)*acos(0.5)
```

Kilépni így lehet

```
exit
```

Adattípusok

Minden szám alapértelmezésben lebeg?pontos, akkor is, ha véletlenül egész:

```
1000/9
ans = 111.11
```

Viszont megadhatjuk, hogy egészekként értelmezze a számokat:

```
int32(1000)/int32(9)
ans = 111
```

Octave-ban egy szám mindaddig valós, amíg komplexnek nem bizonyul:

```
sqrt(2)
sqrt(-2)
```

A számábrázolások

- `double`: dupla lebeg? pontos, 64 bit (8 byte)
  - valós: 8 byte
  - komplex: 16 byte
- `single`: szimpla lebeg? pontos, 32 bit (4 byte)
  - valós 4 byte
  - komplex: 8 byte
- `int32`: 32 bites kettes komplemens egész (4 byte)
- `int8`: 8 bites kettes komplemens egész: -128..127 (1 byte)
- `uint32`: 32 bites el?jel nélküli egész (4 byte)
- `uint8`: 8 bites el?jel nélküli egész: 0..255 (1 byte)

A méret nagyon is számít:

```
log(single(1.0001))
log(double(1.0001))
int32(100+100)
int8(100+100)
```

Mátrixok

Az `octave`-ban **minden szám egy mátrix**

- számok: `1x1`
- vektorok:
  - sorvektor: `1xn`
  - oszlopvektor: `nx1`
- matrix: `nxm`

Ennek alapos oka van, amit majd kés?bb fogunk megérteni és ami a MatLab leglényegéhez vezet bennünket, ezt vette át az `octave` is. B?vebben itt.

Sorvektor:

```
[1, 2, 3, 4]
[1 2 3 4]
```

Oszlopvektor:

```
[1;2;3;4]
```

Ez **nem** oszlopvektor:

```
[[1], [2], [3], [4]]
```

Mátrix:

```
[1 2; 3 4]
[1, 2; 3, 4]
```

Speciális mátrixok:

- `zeros`: csupa 0
- `ones`: csupa 1
- `eye`: diagonálisban 1, máshol 0
- `diag`: négyzetes diagonális mátrix, megadott f?átlóval

```
zeros(2,3)
eye(2,3)
ones(3,1)
diag([1,2,3,4])
```

Próbáljuk ki:

```
size(5)
size([1,2,3])
size([1;2;3])
```

Tartományok

A tartományok speciális sorvektorok, próbáljuk ki:

```
1:10
```

Ha nem egyesével akarunk ugrani:

```
1:0.1:2
1:2:10
```

Komplex számmal nem lehet, mert azok nem rendezhet?ek!
Az eredmény mindig `double` lesz, de utána konvertálhatjuk:

```
int32(1:0.5:10)
```

Leszálló tartományok:

```
4:-1:1
```

Üres tartomány:

```
4:1:1
```

Diagonális mátrixot megadhatunk így is:

```
> diag(1:4)
ans =
  1 0 0 0
  0 2 0 0
  0 0 3 0
  0 0 0 4
```

typeinfo

Egy érték típusáról meggy?z?dhetünk a `typeinfo` paranccsal.

```
typeinfo(1)
```

```
typeinfo(int32(1))
typeinfo(i)
typeinfo(single(i))
typeinfo([1 2 3 4])
typeinfo([1 i -1 -i])
typeinfo(1:4)
typeinfo([1:4])
```

M?veletek mátrixokkal

Mivel minden szám egyben egy `1x1`-es mátrix, így ezek mindig használhatóak.

Transzponált

Transzponált egyszer?en **vessz?vel** (`'`):

```
> [1 2; 3 4]'
ans =
 1 3
 2 4
> _
```

Vagy

```
> (1:4)'
ans =
  1
  2
  3
  4
```

Komplex mátrixokra a vessz? adjungálást jelent:

```
> [1,2i;3i,4]'
ans =
  1 - 0i   0 - 3i
  0 - 2i   4 - 0i
```

Konjugálást így csinálhatunk: `i'`

Összeadás

```
1+(1:4)
eye(2,2)+ones(2,2)
[1;2;3;4]-[4;3;2;1]
```

Szorzás

Minden szorzás mátrixszorzás:

```
> [1 2; 3 4]*[1 2; 3 4]
ans =
   7   10
  15   22
```

Hatványozás szintén, így az invertálás is:

```
[1 2; 3 4]^2
[1 2; 3 4]^-1
```

A szorzásnál a méreteknek kompatibiliseknek kell lenniük:

```
ones(2,3)*ones(3,5)
```

Sorvektor szorozva oszlopvektorral a skalárszorzás, fordítva diádszorzatnak hívjuk:

```
[1,2,3]*[1;2;3]
[1;2;3]*[1,2,3]
```

Tagonként vagy mátrixként

Ha a hatványozást ismételt mátrixszorzásként értelmezi, akkor ez mi?

```
[1 2; 3 4]^0.5
```

És ez mi?

```
sqrt([1 2; 3 4])
```

Bizonyos m?veletek *tagonként hatnak* ha egy mátrixra alkalmazzuk, míg mások mátrix-m?veletként. De tudunk váltani köztük.

```
> (1:4)^2
error: for A^b, A must be a square matrix
```

Hibát ad, mert két `1x4`-es mátrixnak nem értelmes a szorzata. De:

```
> (1:4).^2
ans =
  1  4  9 16
```

Minden m?veleti jel olyan, hogy **ha elé pontot rakunk, akkor elemenként hat**. Például az összeadásnál a mátrix összeadás és az elemenkénti összeadás ugyan az.

```
> [1 2; 3 4]+[1 2; 3 4]
ans =
  2   4
  6   8
> [1 2; 3 4].+[1 2; 3 4]
ans =
  2   4
  6   8
```

De a szorzásnál már nem:

```
> [1 2; 3 4]*[1 2; 3 4]
ans =
   7   10
  15   22
> [1 2; 3 4].*[1 2; 3 4]
ans =
   1    4
   9   16
```

Hatványozás hasonlóan:

```
> [1 2; 3 4]^-1
ans =
 -2.00000   1.00000
  1.50000  -0.50000
> [1 2; 3 4].^-1
ans =
  1.00000   0.50000
  0.33333   0.25000
```

A nevesített függvények általában elemenként hatnak:

```
sin(0:0.1:2*pi)
exp([0,-1;1,0])
```

A m?veleti jelek pedig mátrix m?veletként (`*, ^, /, \`)

Változók

Ahhoz hogy ne csak egy soros dolgokat tudjunk számolni, az adatokat *változókban* tároljuk.

```
a=2
b=3
a+b
```

Mindig van egy `ans` nev? változónak, amiben az *utoljára kiszámolt érték* található.
Ha nincsen érték adva egy változónak, akkor nem tudunk hivatkozni rá:

```
> a/q
error: `q' undefined
```

A kett?sponttal (`;`) csendes számolást végezhetünk, ekkor a parancs eredménye nem kerül kiírásra:

```
a=2;
b=3;
a+b
```

A `whos` paranccsal megnézhetjük az aktuálisan tárolt változóinkat.

```
> whos
Variables in the current scope:
  Attr Name        Size                     Bytes  Class
  ==== ====        ====                     =====  =====
       a           1x1                          8  double
       ans         1x1                          8  double
       b           1x1                          8  double
Total is 3 elements using 24 bytes
> _
```

Egy változó értékét bármikor felülírhatjuk:

```
> a=2;
> a=[1,2;3,4];
> whos
Variables in the current scope:
  Attr Name        Size                     Bytes  Class
  ==== ====        ====                     =====  =====
```

```
       a           2x2                         32  double
```

Indexelés

Legyen `M` egy `3x3`-as mátrix. Ennek az i-edik sorának j-edik eleme a következ?.

```
M = rand(3,3);
i = 1;
j = 3;
M(i,j)
```

Mátrixok összef?zése:

```
[M M]
[M; M]
```

Részsorozat kiválasztása a tartományok használatával:

```
l=0:0.1:1;
l(:)
l(1:11)
l(1:5)
l(5:end)
l(1:2:11)
```

S?t:

```
l(1:2:11)=0
```

Részmátrix hasonlóan, csak két indexszel.

```
A=[1,2,3;4,5,6;7,8,9];
A(1:3,1:2)
A(1:2,1:3)
```

Egy sor kihagyása:

```
A([1,3],:)
```

Vagy részmátrix kiválasztása:

```
S=ones(8,8);
S(3:6,3:6)=-1
```

Vagy adott indexekre:

```
S=ones(8,8);
S([1,2,8],[2,4,6])=-1
```

Mátrix kilapítása:

```
A=eye(3,3);
A(:)
```

Vektorizáció

Az Octave-ban (MatLab-ban) általában egyszerre sok dolgot számolunk, nem csak egy értéken értékelünk ki egy függvényt. Például az `X` mátrix minden sorának számoljuk ki a normáját (`X` lehet `nx3`-as, ahol `n` nagyon sok):

```
X = rand(10,3);
sqrt(sum(X.^2,2))
ans =
  0.99105
  0.86977
  1.29362
  0.91697
  1.26149
  0.84024
  1.45410
  1.19791
  1.01153
  1.07420
```

Belülr?l kifelé haladva elemezzük a függvényeket:

- `X.^2`: kiszámolja az elemenkénti négyzetet
- `sum( , 2)`: összegzi a mátrix sorait egy oszlopvektorba
- `sqrt`: elemenként gyököt von

Számoljuk ki a `2x^2-3x+1` függvényértékeket, ahol `x` egy sorvektor:

```
x=0:0.1:1;
fx=2.*x.^2 - 3.*x + 1
```

A *vektorizáció* lényege, hogy ahol lehet mátrix és vektor m?veletekre vezessük vissza a számításainkat, mert **1000 darab számpár összeszorzása lassabb, mint két darab 1000 hosszú vektor szorzása**!

Függvények

Írjuk be az `octave` a parancssorába a következ?ket, vigyázzunk több soros lesz!

```
> function fx = f(x)
>  fx=1/(x^2+1);
> endfunction
```

Majd próbáljuk ki:

```
> f(3)
ans = 0.10000
```

Függvények megadása:

```
function <<az eredmény>> = <<a függvény neve>> ( <<változó>> )
...
endfunction
```

A függvény hasában bármit számolhatunk, de a végén adjunk értéket `<<az eredmény>>` változónak. A függvény hasában érdemes csendes számítást végezni, itt használjunk mindenütt pontosvessz?t (`;`) a sor végén!

Egy másik függvény:

```
function R = remove_last(x)
 R = x(1:end-1);
endfunction
```

Példa:

```
> remove_last(1:5)
ans =
  1   2   3   4
```

Feladatok

Mi ez?

Figyeljük meg a következ?ket.

```
A=[1,2,3;4,5,6;7,8,9];
B=[9,8,7;6,5,4;3,2,1];
trace(A*B')
A(:)'*B(:)
```

Mi a `trace(A*B')`?

LER

Számoljuk ki a következ? lineáris egyenletrendszer megoldását:

```
 x + 2y = 3
4x + 5y = 6
```

Megoldás:

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

és ekkor egyszer?en:

```
x=A^-1*b
```

Erre van egy speciális szintaxis:

```
(A^-1)*b = A\b
```

És inverzzel jobbról szorozva:

```
B*(A^-1) = B/A
```

Még LER

Oldjuk meg a következõ egyenletrendszereket:

```
 x + 5y = 1
2x + 4y = 2
```

```
 x + 5y = 1
2x + 4y = 2
5x - 6y = -1

 x + 2y + 5z = 1
5x + 4y + 6z = 2
```

Nagy mátrix okosan

- Készítsük el a következõ mátrixot okosan! (Minél kevesebb karaktert használva.)

```
1 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2
2 2 3 2 2 2 2 2 2 2
2 2 2 4 2 2 2 2 2 2
2 2 2 2 5 2 2 2 2 2
2 2 2 2 2 6 2 2 2 2
2 2 2 2 2 2 7 2 2 2
2 2 2 2 2 2 2 8 2 2
2 2 2 2 2 2 2 2 9 2
2 2 2 2 2 2 2 2 2 10
```

- És most ezt:

```
 0  1  0  0  0  0  0  0  0
-1  0  1  0  0  0  0  0  0
 0 -1  0  1  0  0  0  0  0
 0  0 -1  0  1  0  0  0  0
 0  0  0 -1  0  1  0  0  0
 0  0  0  0 -1  0  1  0  0
 0  0  0  0  0 -1  0  1  0
 0  0  0  0  0  0 -1  0  1
 0  0  0  0  0  0  0 -1  0
```

- Sakktáblaszabály

```
  1  -1   1  -1   1
 -1   1  -1   1  -1
  1  -1   1  -1   1
 -1   1  -1   1  -1
  1  -1   1  -1   1
```

Függvény mátrixokon

Írjunk függvényt, mely az adott mátrix minden elemére alkalmazza a $2\sin^2 x + 1$ függvényt.

Részmátrix

Írjunk függvényt, mely egy 5x5-ös mátrix 2. és 4. sorából és 1., 3. és 5. oszlopából álló mátrixot adja.

Részmátrixon függvény

Írjuk meg az elõzõ két függvény kombinációját, mely az adott mátrix 2. és 4. sorából és 1., 3. és 5. oszlopából álló mátrixon alkalmazza a $2\sin^2 x + 1$ függvényt.

Minden második oszlop

Írjunk függvényt, mely tetszõleges mátrix minden második oszlopából álló mátrixot adja vissza. (Segítség, a `size` sorvektorban megadja a mátrix dimenzióját.)

Függvény alkalmazás csak adott elemeken

Írjunk függvényt, mely a kapott mátrix csak minden második oszlopán hajtja végre a $2\sin^2 x + 1$ függvényt. (Az eredmény mátrix dimenziója ugyanaz, mint a kapott mátrix.)

Segíts magadon

A help segítségével próbáljuk kiszámolni a következ?ket.

- `A` determinánsa
- `A` saját értékei, saját vektorai

Numerikus deriválás

Deriváljuk az `f(x)=2x^2-3x+1` függvényt numerikusan! Adott egy `x` sorvektor, ami az abszcissza értékeket tartalmazza, `fx` pedig a hozzájuk tartozó függvényértékeket.

```
x=0:0.1:1
fx=2.*x.^2 - 3.*x + 1
```

Ekkor a függvény numerikus deriváltja:

```
df = (fx(2:end) - fx(1:end-1))./0.1
```

Nem egyenletes lépésközzel pedig:

```
df = (fx(2:end) - fx(1:end-1))./(x(2:end)-x(1:end-1))
```

El?z? gyakorlat - Fel - Következ? gyakorlat