El?z? gyakorlat - Fel - Következ? gyakorlat

Tartalomjegyzék

MatLab

Az MatLab program alkalmas különböz? matematikai számításokat numerikusan elvégzésére, köztük

- lineáris (mátrix) egyenletek megoldása
- differenciál-egyenletek, megoldása
- szabályozástechnika

Neve a *Matrix Laboratory* rövidítése, a 70-es években kezdte el fejleszteni *Cleve Moler*, a *University of New Mexico* egyetem számítástudományi tanszékének vezet?je. Azóta a mathworks cég fizet?s terméke lett.

Van egy ingyenes alternatívája, az Octave.

Kezdeti lépések

Hozzáférés a programhoz

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

- A BME-t?l kaphatunk licenszet: https://net.bme.hu/sw/. Kövessük az utasításokat a honlapon (BME-s hálózaton belülr?l érhet? el).
- Putty-al lépjünk be a `leibniz`-re és írjuk be a terminálba, hogy `matlab`

A Leibniz-en telepítve van, gépteremb?l is használhatjuk.

Számológép

Egy terminálban adjuk ki a következ? parancsot:

```
matlab -nojvm
```

A MatLab egy fejlettebb számológépként is használható. Írjuk be 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
```

MatLab-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 `MatLab`-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. 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
4:1
```

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
```

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, azaz a transzponált elemenkénti konjugáltját:

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

Konjugálást így csinálhatunk: `(2 + 1i)'`

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, nincsen `.`+ m?velet.

```
> [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 pontosvessz?vel (`;`) **csendes számolást** végezhetünk, ekkor a parancs eredménye nem lesz kiírva:

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

Akkor hasznos, ha az eredményt úgyis elmentjük egy változóba és külön nem akarjuk látni. Kés?bb is megnézhetjük.

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 6]) = 8
```

Mátrix kilapítása:

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

Összegzés

- A mátrix sorait összegzi a `sum( , 1)`
- A mátrix oszlopait összegzi a `sum( , 2)`

Azon dimenzió szerint összegez, ami a második paraméter. Ha nincsen második paraméter, akkor 1-nek tekinti. Az összes elemet kilapítással összegezhetjük:

```
sum(A(:))
```

Vektorizáció

Az 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 (a 2 azt jelenti, a tömb második dimenziója, azaz a 2. index mentén)
- `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 a következ?ket egy `f.m` nev? fájlba!

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

Ha a fájl neve megegyezik a függvénnyel és abban a mappában van, ahonnan a programot indítottuk, akkor így használhatjuk:

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

Függvények megadása:

```
function <<az eredmény>> = <<a függvény neve>> (<<változók>>)
  ...
end
```

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);
end
```

Ezt egy `remove_last.m` fájlba tegyük. Példa:

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

Feladatok

Elemi sorm?veletek elvégzése

Hozzunk létre egy egyjegy? nemnegatív egészekb?l álló 4x5-ös mátrixot, majd cseréljük ki két sorát, szorozzuk be a második sorát 2-vel és adjuk az els? sorának kétszeresét a harmadik sorhoz! (A mátrix redukált lépcs?s alakjának meghatározására azért a `rref` függvényt használjuk!)

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
```

1. megoldás:

```
A = [1 2; 4 5]
b = [3; 6]
x = A^-1 * b
```

2. megoldás:

```
x = A \ b
```

Még LER

Oldjuk meg a következ? 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 következ? 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 tetsz?leges 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

- https://www.mathworks.com/matlabcentral/answers
- http://www.mathworks.com/help/matlab/
- Google

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