Tartalomjegyzék

Octave

Alapvet? m?veletek és függvények

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

Octave-ban egy szám mindaddig valós (lebeg?pontosan ábrázolva), amíg komplexnek nem bizonyul:

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

Mátrixok

Sorvektor:

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

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])
```

Tartományokat adhatunk meg gyorsan, pl, 1, 2, 3, 4, 5:

```
1:5
```

Vagy más lépésközzel (akár negatívval), pl 1, 1.2, 1.4, 1.6, 1.8, 2:

```
1:0.2:2
```

Ezeket lehet használni mátrixok létrehozásakor, pl:

```
diag(1:4)
```

Vehetjük mátrixok transzponáltját:

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

összegét:

```
eye(2,2)+ones(2,2)
```

szorzatát:

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

hatványát:

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

Ezek a m?veletek mátrixokon hatnak. De végezhetjük a m?veleteket tagonként is:

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

A legtöbb m?velet elé, ha pontot rakunk akkor tagonként hat.

Lekérhetünk és módosíthatjuk egy adott mátrix valamelyik elemét:

```
M = [1 2; 5 4]
```

```
M(2,1) = 3
```

Ez így a második sor els? elemét módosítja.

Változók

Változókban tárolhatunk adatokat, például:

```
a = 5
```

de akár mátrixokat is tárolhatunk:

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

Változók értéke felülírható, és változók használhatók értékadásnál is, például:

```
M = [1; 1]
a = 4
M = [a 2; 4 a]
```

Függvények

Függvényt írhatunk Octave-ban:

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

Majd így hívhatjuk meg:

```
f(3)
```

ennek az eredménye pl 0.1 lesz.

Sage

Alapvet? m?veletek

```
2 + 5
4 * 2
6 - 1
4 / 3
4 ^ 2
```

Ne felejtsük el, hogy a 4/3 eredménye nem 1.3333 lesz, hanem 4/3, a Sage meghagyja szimbolikusan az eredményt az Octave-al ellentétben.

Változók

Változók hasonlóan m?ködnek, mint Octave-ban:

```
a = 4
b = a * 4
a = 3
```

Ezt lefuttatva a értéke 3 lesz, míg b értéke 16.

Egyenlet megoldás

Egyenletet oldhatunk meg a *solve(fv, változó)* függvénnyel:

```
x = var('x')
solve(x^2 + 2*x - 1 == 0, x)
```

Figyeljük meg, hogy dupla egyenl?ségjel van az egyenletben. A sima 1 egyenl?ségjel az értékadást jelöli, míg a 2 egyenl?ségjel az összehasonlítást. Valamint ne felejtsük el, hogy a szimbolikus változók csak akkor szimbolikusak Sage-ben, ha annak definiáljuk ?ket.

Függvényt is menthetünk változóba:

```
fv = 2 * x + 3
solve(fv == 4, x)
```

Megoldhatunk egyenletet numerikusan is (ha szimbolikusan nem megoldható):

```
find_root(abs(sin(x))^cos(x) == log(x), -10, 10)
```

Ekkor az utolsó két paraméter egy intervallumot határoz meg, amin belül keresi a megoldást.

Deriválás, integrálás

A *diff* paranccsal deriválhatunk egy függvényt:

```
diff(x^2 * sin(x), x)
```

Míg az *integrate*-el integrálhatunk:

```
integrate(x^2 * sin(x), x)
```

Függvény rajzolás

Függvényeket kirajzolhatunk a *plot* paranccsal:

```
plot(x^2 * sin(x), 0, 10)
```

Itt az utolsó két paraméter az intervallum ahol kirajzolja a grafikont (*x* koordináta).

Kirajzolhatunk több függvényt is, ha változóba mentjük ?ket, majd a *show*-val közösen mutatjuk ?ket:

```
p1 = plot(x^2 * sin(x), 0, 10)
p2 = plot(x^2 * cos(x), 0, 10)
show(p1 + p2)
```

Figyeljük meg, hogy össze kellett adni a két függvény rajzot, így tudtuk egybe kirajzolni.

Listák

Listákat máshogy kell készítenünk, mint Octave-ban:

```
simaLista = [2, 3, 6]
tobbSzintu = [[2, 3], [4, 7]]
felsorolas = range(1, 10) # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Listaértelmezés

Ezt a részét vizsgáltuk a legrészletesebben a Sage-nek:

```
[kifejezés for elem in bejárható_objektum]
```

Egy olyan listát hoz létre melyben a *kifejezés* szerepel a *bejárható_objektum* minden elemére. Bejárható objektum például egy lista, az is amit a *range* függvény hoz létre.

```
[kifejezés for elem in bejárható_objektum if feltétel]
```

Mint az el?z?, de csak azok az elemek lesznek benne melyekre teljesül a *feltétel*.

```
[kifejezés for elem1 in bejárható_objektum1 if feltétel1
           for elem2 in bejárható_objektum2 if feltétel2
           for elemN in bejárható_objektumN if feltételN]
```

Több feltétel és ciklus is írható akár.

Például az összes prím pár, melyeknek az egészosztottjuk is prím: [(n, m) for n in range(1, 100) for m in range(1, 100) if is_prime(n) and is_prime(m) and is_prime(n // m)]

Wolfram Mathematica

Listák

Listákat kapcsos zárójellel adunk meg

```
{1,2,3,4}
```

Mátrixokat listák listájaként:

```
{{1,2}, {3,4}}
```

De listában bármi lehet:

```
{1,x,{E,Pi}, Function[x,x^x]}
```

Hasznos parancsok

- Range

```
In[1]:= Range[5]
Out[1]= {1,2,3,4,5}
```

```
In[2]:= Range[2,6]
Out[2]= {2,3,4,5,6}

In[3]:= Range[2, 5, 0.5]
Out[3]= {2., 2.5, 3., 3.5, 4., 4.5, 5.}
```

- Table

Range-hez hasonlóan

```
In[1]:= Table[i,{i,1,5}]
Out[1]= {1,2,3,4,5}
```

A Table hasába bármilyen kifejezést írhatunk

```
In[1]:= Table[x^i,{i,1,5}]
Out[1]= {x,x^2,x^3,x^4,x^5}
```

Az elemek explicit felsorolásával

```
In[1]:= Table[f'[x],{f,{Sin, Cos, Log}}]
Out[1]= {Cos[x], -Sin[x], 1/x}
```

Több dimenziós Table, például szorzótábla 1-tõl 10-ig:

```
Table[i * j, {i, 1, 10}, {j, 1, 10}]
```

Függvények hattatása

- A függvényhívás jele a **szögletes zárójel**! `Sin[x]`
- Postfix jelöléssel: `x//Sin`. Azt mondhatjuk: *x-re hattatom a szinuszt*
- Prefix jelöléssel: `Sin@x`. Más szóval: *a szinusz hat x-en*
- A dupla hattatás argumentumául veszi a lista elemeit:

```
f @@ {x,y,z} === f[x,y,z]
```

Más mint az alábbi:

```
f @ {x,y,z}
```

ami

```
f[{x,y,z}]
```

Például vegyük a `Sin` függvényt, ekkor az alábbiak megegyeznek:

```
Sin[Pi]
Pi // Sin
Sin @ Pi
Sin @@ {Pi}
```

Az összeadásra:

```
x+y
Plus[x,y]
Plus @@ {x,y}
```

```
{x,y} // Total
Total[{x,y}]
```

Listára elemenként hattathatunk függvényt a `Map` függvénnyel vagy a `/@` szimbólummal.

```
f/@{x,y,z} === {f[x],f[y],f[z]}
```

Példák

```
Table[Cos[x], {x,0,Pi,Pi/6}]
Cos /@ Table[x,  {x,0,Pi,Pi/6}]
Cos /@ Range[0,Pi,Pi/6]
Cos[Range[0,Pi,Pi/6]]
```

Függvények megadása

Az alábbiakban ugyanazt a függvényt definiálom:

```
f[x_] := Sin[x]
f = Sin[#]&
f = Sin
f = Function[x,Sin[x]]
```

Ezek bármelyikével a következ? eredményt kapom

```
In[1]:= f'[x]
Out[1]= Cos[x]
```