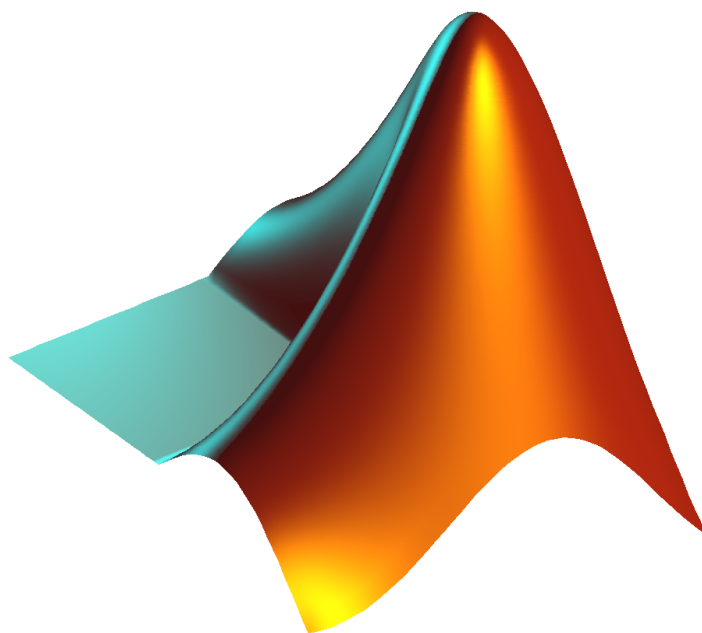


Miliczki József

MatLab Bootcamp

Egy átfogó dokumentáció a Matrix Laboratory alapvető
használatához



2021. 09. 27

Tartalomjegyzék:

Bevezető	2
Alapvető eszközök	3
A munka felület és változók	3
Output megkerülése	4
Legnagyobb és legkisebb lebegőpontos számok	4
Több parancs egy sorban	5
Sortörés végrehajtás nélkül	5
Változó kilistázás	6
Változó(k) törlése	7
Formázás	8
Konzol törlése	9
Mátrixok	10
Vektor: Az egysoros mátrix	10
A mátrix	13
Mátrix generálás	16
Műveletek mátrixon	20
Műveletek mátrixok között	29
Haladó mátrix kezelés	35
M-FILE	36
Létrehozás	37
Futtatás	37
Programozói eszközök	38
Függvény Írása	47
Grafika	49
Plot	49

Bevezető

Ezen dokumentáció célja megismertetni a Matrix Laboratory programot és nyelvet alapvető / középhaladó szinten. A dokumentumban feltüntetett anyagok saját tesztelési eredményekből és - főleg - a MatLab saját dokumentációjából származik. Az ott fellelhető anyagokat kipróbáltam, megnéztem mikor működnek és mikor nem. A tapasztalataimat pedig megosztom veled, kedves olvasó.

A MatLab alapján véve egy rendkívül komplex kalkulátor, amely - hihetetlenül meglepő módon - mátrixok köré építi a matematikát. Ez viszont nem azt jelenti, hogy az alkalmazási területe csak a kettő-és több dimenziós matematikai egységek körében alkalmazható, hiszen a nyelv képes bonyolult egyenletek megoldására, függvényábrázolásra és még rengeteg más, a matematika fogalmán belül helyet foglaló feladatok elvégzésére. Mi kifejezetten a mátrixokra fogjuk kihegyezni a tekintetünket, majd később megtanulunk M-FILE-okat írni, valamint betekintünk a függvényábrázolás függőnye mögé is, "megkukkantjuk" mit, hogyan, és miért.

A dokumentum egy alapvető programozói tudást elvár, nem megyünk bele hogy működik egy IF vagy miért fut le egy FOR pont 10-szer. Ezeket már számtalanszor elmagyarázták nálam sokkal képzetesebb emberek. Ezen kívül csak a figyelmed kéri meg a dokumentum!

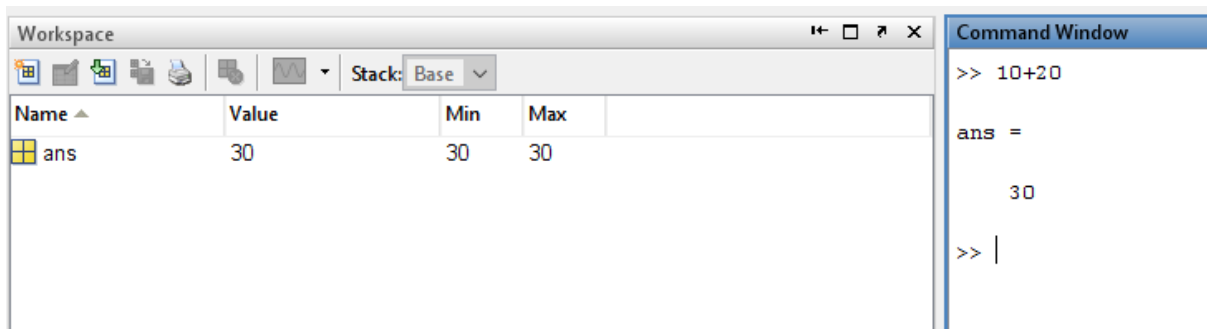
Nem is szaporítom tovább a szót, engedjük ki a Krakent...

Miliczki József

Alapvető eszközök

A munka felület és változók

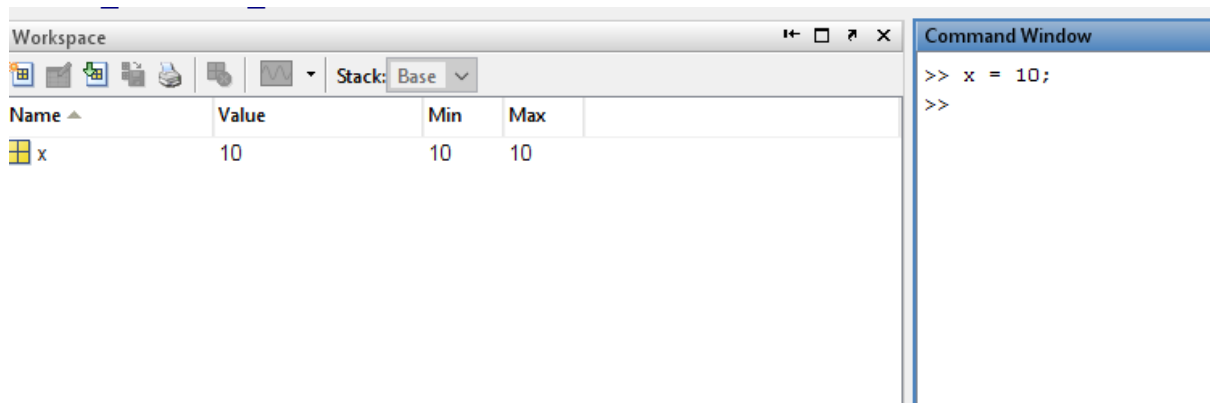
Ez a felület mindennek az alapja. Itt tudunk kiadni különböző utasításokat, értékadásokat, egy kicsit olyan, mintha élőben programoznánk, hiszen ha kiadjuk, hogy $x=10$, akkor az rögtön létrehoz egy x változót 10-es értékkel. Erről lejjebb bővebben. Az ablakot magát el is lehet tüntetni, ilyenkor a Window > Command Window menüpontot kell kiválasztanunk ha vissza szeretnénk szerezni a munka felületünket. Érdeemes figyelmet adni még a Current Directory és Workspace menüknek is. A Workspace lényegében a programban “élő” változókat mutatja, míg a Current Directory azt a mappát listázza ki, amit éppen M-FILE-ok számára jelölt ki a program (ez általában a Dokumentumok > MATLAB mappa). Érdekes dolog még a változókról: Mivel ez a program mégis csak Matrix Laboratory, ezért nem meglepő, hogy lényegében minden szám (és egyéb típus) egy mátrixban helyezkedik majd el. Ez csak terminológia igazából, 1 db szám egy 1x1-es mátrixban helyezkedik el, ami pont úgy viselkedik mintha csak simán egy szám lenne. Fontos megjegyezni a változóknál a tényt, hogy ha nem adunk meg változó nevet, az operáció így is végrehajtódik, és eltárolódik egy alapértelmezett változóba. Ez az `ans` (Answer). Ebbe mindig a legújabb olyan eredmény fog bekerülni, amelyet nem adunk át egyetlen változónak sem.



Az `ans` változó létrejön

Output megkerülése

A MatLab-ben minden operáció valamilyen visszajelzéssel, outputtal csatol vissza a felhasználó felé. Ez mondjuk kényelmes ha figyelni szeretnénk, mi a számolás menete, viszont ha 0-tól 0.1-es lépésekkel elmegyünk 1000-ig, akkor nem feltétlenül szeretnénk látni mind a 10.000 lépést a konzolban. Ilyenkor a más nyelvekben már jól ismert pontosvessző(;) kerül a képbe. Ha ezt egy parancs végére tesszük ki, akkor az output kiírását megkerüljük, a program nem mutatja meg a számítás eredményét. Megjegyzés: A Workspace-ben így is meg tudjuk nézni a kalkuláció végeredményét! Csak menjünk rá a változóra amivel számoltunk!



Az outputot ignoráljuk, de a változó létrejön

Legnagyobb és legkisebb lebegőpontos számok

Minden számítógépnek van egy számításbeli határa, amely azt takarja, mekkora számokat tud kiírni. A következő 2 paranccsal lekérhetjük a MATLAB-tól azt a 2 lebegőpontos számot, ami a legkisebb és a legnagyobb (ilyen sorrendben). Ezt exponenciálisan fogja megadni a program.

Legkisebb lebegőpontos szám: `realmin`

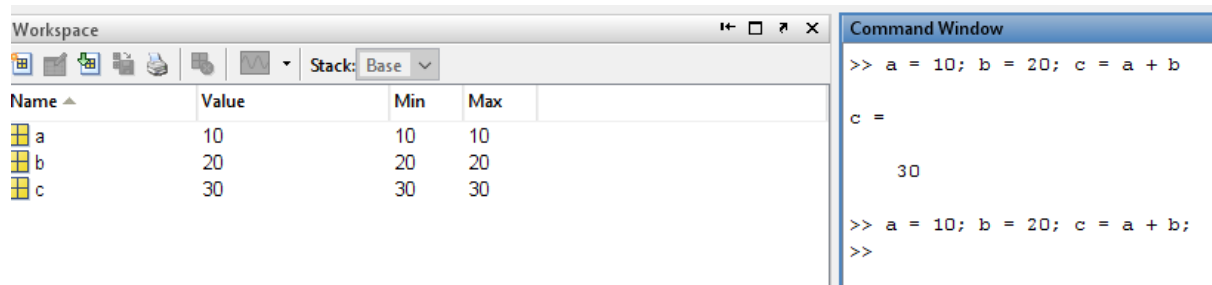
Legnagyobb lebegőpontos szám: `realmax`

```
>> realmin  
  
ans =  
  
2.2251e-308  
  
>> realmax  
  
ans =  
  
1.7977e+308  
  
>> |
```

A realmin és a realmax exponenciális értéke

Több parancs egy sorban

Előfordulhat, hogy a felhasználó több parancsot szeretne végrehajtani egyszerre, anélkül, hogy egyenként ENTER lenyomásokkal fégrehajtsa azokat. Ilyenkor a már ismert [output megkerülő](#) karakterrel elszeparáljuk a parancsokat, és egymás után írjuk őket. Példa:

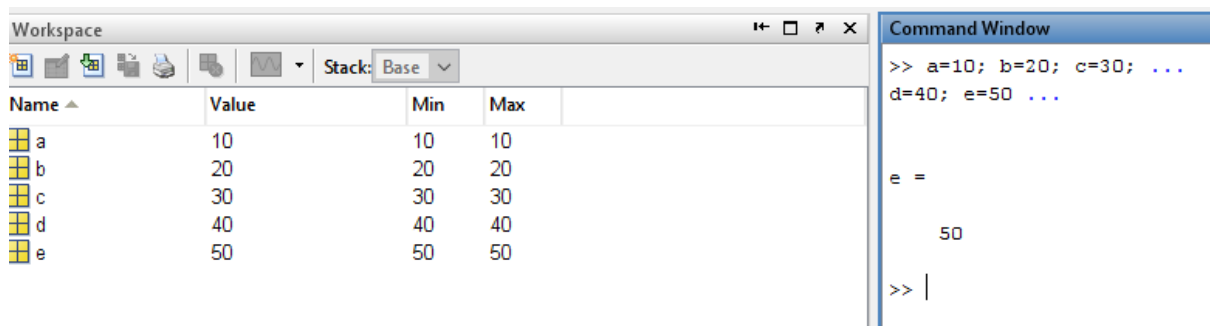


Több parancs egy sorban.

A sor végét természetesen terminálhatjuk a ; karakterrel, így nem írjuk ki a kalkuláció eredményét.

Sortörés végrehajtás nélkül

Ha rengeteg parancsot szeretnénk kiadni egy utasításban (sorban), akkor elég macerás egy sorba pontosvesszőkkel elválasztani minden egyes lépését a kalkulációnak. Ilyenkor használhatunk 3 pontot, amivel eltörjük a sort és átláthatóbb lesz a parancsok folyamata. A 3 pont kiírása után nyomjunk egy ENTER-t az új sorhoz!



Sortörés parancsokkal. Miért íródott ki az e változó? Nincs pontosvessző!

Változó kilistázás

A konzolos felületen kilistázhatjuk a változókat 2 féle módon: Csak a nevüket, tehát milyen változók léteznek jelenleg a memóriában; és részletesen, amely leírja mennyi helyet foglalnak, mi a típusuk, és további attribútumokat.

`who` = Milyen változóink vannak? (Egyszerű)

`whos` = Milyen tulajdonságokkal rendelkeznek a változóink?
(Részletes)

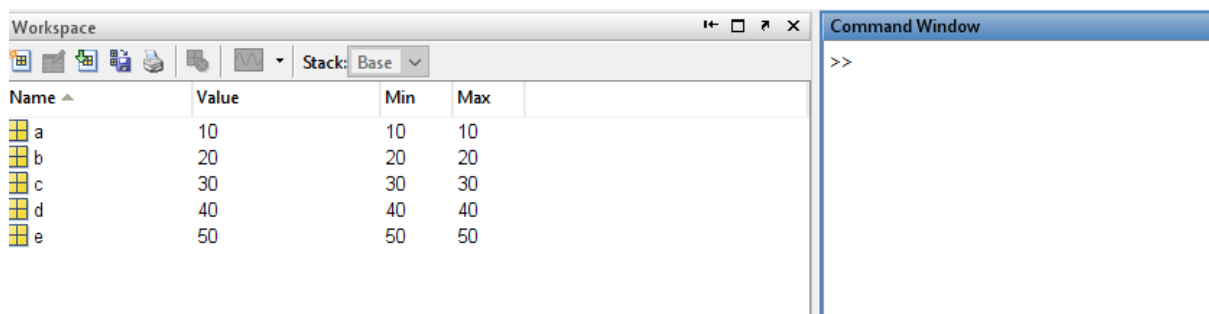
```
>> who  
  
Your variables are:  
  
a b c d e f  
  
>> whos  
  
Name      Size      Bytes  Class  Attributes  
  
a         1x1         8  double  
b         1x1         8  double  
c         1x1         8  double  
d         1x1         8  double  
e         1x1         8  double  
f         1x1         8  double  
  
>>
```

Egyszerű és részletes változó kiíratás

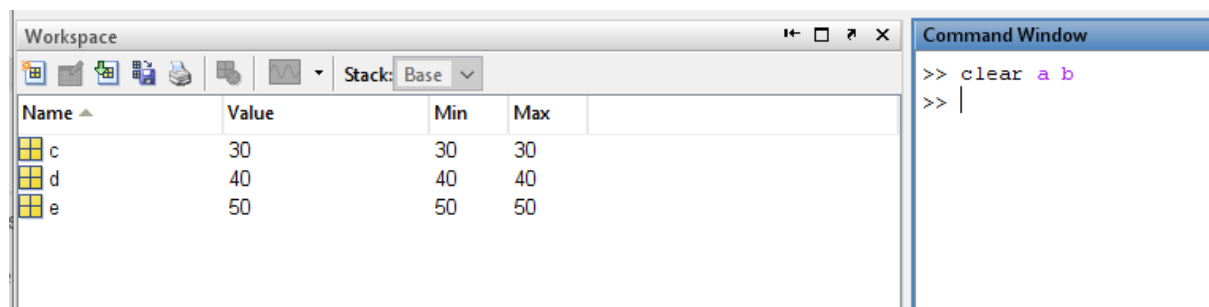
Érdekesség: A MatLab minden változót alapvetően double típussal tárol. Ez minket nem akadályoz meg semmiben, pontosabbak lesznek a számításaink. Van lehetőség más numerikus típusok megadására, mint például egész Integer, ám ezekkel nem foglalkozunk most. Számításaink szempontjából a double a preferálandó típus.

Változó(k) törlése

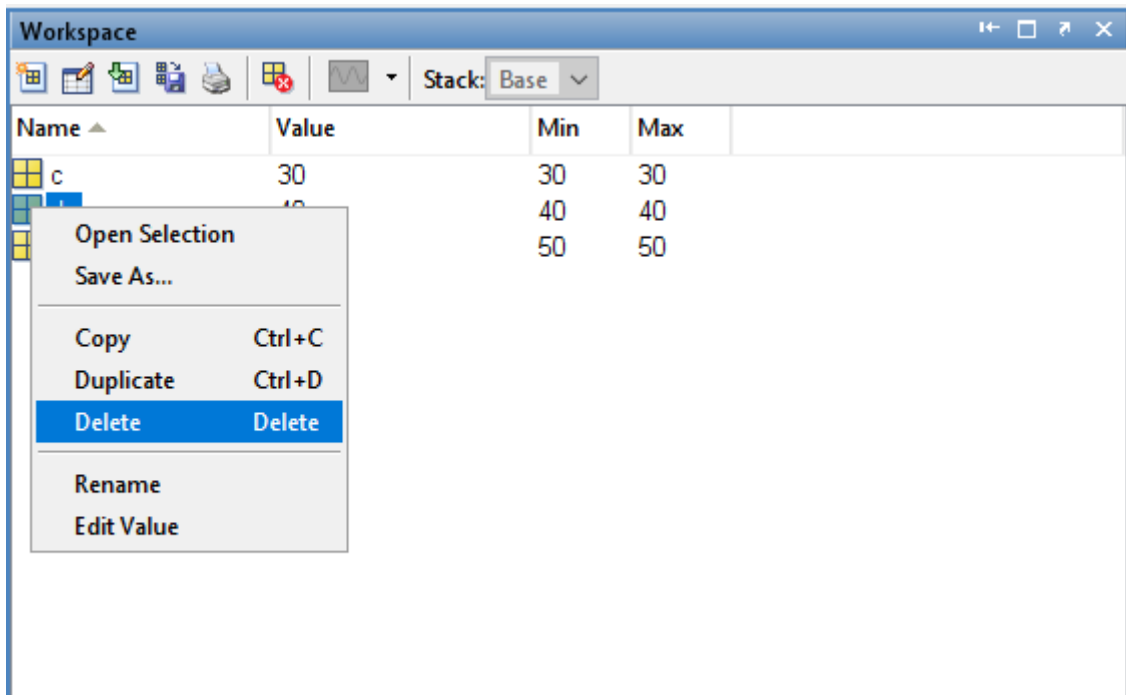
Változót törölni a `clear` paranccsal tudunk. Ennek használata 2 féleképpen történhet. Ha simán kiadjuk a `clear` kulcsszót, akkor minden változó törlésre kerül, viszont ha a `clear` argumentumaként megadunk egy (vagy több) változót, akkor csak a megadott változók kerülnek terminálásra. Alternatívan lehet még a Workspace menüben kézzel törölni (jobb klikk > Delete, vagy csak simán kijelöljük a változót és nyomunk egy Del gombot a billentyűzeten).



Törlés előtt...



...és törlés után



Az alternatíva

Formázás

A MatLab alapvetően 4 tizedesjegyre írja ki a számokat:

```
Command Window
>> a = 0.123123212143

a =

    0.1231

>>
```

A short formátum

Ez az alapértelmezett formátuma a MatLab-nak, amely a short. Formátumot válthatunk a format kulcsszóval, majd utána argumentumként meg kell adnunk, hogy milyen formátumra szeretnénk váltani. Rengeteg formátumot támogat a MatLab, ezek közül csak a legfontosabbakat fogjuk most megemlíteni:

format short = 4 tizedesjegy pontosság

format long = 15 tizedesjegy pontosság

format bank = banki(dollár) kerekítés, 2 tizedes jegy

format rat = racionális számaábrázolás

`format hex` = hexadecimális számábrázolás

A kódok élesben:

```
Command Window
>> a = pi

a =

    3.1416

>> format long; a

a =

    3.141592653589793

>> format bank; a

a =

    3.14

>> format rat; a

a =

    355/113

>> format hex; a

a =

    400921fb54442d18

>> |
```

Formátum váltás a pi-vel

Konzol törlése

A konzol egy idő után akarva akaratlanul is tele lesz mindenféle parancsokkal, amelyek egyesek számára idegesítő lehet. Ha egy újabb, tiszta kódfelületet szeretnénk, adjuk ki a `clc` (Clear Console) parancsot, és a MatLab lemossa az eddig felvázolt parancsokat a konzolról.

Mátrixok

Vektor: Az egysoros mátrix

Vektor létrehozása

Amint azt a címsor is állítja, a vektor az lényegében egy 1 soros, több oszlopos mátrix. Ennek fejében úgy is fogjuk definiálni a MatLab-ban. Vektor létrehozásakor 2 fő módszert szoktunk alkalmazni: Megadjuk az elemeket egyesével; vagy megadunk egy intervallumot amiben benne van minden olyan szám, amit szeretnénk látni a vektorunkban.

Az első módszer (felsorolás):

```
v1 = [1, 2, 3, 4, 5]
```

```
v2 = [1 2 3 4 5]
```

Amint látható, megadhatjuk az elemet vesszővel elválasztva, vagy simán space-el elszeparáljuk az elemeket egymástól. Mind a kettő teljesen megfelelő, ízlés kérdése.

Mielőtt beszélnénk a második esetről, meg kell ismernünk egy új operátort, ez a kettőspont (:). Talán a MatLab egyik legerősebb eszköze, általánosítva 2 dolgot jelent: Először is, “minden”, ergo ezzel fogunk majd a jövőbeli fejezetekben kijelölni minden elemet egy mátrixban, másodszorra pedig intervallum definiáló eszköz. Működését legjobban egy példával lehet bemutatni, ami egyben a második vektor készítési módszer:

```
v = [1:2:10]
```

Ezzel a paranccsal megmondtuk, hogy egy olyan vektort szeretnénk, ami 1-től 10-ig megy, **2-es lépésekkel**. Mivel az 1-től kezdünk, így a vektorban az alábbi számok fognak szerepelni: 1 3 5 7 9. Mivel a 11 már túlmutat a határon, ezért a 9 az utolsó elem. Ezt az intervallum generálást érdemes lesz megjegyezni, amint már írtam: RENDKÍVÜL ERŐS!

Még egy megjegyzés: A MatLab alapértelmezett lépésszáma az 1, így ha az írjuk, hogy

```
v = [1:10]
```

akkor valójában azt írjuk, hogy

```
v = [1:1:10]
```

ami azt jelenti: 1-től 10-ig egyes lépésekkel kérem az összes számot!

Ez természetesen egy 1 2 3 4 5 6 7 8 9 10 elemekből álló vektort fog eredményezni.

Példa az éles működésről:

```
Command Window
>> v = [1 2 3 4 5]

v =

     1     2     3     4     5

>>
```

Vektor építés megadott elemekkel

```
Command Window
>> v = [1:2:10]

v =

     1     3     5     7     9

>> |
```

Vektor építés intervallum definiálása

Vektor elem(ek) lekérése

A vektor elemeit zárójelben megadott paraméterrel tudjuk elérni. A paraméter maga az nem a szám amit szeretnénk, hanem annak a helye. Kérdezzük le a 3. vektor elemet!

$v(3)$ = “Kérem a 3. elemet”

De mi van akkor, ha több elemet is el szeretnénk érni? Akkor van szerencsénk, ha ezek az elemek egymás mellett vannak, vagy legalább valamilyen sorozatban, mivel ilyenkor használhatjuk az intervallum operátort!

$v(3:5)$ = "Kérem a 3. elemtől az 5. elemig mindent"

$v(1:2:5)$ = "Kérem az első elemet, és minden második elemet az 5. elemig!"

Nézzünk a fenti 3 lekérdezésre példát!

```
Command Window
>> v = [10:-1:1]

v =

    10     9     8     7     6     5     4     3     2     1

>> v(3)

ans =

     8

>> v(1:2:10)

ans =

    10     8     6     4     2

>> v(1:5)

ans =

    10     9     8     7     6

>> |
```

Vektor elemeinek lekérdezése

Bónusz: válasszunk ki minden elemet!

$v(:)$

Ez a korábban említett kettőspont másik tulajdonsága, megadjuk vele, hogy minden elemet kérünk a vektorból.

A mátrix

Mátrix létrehozása

Azt már láttuk, hogyan kell létrehozni egy vektort. A mátrixok létrehozása se sokkal bonyolultabb, ám itt már 2 dimenzióban kell gondolkodnunk (később betekintünk majd az n-dimenziós mátrixokba is!). A vektoroknál a sor az mindig 1 volt, ám most már több sorral kell dolgoznunk. Ahhoz hogy egy új sort csináljunk, a pontosvesszőt kell alkalmaznunk a mátrix megadásánál:

```
Command Window
>> m = [1 2 3; 4 5 6]

m =

     1     2     3
     4     5     6

>>
```

Mátrix megadása

Láthatjuk, hogy az elemek felsorolása úgy zajlik, mint a vektoroké, csak most új sorba tettük a 4 5 6 számokat, ezzel 2 dimenzióssá téve a mátrixot. Amikor megadunk egy sort, akkor alkalmazhatunk minden eddig tanult vektor képzési módszert. Például:

```
Command Window
>> m = [1:3 ; 4:6]

m =

     1     2     3
     4     5     6

>> |
```

Mátrix megadása intervallumokkal

Figyelem: Ügyelj arra, hogy a megadás konzisztens legyen! Ez alatt azt értem, hogy ha az első sorba 3 elemet adunk meg, akkor a

második sorba is 3 elemet definiáljunk. Ha nem így teszünk, akkor nyertünk egy szép piros MatLab exception-t.

Mátrixokat még generálhatunk egyéb módokkal is, ám ezeket majd a következő fejezetben tekintjük át!

Mátrix elemeinek lekérése

Egy mátrix elemeit hasonlóképpen érhetjük el, mint egy vektorét, ám ügyelnünk kell melyik sorból és oszlopból választunk. Fontos már most megjegyeznünk, hogy akármikor paramétert adunk meg egy mátrixról, akkor a sorrend MINDIG sor aztán oszlop! Ez alapján a filozófia alapján nézzük meg, hogy lehet kiválasztani például a 2. sorból a 3. elemet:

`m(2,3)`

```
Command Window
>> m = [1:3 ; 4:6]

m =

     1     2     3
     4     5     6

>> m(2,3)

ans =

     6

>> |
```

m mátrix létrehozása, és 2. sor, 3. elem kiírása

Itt is lehet használni a már vektornál tanult lekérdezési módszereket!

~MatLab Bootcamp~

```
>> m(1:2,1:2)
```

```
ans =
```

```
    1    2  
    4    5
```

```
>> m(1:2,1)
```

```
ans =
```

```
    1  
    4
```

```
>> |
```

Az első és második sorból lekérem az 1. és 2. oszlopot, majd az első és második sorból lekérem az 1. oszlopot

Használhatjuk a kettőspont operátort is, ekkor megmondjuk hogy abból az opcióból (sor vagy oszlop) mindent szeretnénk:

```
Command Window
```

```
>> m(:,3)
```

```
ans =
```

```
    3  
    6
```

```
>>
```

Mindent kérek a 3. oszlopból

```
>> m(2,:)
```

```
ans =
```

```
    4    5    6
```

```
>> |
```

A második sorból kérek mindent

Akár megadhatunk egy intervallumot is “Ettől eddig kérem a sorokból azokat az oszlopokat, amik ettől eddig mennek.”. Gyakorlatban:


```
Command Window
>> m(1:2, 1:3)

ans =

     1     2     3
     4     5     6

>> |
```

Az első és második sort kérem, ezekből pedig az 1.-től a 3. oszlopig mindent

Ahogy azt láthatjuk, rengeteg módon meg tudjuk adni egy mátrix elemeit, legyen az egy elem, több elem, vagy teljes sorok. Próbáljuk meg minél kreatívabban alkalmazni az eszközeinket! A legjobb módszer a sorok és oszlopok elérésének elsajátítására az a gyakorlás: Generáljatok saját mátrixokat, minél nagyobbakat, nézzetek ki magatoknak például egy elemet, aztán többet, és próbáljátok meg kiszedni őket a mátrixokból a fenit módszerekkel!

Mátrix generálás

Mátrixokat nem csak saját kezűleg tudunk megadni, hanem vannak specifikus mátrixok, amiket maga a MatLab tud nekünk generálni. Ezekre nézzük meg a leghasznosabb parancsokat! Ne feledjük: ezek a parancsok egy mátrixot adnak vissza! (A random szám kivétel lehet ez alól, erről majd később)

Nullmátrix

A nullmátrix egy olyan mátrix, amelynek minden eleme nulla. Ezt a `zeros(x)`

paranccsal tudjuk előállítani, ahol az x a sorok és oszlopok számát jelenti. Ebből adódik, hogy a mátrix egy négyzetes mátrix lesz.

Gyakorlatban:

Command Window

```
>> zeros(2)
```

```
ans =
```

```
    0    0
    0    0
```

```
>> zeros(4)
```

```
ans =
```

```
    0    0    0    0
    0    0    0    0
    0    0    0    0
    0    0    0    0
```

```
>> zeros(10)
```

```
ans =
```

```
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0
```

```
>>
```

Nullmátrix generálás különböző méretekben

“Egysésmátrix”

Nullmátrix párja. Ez egy olyan négyzetes mátrix lesz, amely minden eleme 1-esekből áll. Használata:

```
ones(x)
```

Egységmátrix

Az egységmátrix egy olyan mátrix, amelynek főátlója egyesekből áll, többi eleme 0-ból. Képzése:

`eye(x)`

ahol az x a mátrix sorát és oszlopát adja meg. Lehetséges még az

`eye(x, y)`

képzés, ahol az x a sorok számát, az y pedig az oszlopok számát jelenti, ezzel lehet “hamis” egységmátrixot képezni. Nézzünk ezekre példát!

```
Command Window
>> eye(5)

ans =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1

>> eye(4,5)

ans =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0

>> |
```

Helyes és hamis egységmátrix

Random szám és Random mátrix

A random szám generálása hasonlóan működik a Java-s random számhoz: 0 és 1 között generál egy lebegőpontos számot. Ehhez adhatunk hozzá vagy elvehetünk belőle, attól függ milyen intervallumot szeretnénk előállítani. Használata MatLab-ban:

`rand()`

Ez egy fent már definiált, véletlen számot fog visszaadni. Nézzük meg a gyakorlatban hogy működik ez:

~MatLab Bootcamp~

```
>> rand()

ans =

    0.8147
```

Random szám generálás

A `rand()` függvény abból a szempontból válik érdekessé, amikor rájövünk, hogy tökéletesen alkalmas mátrixok generálására is:

`rand(x)` = x-szer x-es négyzet mátrix véletlen számokkal

`rand(x, y)` = x-szer y-os mátrix véletlen számokkal.

Nézzük meg hogy fest ez a gyakorlatban:

```
>> rand(10)

ans =

    0.9058    0.9706    0.0357    0.0318    0.3816    0.6797    0.2551    0.2543    0.8308    0.0540
    0.1270    0.9572    0.8491    0.2769    0.7655    0.6551    0.5060    0.8143    0.5853    0.5308
    0.9134    0.4854    0.9340    0.0462    0.7952    0.1626    0.6991    0.2435    0.5497    0.7792
    0.6324    0.8003    0.6787    0.0971    0.1869    0.1190    0.8909    0.9293    0.9172    0.9340
    0.0975    0.1419    0.7577    0.8235    0.4898    0.4984    0.9593    0.3500    0.2858    0.1299
    0.2785    0.4218    0.7431    0.6948    0.4456    0.9597    0.5472    0.1966    0.7572    0.5688
    0.5469    0.9157    0.3922    0.3171    0.6463    0.3404    0.1386    0.2511    0.7537    0.4694
    0.9575    0.7922    0.6555    0.9502    0.7094    0.5853    0.1493    0.6160    0.3804    0.0119
    0.9649    0.9595    0.1712    0.0344    0.7547    0.2238    0.2575    0.4733    0.5678    0.3371
    0.1576    0.6557    0.7060    0.4387    0.2760    0.7513    0.8407    0.3517    0.0759    0.1622

>> rand(2,3)

ans =

    0.7943    0.5285    0.6020
    0.3112    0.1656    0.2630

>>
```

Random négyzetes és változó nagyságú mátrix generálás

Ne feledkezzünk meg a vektorokról! Hogy lehetne ezt vektoron használni?

```
v = rand(1,10)
```

Egyszerűen csinálunk egy 1 soros, 10 oszlopos mátrixot, és tudjuk: A vektor, az az egy soros mátrix!

Magic

Ez most sajnos nem a “kalapból nyúl előhúzás” mágia lesz, de hasonlóan érdekes. A magic függvény egy olyan n méretű négyzet mátrixot állít elő, amelynek minden eleme 1-től n^2 -ig megy. Ezt a legjobban egy példával érdemes szemléltetni:

```
Command Window
>> magic(5)

ans =

    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

>> |
```

Magic 5

A legszebb ebben az egészben, hogy minden szám csak egyszer fordul meg benne: 1-től 25-ig!

Műveletek mátrixon

Mátrix mérete

Egy mátrix méretét 2 komponens határozza meg: Sor és oszlop. Ezeket az információkat tudjuk kinyerni egy mátrixból a

`size(m)`

paranccsal, ahol m egy tetszőleges mátrixot jelent.

Nézzünk meg egy gyakorlati példát rá:

```
>> m = [1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15]

m =

     1     2     3     4     5
     6     7     8     9    10
    11    12    13    14    15

>> size(m)

ans =

     3     5
```

Sor és oszlop lekérdezése = méret

Legnagyobb és legkisebb elem keresése

(Figyelem: Ez csak egy oszlopban számolja ki a legkisebb és legnagyobb elemeket. Kombinálni kell külön függvényeket ha az egész mátrixban szeretnénk megtalálni például a legkisebb elemet. Erre javaslat: Kérdezzük le az oszlop minimumokat a `min(m)` -el, majd rendezzünk és aztán az első elem lesz a legkisebb.)

A legnagyobb és a legkisebb elemeket a következő 2 paranccsal tudjuk lekérdezni egy mátrixból:

`min(m)`

`max(m)`

ahol `m` egy tetszőleges mátrix. Gyakorlatban:

```
m =

    10     9     8     7     6     5
     5     6     7     8     9    10

>> max(m)

ans =

    10     9     8     8     9    10

>> min(m)

ans =

     5     6     7     7     6     5

>>
```

Minden oszlop legkisebb és legnagyobb elemei

Legnagyobb dimenzió keresése

Egy mátrix legnagyobb dimenzióját a

`length(m)`

paranccsal kérhetjük le, ahol `m` egy tetszőleges mátrix.

Gyakorlatban:

```
m =  
  
    10     9     8     7     6     5  
     5     6     7     8     9    10  
  
>> length(m)  
  
ans =  
  
     6  
  
>> |
```

Legnagyobb dimenzió keresése

Sor és oszlop összeadása

A MatLab eszközt nyújt arra, hogy összeadjuk egy mátrix oszlopait és sorait. Mind a kettő műveletet ugyanaz a parancs hajtja végre, csak más paramétert kell megadni:

`sum(m)` vagy `sum(m,1)`

ahol `m` egy tetszőleges mátrix, egy sorvektort fog visszaadni, ami annyit tesz, hogy a közös oszlopban lévő elemeket adjuk össze.

A

`sum(m,2)`

pedig az egy sorban lévő elemeket adja össze, eredményezve ezzel egy oszlopvektort.

Gyakorlatban:

```
m =  
  
    10     9     8     7     6     5  
     5     6     7     8     9    10  
  
>> sum(m)  
  
ans =  
  
    15    15    15    15    15    15  
  
>> sum(m,2)  
  
ans =  
  
    45  
    45
```

Először a közös oszlopban lévő elemeket, aztán pedig a közös sorban lévő elemeket adjuk össze

Talán már rájöttetek, hogy ha egy `sum()`-ba beágyazunk egy másik `sum()` parancsot, akkor először az oszlop elemeket adjuk össze, ami egy sorvektort eredményez. Ezután a második összeadás a már kiszámolt sorvektor elemeit adja össze, ezzel 1 darab számot eredményezve:

```
>> sum(sum(m))  
  
ans =  
  
    90  
  
>>
```

Az előző példa után $45+45=90$

Mátrix négyzetre emelése

Mint egy sim számot, egy mátrixot is négyzetre tudunk emelni a szokásos $^$ jelöléssel, viszont van egy kikötésünk: A mátrixnak négyzetes mátrixnak kell lennie. Ha ezt nem tartjuk be, büszke tulajdonosai lehetünk egy MatLab piros exception üzenetnek ami megkér minket a négyzetes mátrix megadására.

A gyakorlatban ez így néz ki:

```
Command Window

>> m

m =

    10     9     8     7     6     5
     5     6     7     8     9    10

>> m^2
??? Error using ==> mpower
Matrix must be square.

>> m = [1 2; 3 4]

m =

     1     2
     3     4

>> m^2

ans =

     7    10
    15    22

>> |
```

Négyzetre emelés 2 módja

A fenti ábrán látható a 2 módszer amellyel lehet dolgozni: Ha négyzetre emelünk, megtehetjük azt a rendes szorzási módszerrel, vagy ha a `.`[^] operátort használjuk, akkor elemenként emelünk a négyzetre.

(A “rendes szorzási módszer” az órán tanult mátrix * mátrix módszert alkalmazza! Ez a tradicionális szorzás)

```
>> m.^2

ans =

     1     4
     9    16

>> |
```

Elemenkénti négyzetre emelés

Transzponálás

A transzponálás a főátló menti tükrözés, matematikában jele a $'$. Ez MatLab-ban sem változott meg, egy mátrix transzponáltját így kapjuk meg:

m'

Például:

```
>> m = [1 2 3; 4 5 6]

m =

     1     2     3
     4     5     6

>> m'

ans =

     1     4
     2     5
     3     6

>> |
```

Egy 2x3-as mátrix transzponálása. Láthatjuk, hogy 1 és 5 a főátló

Determináns

Egy mátrix determinánsát megint csak úgy jelöljük, mint a matematikában:

`det(m)`

ahol m egy négyzetes mátrix.

Élesben:

~MatLab Bootcamp~

```
>> m = [4 5 6 ; 2 3 1; 5 6 7]
```

```
m =
```

```
     4     5     6
     2     3     1
     5     6     7
```

```
>> det(m)
```

```
ans =
```

```
    -3
```

```
>> |
```

Determináns kiszámítása

Inverz

Egy mátrix inverzének kiszámítására a MatLab egy külön kulcsszót vezetett be:

`inv(m)`

ahol m egy négyzetes mátrix.

Élesben:

```
m =
```

```
     4     5     6
     2     3     1
     5     6     7
```

```
>> inv(m)
```

```
ans =
```

```
 -5.0000  -0.3333   4.3333
  3.0000   0.6667  -2.6667
  1.0000  -0.3333  -0.6667
```

Mátrix inverze

Mátrix rendezése

Egy mátrixot 2 féleképpen tudunk (beépített függvénnyel) rendezni. Maga a függvény neve a `sort(m)`, ahol `m` egy tetszőleges mátrix. Hasonlóan a `sum()` parancshoz, ennek is van egy paramétere ami oszlopra vagy sorra mutat:

`sort(m)` vagy `sort(m,1)`

az `m` mátrix oszlopait rendezi növekvő sorrendbe, míg a

`sort(m,2)`

az `m` mátrix sorait rendezi növekvő sorrendbe.. Nézzünk a 2 rendezésre példát!

```
m =  
  
     4     5     6  
     2     3     1  
     5     6     7
```

```
>> sort(m)
```

```
ans =  
  
     2     3     1  
     4     5     6  
     5     6     7
```

```
>> sort(m,2)
```

```
ans =  
  
     4     5     6  
     1     2     3  
     5     6     7
```

Először oszlop, majd sor rendezése növekvő sorrendbe

Törlés mátrixból

A törlés viszonylag érdekesen van megoldva a MatLab-ban. Nincs külön parancs rá, helyette azt fogjuk csinálni, hogy egy üres mátrixot teszünk a törölni kívánt elemek helyére. Például töröljük ki minden oszlopot a 3. sorból (ergo, a 3. sort kitöröljük):

```
m(3,:) = []
```

Már korábban tanultuk ez mit jelent, de azért még egyszer: Megmondjuk, hogy a 3. sor összes oszlopa legyen üres.

Nézzük meg élőben:

```
>> m

m =

     4     5     6
     2     3     1
     5     6     7

>> m(3,:) = []

m =

     4     5     6
     2     3     1
```

Egy sor törlése

Innentől kezdve törlés szintjén határ a csillagos ég, mivel mindent tudunk az elemek eléréséről és beállításáról. (A beállítás az szimpla értékadás, $m(3,2) = 10$ például).

Műveletek mátrixok között

Elemenkénti összeadás

2 mátrixot elemeenként a sima + jellel tudunk összeadni. Példa:

```
>> m1

m1 =

     1     2     3
     4     5     6
     7     8     9

>> m2

m2 =

     4     5     6
     1     2     3
     7     8     9

>> m1+m2

ans =

     5     7     9
     5     7     9
    14    16    18
```

Elemenkénti összeadás

Elemenkénti kivonás

2 mátrixot elemeenként a sima - jellel tudunk kivonni. Példa:

~MatLab Bootcamp~

```
>> m1

m1 =

     1     2     3
     4     5     6
     7     8     9

>> m2

m2 =

     4     5     6
     1     2     3
     7     8     9

>> m1-m2

ans =

    -3    -3    -3
     3     3     3
     0     0     0
```

Elemenkénti kivonás

Elemenkénti szorzás

2 mátrixot elemenként a `. *` jelekkel tudunk összeszorozni. Példa:

~MatLab Bootcamp~

```
>> m1

m1 =

     1     2     3
     4     5     6
     7     8     9

>> m2

m2 =

     4     5     6
     1     2     3
     7     8     9

>> m1.*m2

ans =

     4    10    18
     4    10    18
    49    64    81
```

Elemenkénti szorzás

Elemenkénti osztás

2 mátrixot elemenként a `./` jelekkel tudunk elosztani. Példa:

~MatLab Bootcamp~

```
>> m1

m1 =

     1     2     3
     4     5     6
     7     8     9

>> m2

m2 =

     4     5     6
     1     2     3
     7     8     9

>> m1./m2

ans =

    0.2500    0.4000    0.5000
    4.0000    2.5000    2.0000
    1.0000    1.0000    1.0000
```

Elemenkénti osztás

Kereszt operáció

MatLab-ban a kereszt operációra külön kulcsszó van bevezetve, ez a `cross(m1,m2)`

ahol `m1` és `m2` hosszának minimum 3-nak kell lennie (`length()`), és a hosszuknak meg kell egyeznie egymással.

Példa:

```
>> cross(m1,m2)

ans =

    21    24    27
    21    24    27
   -15   -21   -27
```

Korábban látott `m1` és `m2` mátrixok kereszt operációja

Pont operáció

A pont operáció lényegében arról szól, hogy a 2 mátrix közös sorszámú oszlopain végzünk egy elemenkénti szorzást, és a kapott szorzatokat összeadjuk. Maga az operációt a

`dot(m1,m2)`

végzi, ahol `m1` és `m2` megegyező sor és oszlop számmal rendelkeznek.

Az operációt magát legjobban egy példával lehet szemléltetni:

```
m1 =  
  
     1     2     3  
     4     5     6  
     7     8     9  
  
>> m2  
  
m2 =  
  
     4     5     6  
     1     2     3  
     7     8     9  
  
>> dot(m1,m2)  
  
ans =  
  
    57    84   117
```

Pont operáció

Nézzük például az 57 hogy jött ki!

$$(1*4) + (4*1) + (7*7) = 4 + 4 + 49 = 57$$

Esetleg MatLab nyelven:

$$m1(1,1) * m2(1,1) + m1(2,1) * m2(2,1) + m1(3,1) * m2(3,1)$$

És ezt megismétljük minden oszlopra.

Mátrix-Mátrix szorzás

Ez a sima, már órán tanult szorzás ahol az egyik mátrixot fentre, a másikat pedig bal oldalra írjuk, és a 2 mátrix között megjelenik a megoldás. Példa:

```
>> m1

m1 =

     1     2     3
     4     5     6
     7     8     9

>> m2

m2 =

     4     5     6
     1     2     3
     7     8     9

>> m1*m2

ans =

    27    33    39
    63    78    93
    99   123   147
```

Mátrix-mátrix szorzás

A 27 itt úgy jött ki, hogy $1*4 + 2*1 + 3*7 = 4 + 2 + 21 = 27$. Tehát helyes a Matlab számítása!

Haladó mátrix kezelés

Többdimenziós mátrix

A mátrixokat eddig csak 2 dimenzióban kezeltük, sőt, az éven nagy valószínűséggel csak így fogunk dolgozni velük, ám a MatLab képes 2-nél nagyobb dimenziók kezelésére is. Gondoljunk a több dimenzióra úgy, mint a 2 dimenziós továbbfejlesztett változatára. A 3. dimenziót szeretném megismertetni egyenlőre, ez igazából több 2 dimenziós mátrixot foglal egybe. Képzeljük a 2 dimenziós mátrixot fiókoknak, akkor a 3. dimenzió az maga az éjjeliszekrény.

Példának okául helyezzünk el egy 3 dimenziós mátrixban 2 darab 2 dimenziósat:

1. lépés, hozzuk létre a 2 mátrixot:

```
>> m1 = [1 2 3; 4 5 6; 7 8 9]

m1 =

     1     2     3
     4     5     6
     7     8     9

>> m2 = [10, 11, 12; 13 14 15; 16 17 18]

m2 =

    10    11    12
    13    14    15
    16    17    18
```

2. lépés, egy harmadik mátrixot hozunk létre amely 3 dimenziós, és már el is helyezzük benne az m1-et:

```
>> m(:, :, 1) = m1

m =

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

Fontos megjegyeznünk, hogy még most mivel csak 1 db 2 dimenziós mátrix tartozkodik az `m` mátrixban, ezért sima 2 dimenziósként is kezelendő.

3. lépés, adjuk hozzá a 2. mátrixot, és nézzük meg mi történik!

```
>> m(:, :, 2) = m2
```

```
m(:, :, 1) =
```

1	2	3
4	5	6
7	8	9

```
m(:, :, 2) =
```

10	11	12
13	14	15
16	17	18

```
>> |
```

És kész a 3 dimenziós mátrixunk! Ezt el lehet játszani n dimenzióval, a további játszadozást rátok bízom!

M-FILE

Az M-FILE-ok a mi alprogramjaink, függvényeink, amelyeket az életünk megkönnyebbítésére írunk. Na jó, nem teljesen, rengeteg mátrix operációra nincs külön definiálva függvény, mivel túl komplexek vagy egyszerűen nem érte meg a fejlesztői csapatnak egy olyan függvény megírása, ami igazából 2, már megírt függvény együttes munkája.

Akármilyen okból is szeretnénk programozni, a `.m` kiterjesztésű fájlok adnak erre megoldást a MatLab-ban. A nyelv amit használ Java és C közös gyermeke, megfűszerezve a saját MatLab-os mátrix és matematikai szintaktikájával. A következőkben utánajárunk, hogy mi fán terem egy M-FILE, milyen eszközöket nyújt a kényelmes programozás megsegítéséhez, és milyen logikát kell alkalmaznunk ennél a nyelvnél.

(Érdemes még megemlíteni a programozói filozófiát az M-FILE-ok körében: Az M-FILE fő célja az, hogy olyan kódot írjunk vele, amit amúgy a konzolba is lehetne, viszont így sokkal kompaktabb, és itt használhatunk ciklusokat, egyéb eszközöket is. Még egyszerűbben, az M-FILE az egy csomó konzol parancs becsomagolva egy programba.)

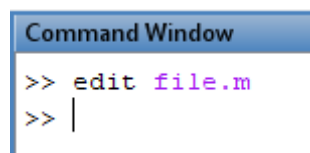
Létrehozás

Egy M-FILE létrehozását számos módon meg lehet tenni. Aki a konzolt kedveli, az beírhatja, hogy

`edit` vagy `edit fájlnev.m.`

Ezen kívül akár csak simán jobb klikk a Current Directory-n belül, `new > M-FILE`.

Ekkor elsőként kapunk egy üzenetet, hogy a file nem létezik, létrehozzuk-e (ez csak akkor ha elneveztük a második módszerrel). Elfogadás után a Current Directory ablakban megjelenik a saját M-FILE-ünk, és innen meg is nyithatjuk innentől kezdve. Ha a Windows-on belül szeretnétek megtalálni, hogy hol a munkátok, akkor a Current Directory mellett leírja az elérési útvonalat, ami valószínűleg a Dokumentumok mappa. A Current Directory ablakban kezelhetjük a file nevét, ki is törölhetjük onnan azt.



M-FILE születik

Futtatás

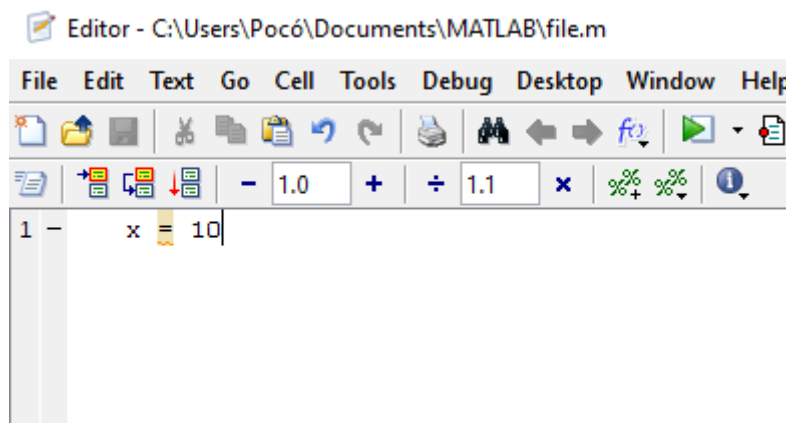
A futtatás két módon történhet, vagy beírjuk a konzolba a file nevét, vagy jobb klikkelünk a Current Directory-n belül a file-ra, és ott a Run opcióval elindítjuk.

Programozói eszközök

FIGYELEM: Ettől a ponttól kezdve egy `file.m` nevű programon fogok dolgozni, a tartalmát fogom kiírni, nem a konzolét. Ha ez máshogy lesz, azt feltüntetem külön szövegesen!

Értékadás

Értékadást az egyenlőségjellel tudjuk végrehajtani, ezt már láttatok fentebb is rengeteg példánál. Az M-FILE-on belül se tér el ez semmivel. Hozzunk létre egy `x` változót 10-es értékkel!



Értékadás

És most futtassuk le:

```
>> file  
  
x =  
  
    10  
  
>>
```

file.m futtatása

Megjegyzés: Azért írja ki az `x` értékét a `file`, mert nem tettem pontosvesszőt a parancs végére. Ezt jobb ha még most megtanuljuk, mert kiírni értéket M-FILE-ban érdekesebb az erre definiált függvénynel, és mindent le tudunk zárni így pontosvesszővel.

MatLab operátorok

A MatLab is definiálja a szokásos logikai kapukat, amelyeket már jól ismerünk. Nézzük meg a szintaktikákat, egy kicsit eltér némelyik!

AND : &&

OR : ||

NOT : ~=

Relációs : > < >= <= ==

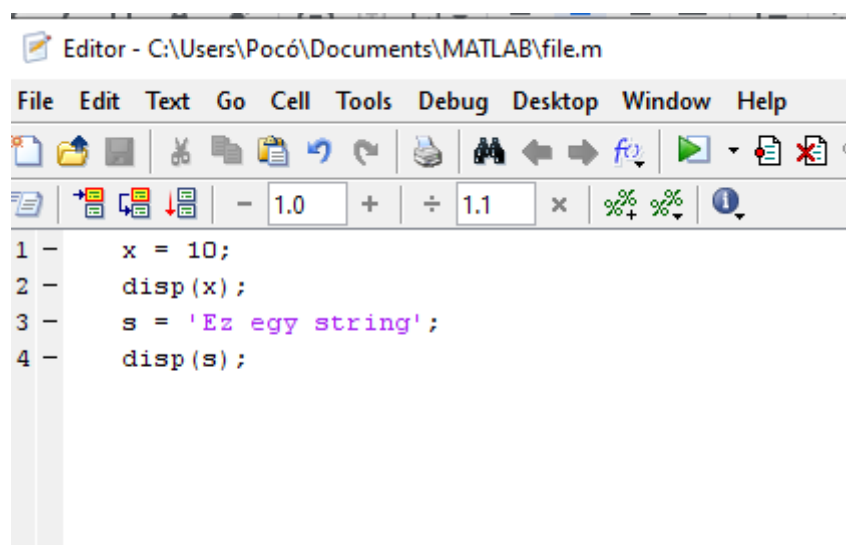
Kiírás fajtái és alkalmazásai

M-FILE-ban kiírásról már korábban volt szó, ahol azt taglaltuk, hogy ki lehet íratni egy változót csak a változó leírásával, ám sokszor szöveget is szeretnénk társítani a szám kiírása mellett. Ilyenkor jön segítségünkre két függvény, amelyek célja a bonyolultabb kiíratás.

Első körben a

`disp(x);`

függvény, ahol `x` egy változó. Na most ez CSAK változók kiírását támogatja, ám be lehet csapni! Ha egy változónak szöveget adok meg, ergo string típusú változó, akkor a `disp(x)` simán működni fog és kiírja a szövegünket, mivel kiírja a változó értékét, ami jelen esetben szöveg.



`disp()` tesztelése


```
>> file
    10

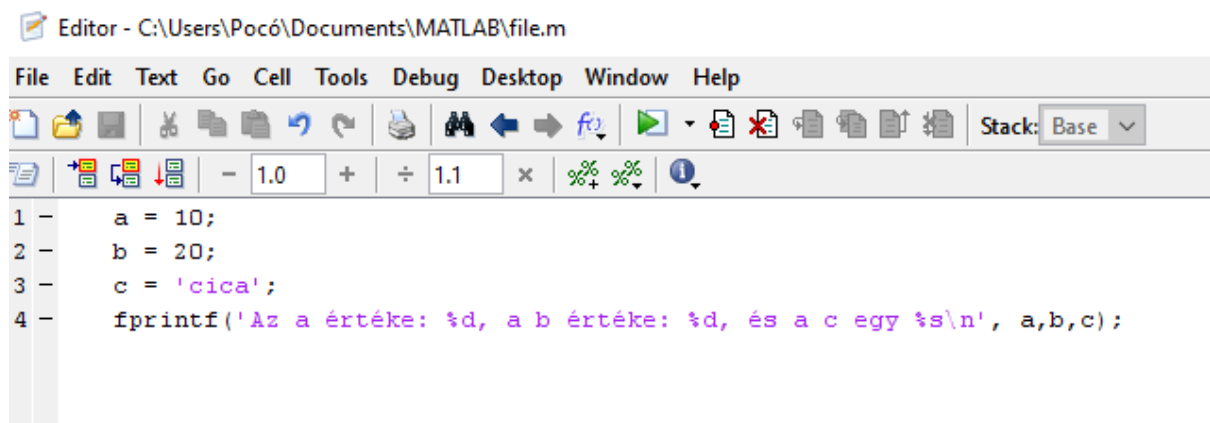
Ez egy string
>>
```

A változók kiíródtak

Ennél kicsit okosabb találmány a C-re hajazó

`fprintf()`;

ami alapjában argumentumként kér egy string-et, amelyben leírjuk szövegünket, és hogy milyen adatokat akarunk kiírni, ezután sorrendnek megfelelően megadjuk a szöveg után a kiírandó változókat és adatokat. Például:



Megadjuk az `fprintf()`-nek a megfelelő paramétereket, `%d` az szám, `%s` szöveg

```
>> file
Az a értéke: 10, a b értéke: 20, és a c egy cica
>> |
```

És a kiíratás értéke

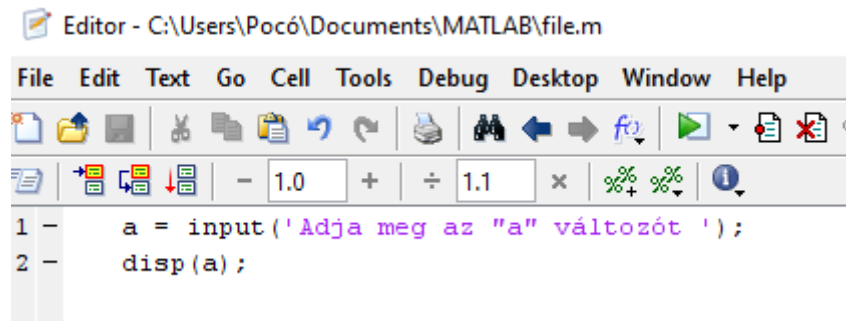
Fontos megjegyzés: Az `fprintf()` nem tesz sortörést a string végére, nekünk kell külön megtenni ezt egy `\n` karakterrel!

Információ bekérés a felhasználótól

Input kérésére a nagyon találó (és egyszerű)

`input(text)`

parancs használandó. A `text` az az a szöveg, amit az `input` a változó bekérése előtt ír ki, mint üzenet a felhasználó felé. Nézzük meg példában!



Bekérünk egy számot a felhasználótól

```
>> file
Adja meg az "a" változót 10
10

>>
```

Élesben futás

Feltétel-képzés

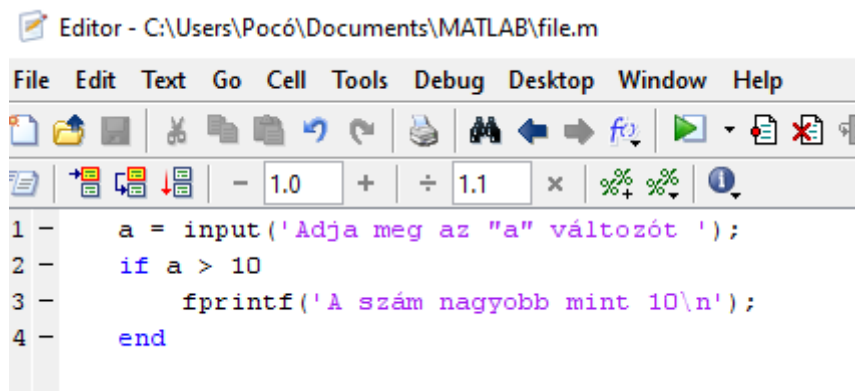
Feltételt a már jól ismert **IF...ELSE** fogja alkotni, ám bejön egy új elem amivel eddig nem foglalkoztunk, az **END**. Az **END** alapvetően azt jelzi, hogy egy kifejezésnek vége szakad. Az editor azt is jelzi, hogy egy **END** mihez kötődik, így nem veszünk el az **END**-ek számolásában. Az **IF** szerkezete így néz ki:

IF Feltétel

csinál valamit

END

Jól látjátok, nincs zárójel, de nem fog bántani a MatLab ha kiteszed mint azt szoktuk a többi programnyelvben. Nézzünk egy példát hogy hogyan is néz ki ez élőben:

The screenshot shows the MATLAB Editor interface. The title bar reads 'Editor - C:\Users\Pocó\Documents\MATLAB\file.m'. The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, and Help. The toolbar contains various icons for file operations, editing, and execution. Below the toolbar is a numeric keypad with buttons for minus, 1.0, plus, divide, 1.1, multiply, and percentage. The main editor area displays the following code:

```
1 - a = input('Adja meg az "a" változót ');
2 - if a > 10
3 -     fprintf('A szám nagyobb mint 10\n');
4 - end
```

If szerkezet

```
>> file
Adja meg az "a" változót 10
>> file
Adja meg az "a" változót 11
A szám nagyobb mint 10
>> |
```

A program működése élesben

Ez rendben is van, most próbáljuk meg implementálni az ELSE ágat. Az IF szerkezetünk most már így fog kinézni:

IF Feltétel

csinál valamit

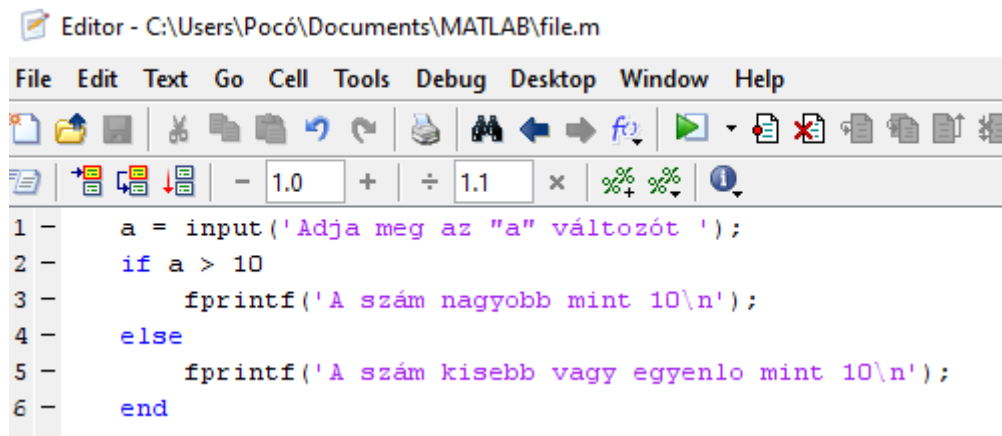
ELSE

csinál valamit

END

Módosítsuk az előbbi példánkat ELSE-re!

~MatLab Bootcamp~



The image shows a MATLAB Editor window titled 'Editor - C:\Users\Pocó\Documents\MATLAB\file.m'. The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, and Help. The toolbar contains various icons for file operations, editing, and execution. Below the toolbar is a numeric keypad with buttons for digits 1-9, 0, and mathematical operators like +, -, *, /, and %%. The main editor area contains the following MATLAB code:

```
1 - a = input('Adja meg az "a" változót ');
2 - if a > 10
3 -     fprintf('A szám nagyobb mint 10\n');
4 - else
5 -     fprintf('A szám kisebb vagy egyenlo mint 10\n');
6 - end
```

Else implementálása

```
>> file
Adja meg az "a" változót 10
A szám kisebb vagy egyenlo mint 10
>> file
Adja meg az "a" változót 11
A szám nagyobb mint 10
>>
```

Output

Még van egy utolsó elemünk amit hiányolunk, ez pedig az **ELSE** feltétellel együtt: **ELSEIF**. Ez módosítja a teljes **IF** szerkezetünket erre:

IF Feltétel

csinál valamit

ELSEIF Feltétel

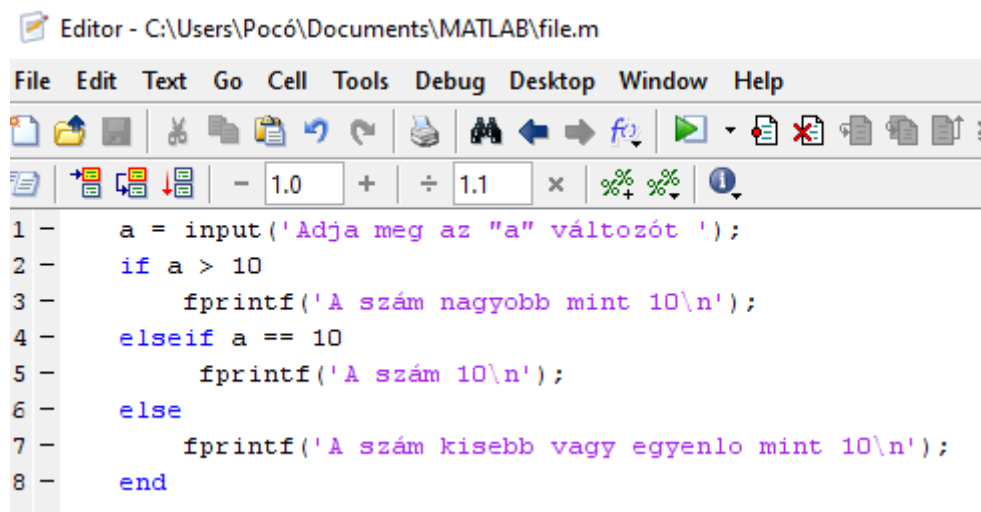
csinál valamit

ELSE

csinál valamit

END

Írjuk meg a programunkat **ELSEIF** használatával!



The image shows a screenshot of the MATLAB Editor window. The title bar reads 'Editor - C:\Users\Pocó\Documents\MATLAB\file.m'. The menu bar includes File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, and Help. The toolbar contains various icons for file operations, editing, and execution. The code editor shows the following MATLAB code:

```
1 - a = input('Adja meg az "a" változót ');
2 - if a > 10
3 -     fprintf('A szám nagyobb mint 10\n');
4 - elseif a == 10
5 -     fprintf('A szám 10\n');
6 - else
7 -     fprintf('A szám kisebb vagy egyenlo mint 10\n');
8 - end
```

Elseif implementálása

```
>> file
Adja meg az "a" változót 9
A szám kisebb vagy egyenlo mint 10
>> file
Adja meg az "a" változót 10
A szám 10
>> file
Adja meg az "a" változót 11
A szám nagyobb mint 10
```

Output

Ciklus-képzés

Ciklusokból pontosan 2 fajtát fogunk megnézni, ezek a **FOR** és a **WHILE** ciklusok lesznek. Szintaktikai eltérés csak a **FOR**-nál lesz. Nem ok nélkül hagytam ki a hátultesztelő ciklusokat, mivel a MatLab-ban olyan nincs, saját magunknak kell eszkábálni egyet (ezt amúgy egy **IF**-el lehet amit a **WHILE** feltételébe ágyazunk bele, de nem megyek bele részletesen). Nézzük is meg, miből is épül fel egy **FOR** ciklus:

FOR index = mettől : (esetleges lépés módosító, amúgy 1) : meddig
 csinál valamit
END

Lehet ez így még nem teljesen világos, implementáljuk előben, ebből majd kiderül minden!



For ciklus implementálása

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

Output

Lényegében írtunk egy ciklust, ami 10-szer fut le (a ciklus változó 1-től 10-ig megy, 1-es lépésekkel), A cikluson belül pedig az x változót inkrementálom 1-el és iratom ki a konzolra. A lépés szám meghatározása hasonló logika alapján megy, mint a vektor generálásnál tanult kettőspont operátor használata. Ha mondjuk 2-esével akarunk lépkedni 1-től 10-ig, akkor $1:2:10$ a helyes!

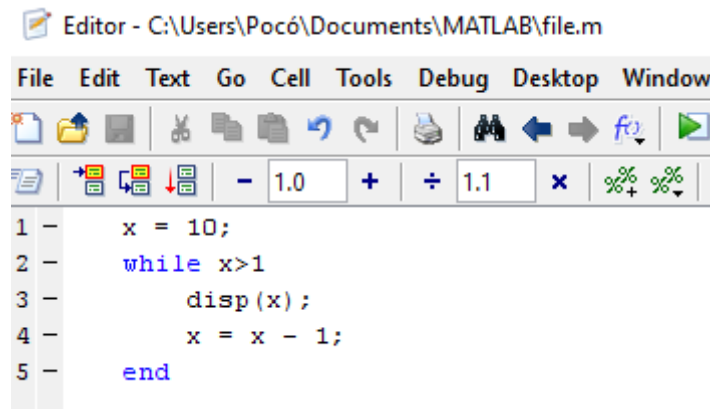
A másik ciklus amit meg kell néznünk, az a **WHILE** ciklus. Ennek a szintaktikája megegyezik az eddig tanult nyelvekével:

WHILE feltétel

csinál valamit

END

Nézzük meg gyakorlatban hogy működik!



The image shows a MATLAB Editor window titled 'Editor - C:\Users\Pocó\Documents\MATLAB\file.m'. The window contains a MATLAB script with the following code:

```
1 - x = 10;  
2 - while x>1  
3 -     disp(x);  
4 -     x = x - 1;  
5 - end
```

While ciklus implementálása

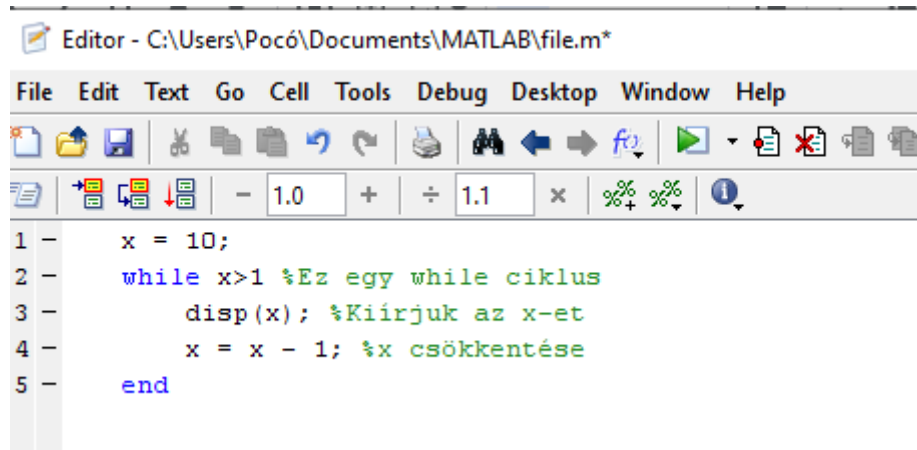
```
>> file  
10  
  
9  
  
8  
  
7  
  
6  
  
5  
  
4  
  
3  
  
2
```

A ciklus addig ment, amíg x nagyobb volt mint 1

Figyeljük meg, hogy a zárójel most is elhagyható a feltételnél!

Kommentálás

Mint minden nyelvben, itt is lehetőség nyílik a kommentálásra, itt a kulcs karakter a százalékjel (%).



Kommentelés

Függvény Írása

Mint azt már tanultuk más tárgyakból, a függvény az egy olyan metódus amelynek van bemeneti és kimeneti értéke is. Ez itt sincs máshogy, akkor használjuk ha fixen ki szeretnénk számolni valamit, változó paraméterekkel, és várunk vissza valamilyen választ a függvénytől. Nézzük meg, milyen egységekből áll egy MatLab függvény:

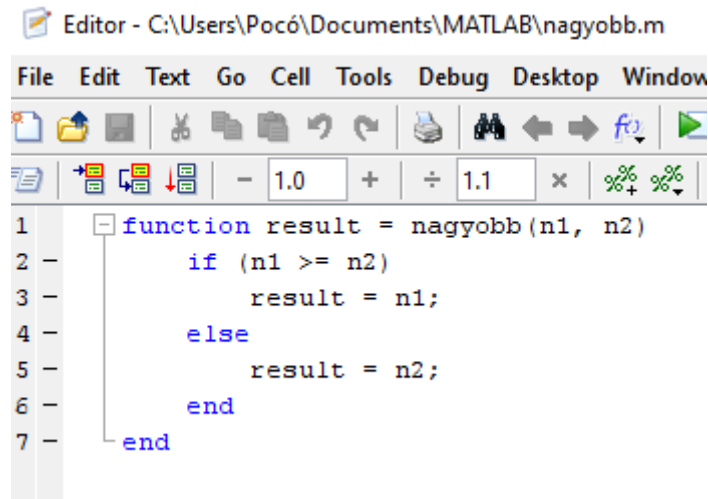
FUNCTION visszatérési-változó = függvény neve(param1, param2 ...)
munka a paraméterekkel

END visszatér a függvény a visszatérési-változóval

A belső szerkezeten kívül még van egy tennivalónk, még pedig átnevezni magát az M-FILE-t amiben dolgozunk. Meg KELL egyeznie a függvény nevének az M-FILE névével. Logikus, mivel mikor metódust írtunk akkor is úgy hívtuk meg magát a programunkat, hogy beírtuk a file nevét a konzolba.

Írjunk egy függvényt amely megmondja melyik szám nagyobb 2 közül!

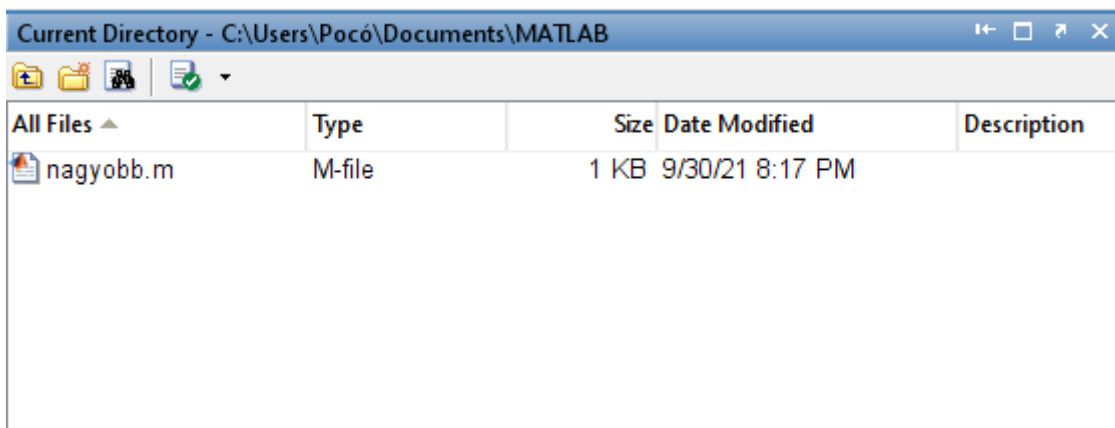
~MatLab Bootcamp~



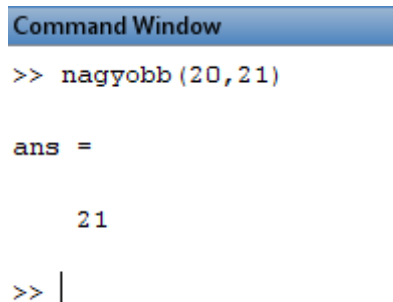
The image shows the MATLAB Editor window with the file 'nagyobb.m' open. The code defines a function 'result = nagyobb(n1, n2)' that returns the maximum of two numbers. It uses an 'if' statement to check if 'n1' is greater than or equal to 'n2'. If true, it sets 'result' to 'n1'; otherwise, it sets 'result' to 'n2'. The function ends with 'end'.

```
1 function result = nagyobb(n1, n2)
2     if (n1 >= n2)
3         result = n1;
4     else
5         result = n2;
6     end
7 end
```

Függvény implementáció



Az M-FILE-t is átneveztük



The image shows the MATLAB Command Window with the following commands and output:

```
>> nagyobb(20, 21)

ans =

    21

>> |
```

Függvény használata és az output

Érdemes megjegyezni, hogy a függvény automatikusan kiírja a visszatérési értékét, nem kellett semmilyen kijelző metódust alkalmaznunk. Ha a függvényt egy másik program alkalmazza, akkor az abban a programban fog visszatérni, ahonnan meghívták, és nem a konzolra. Jó példa erre a GitHub-ra feltöltött Normák példa.

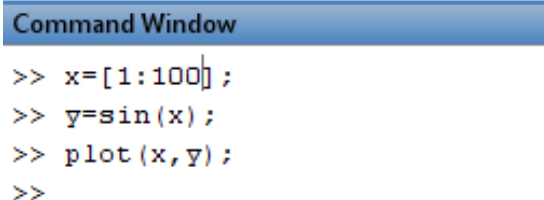
Grafika

Plot

A matematika egyik fontos ága a függvények, és ábrázolásuk. A `plot()` az utóbbit teszi lehetővé számunkra. Nagyon nem fogunk belelendülni a témába, de azért egy alkalmazói szintet megpróbálunk megütni most pár alapfogalommal és paranccsal ami a `Plot` családot alkotja a MatLab-ban.

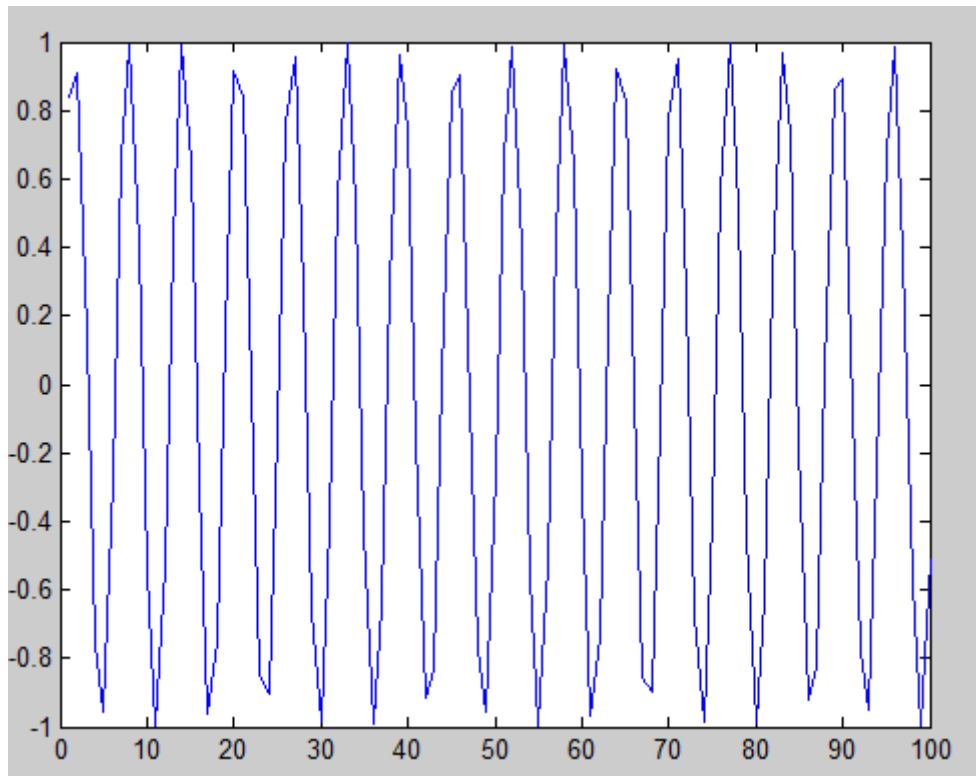
Létrehozás

Egy `plot()` létrehozásához 3 dolog szükséges: Egy `x` vektor, amellyel megmondjuk, mettől meddig terjed a függvény, egy `y` függvény, amit ábrázolni szeretnénk, és maga a `plot()` parancsba elültetni ezeket a paramétereket. Hozzunk létre egy `X=1` től `X=100`-ig terjedő szinusz függvényt!



```
Command Window
>> x=[1:100];
>> y=sin(x);
>> plot(x,y);
>>
```

A függvény létrehozása



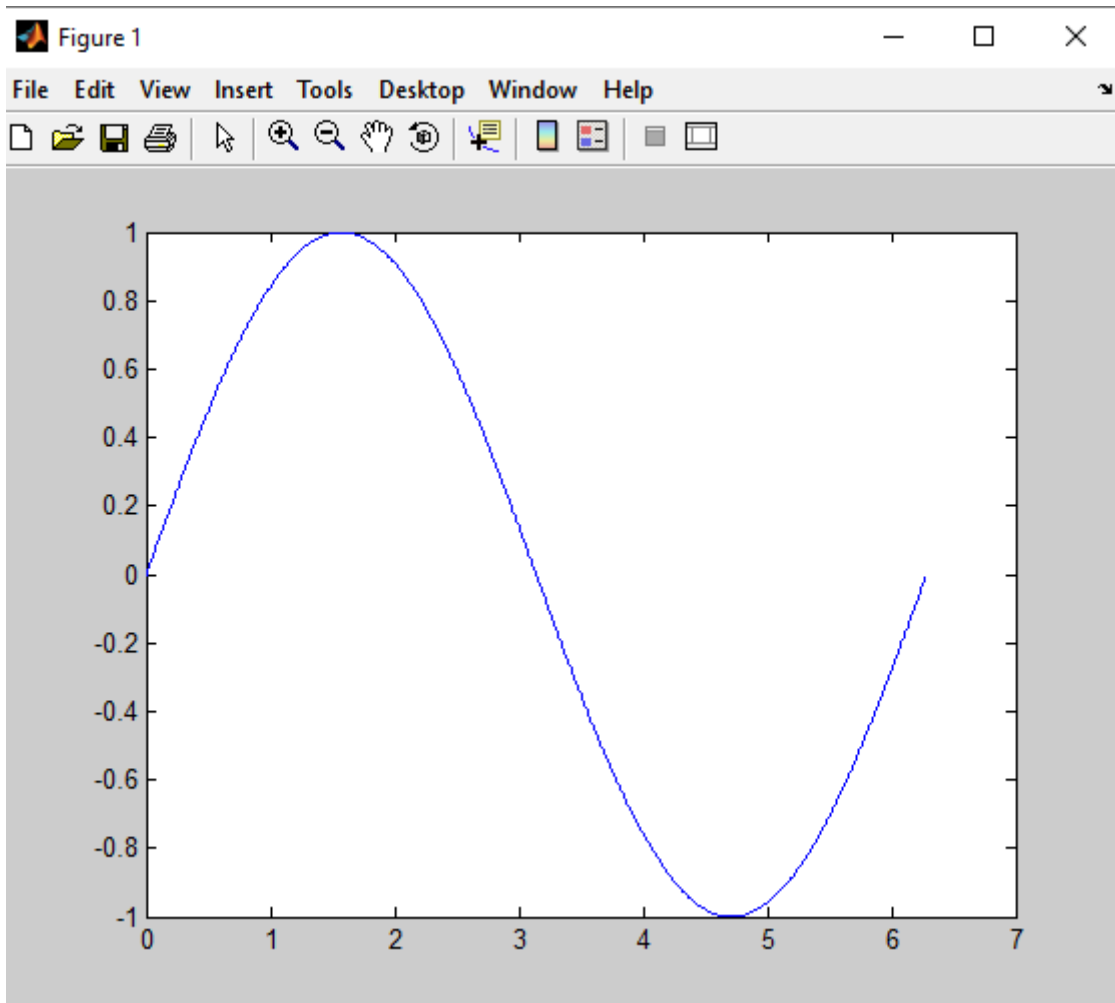
A függvény

Amint láthatjuk, azért érezzük hogy ez egy szinusz, de ugye 100-ig megy, elég töredezős, egyértelműen nincs elég X pont definiálva ahhoz, hogy ez “szép” íves szinusz legyen. Változtassuk meg úgy, hogy csak 0-tól 2π -ig menjen, és 0.01 X-enként haladjunk a 2π felé! (Természetesen ez egy rendes szinusz hullám lesz.)

Command Window

```
>> x=[0:0.01:2*pi];  
>> y=sin(x);  
>> plot(x,y)  
>>
```

Újradefiniált függvény klasszikus szinusz görbére



Az újdonsült szinusz görbe

Lényegében ez lenne a koordináta rendszer használata felé az első lépés. Jegyezzük meg a 3 alappillért: x terjedelem, y függvény, ezek alkalmazása `plot()`-al!

Címkézés és koordináta rendszer beállítás

A függvény megrajzolása még csak az első lépés, de mi van ha a felhasználó nem tudja mit lát? Mi az X koordináta? Mi az Y ? Természetesen ez triviális, de mégis fel szoktuk írni melyik tengely melyik, és ezt itt saját kezűleg kell megtennünk. Ez főleg akkor lesz hasznos, ha mást jelentenek a tengelyek, teszem azt idő / teljesítmény függvény.

Az X és Y koordináta címkézésére az

`xlabel()`

és az

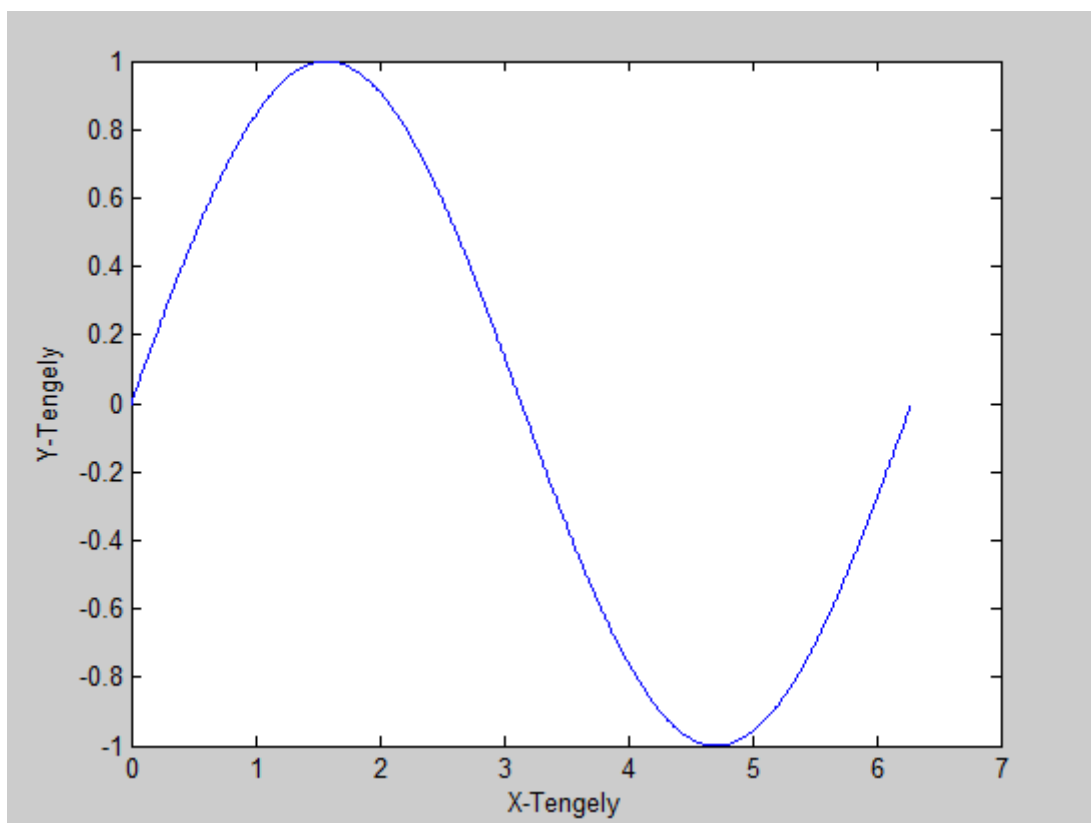
`ylabel()`

szolgál. Ezek szöveget fogadnak el 2 felsővonás között. Fontos megjegyeznünk, hogy ezek nem állandó paraméterek, hanem egy jelenleg definiált `plot()`-hoz fognak majd csatlakozni, és ebből az okból a `plot()` után kell őket megadni, vesszővel elválasztva egymást. Ez minden személyre szabási szempontra teljesül.

Címkézzük fel az X és Y tengelyt a nevükkel!

```
>> plot(x,y), xlabel('X-Tengely'), ylabel('Y-Tengely')  
>> |
```

Függvény felcímkézés



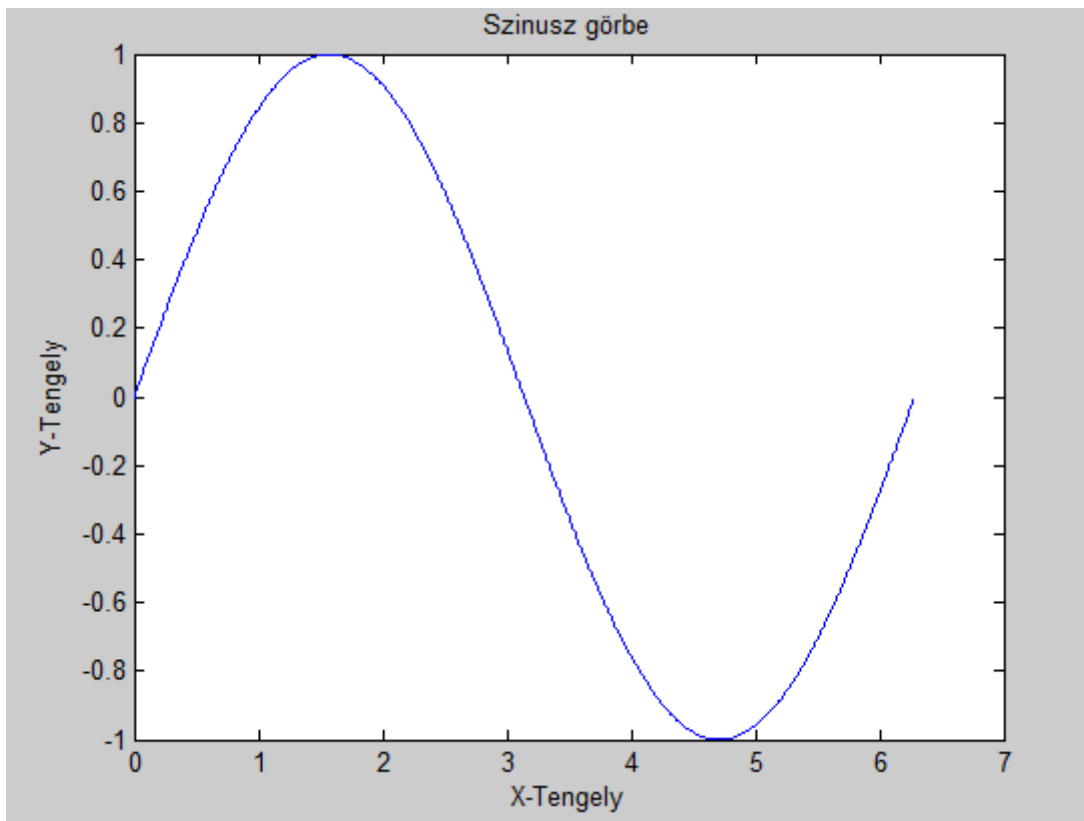
Maga a függvény most így néz ki

A függvénynek nevet is adhatunk, ezt a `title()` paranccsal tehetjük meg! Adjuk meg a függvény nevét!

~MatLab Bootcamp~

```
>> plot(x,y), xlabel('X-Tengely'), ylabel('Y-Tengely'), title('Szinusz görbe')  
>>
```

Az előző címkézés kiegészítése

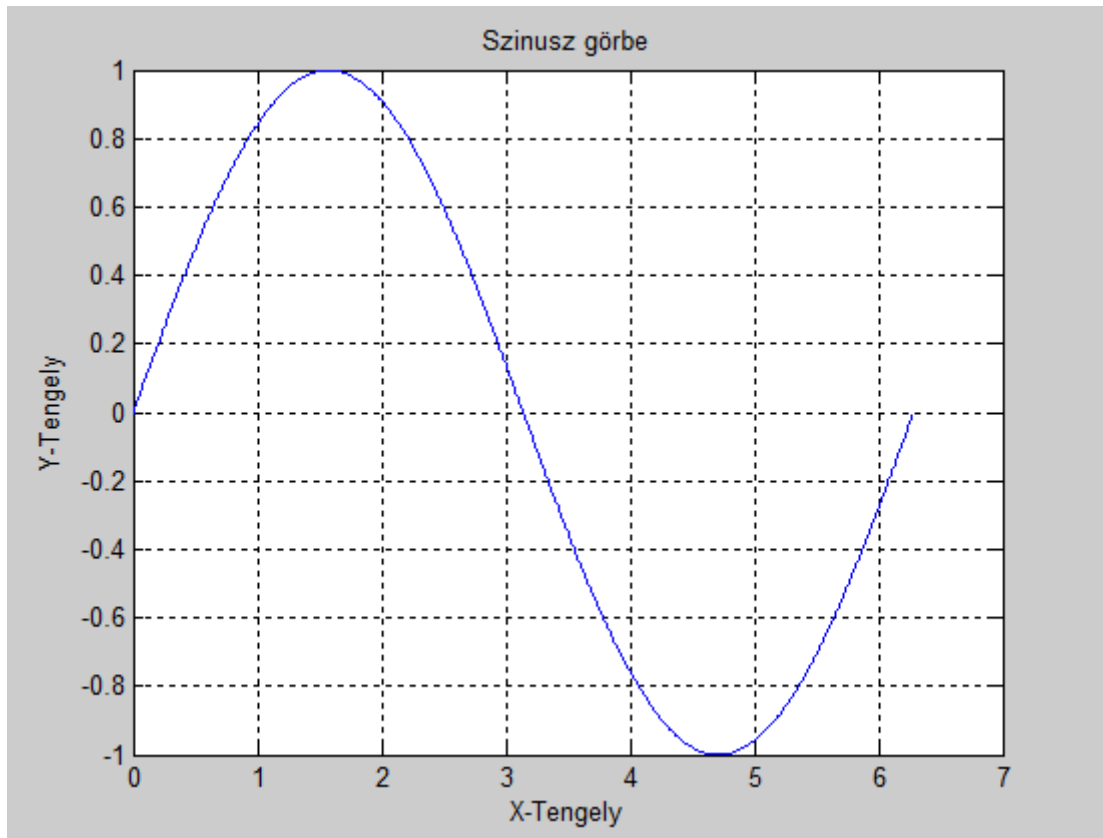


Most már címe is van a függvénynek

Most már felcímkeztünk mindent, viszont nagyon nehéz megmondani, hol van például pontosan a (3,0.2) pont. Ez azért van, mert nincs bevonalazva a koordináta rendszer. Ezt a `grid on` paranccsal tudjuk elérni.

```
>> plot(x,y), xlabel('X-Tengely'), ...  
    ylabel('Y-Tengely'), ...  
    title('Szinusz görbe'), ...  
    grid on  
>>
```

Grid on, és használtuk a ... operátort a hosszú sor miatt!



Bevonalmazott rendszer

Egyenlőre kész a szép szinusz koordináta rendszerünk, ám mi van akkor, ha több függvényt szeretnénk egyszerre mutatni?

Több-függvényes plot

Ha több függvényt szeretnénk egyszerre ábrázolni, akkor azt egy darab `plot()` parancsban kell megint csak megadnunk. Ennek a szerkezete kicsit bonyolultabb, nézzük meg!

```
plot(x1, y1, x2, y2, 'stílus')
```

Ahol:

x1 = Első függvény X tengelyes terjedelme

y1 = Első függvény

x2 = Második függvény X tengelyes terjedelme

y2 = Második függvény

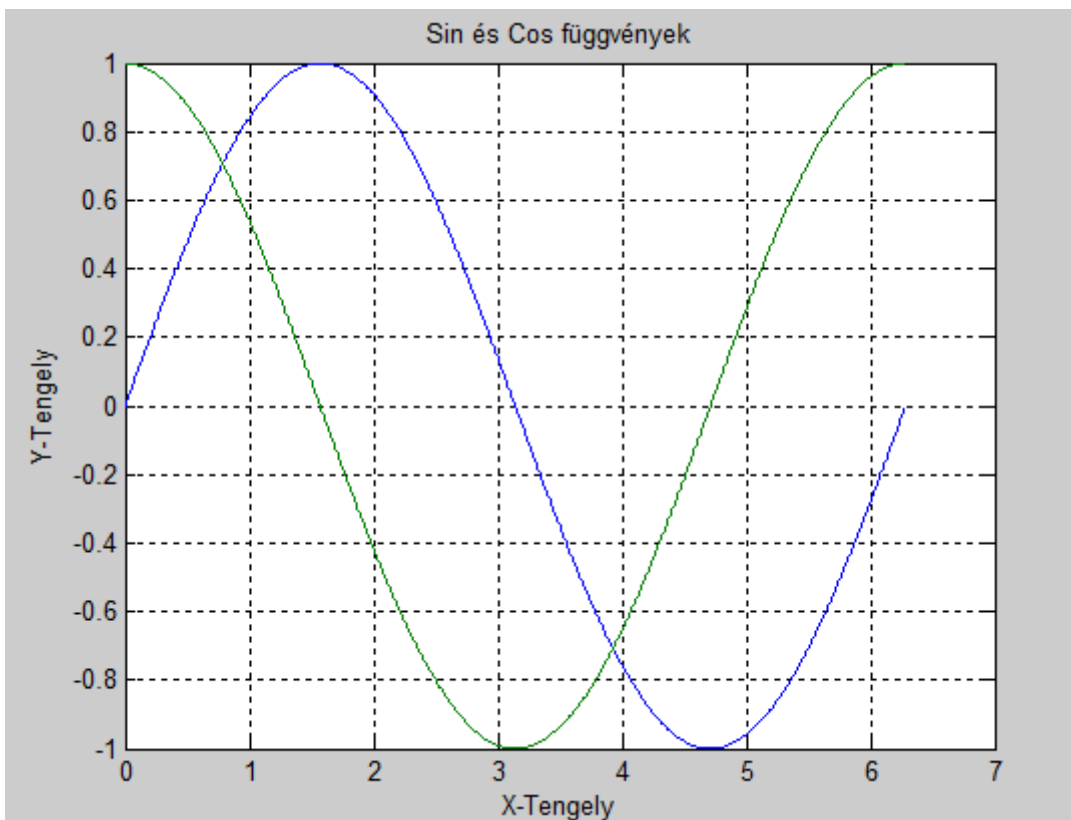
Stílus = Ez egy karakter kombináció, amivel jellemezzük, milyen vonalat húzzon a második függvény. (- például vessző és ponttal

szaggatott, vagy -- ha szaggatott vonalat szeretnénk, és - ha folytonosat).

Adjunk meg a szinusz függvényünk mellé egy koszinusz függvényt!

```
Command Window
>> y2 = cos(x);
>> plot(x,y1,x,y2,'-'), ...
xlabel('X-Tengely'), ylabel('Y-Tengely'), ...
grid on, ...
title('Sin és Cos függvények')
>>
```

Koszinusz függvény hozzáadása. Most ugyanazt az X terjedelmet használjuk.

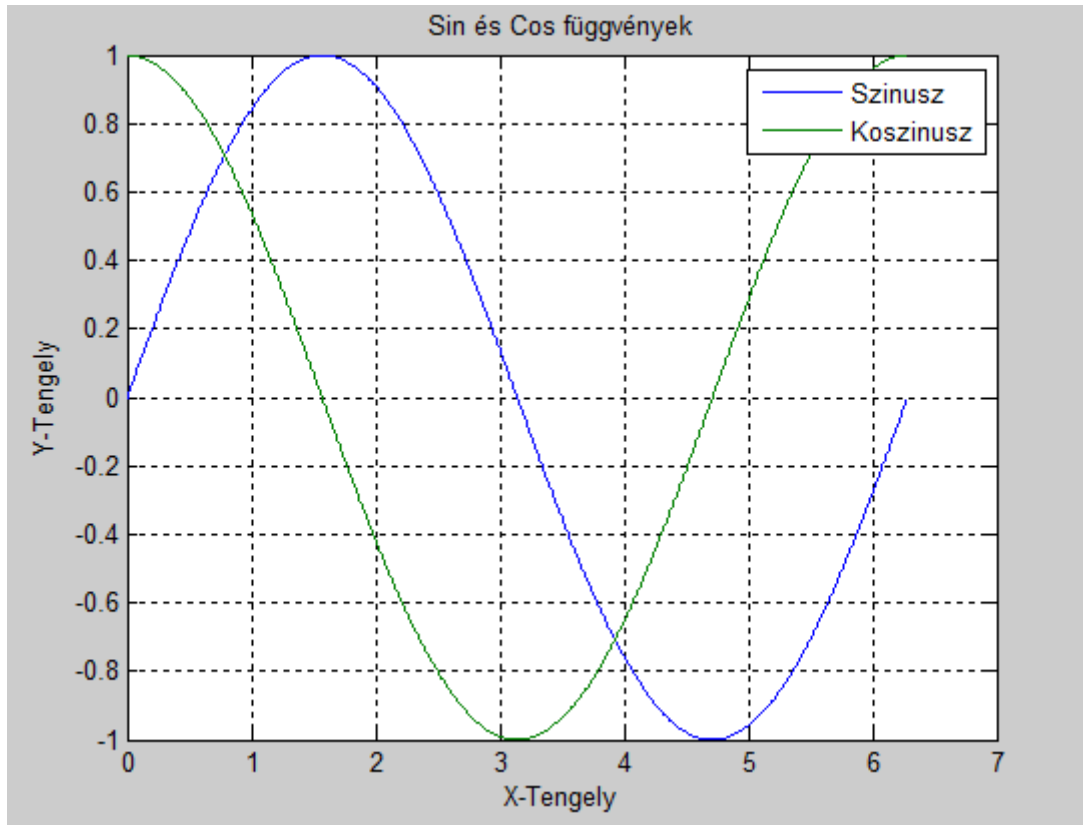


Koszinusz és Szinusz függvény

Most Már van mindenünk, de mégis egy dolog még hiányzik. Ez pedig a függvény magyarázat; melyik vonal melyik függvényt jelzi? Ezt a `legend()` függvényben lehet megadni, egymás után ahogy a függvényeket definiáltuk, abban a sorrendben kell nevet is adnunk nekik.


```
>> plot(x,y1,x,y2,'-'), ...  
xlabel('X-Tengely'), ylabel('Y-Tengely'), ...  
grid on, ...  
title('Sin és Cos függvények'), ...  
legend('Szinusz', 'Koszinusz')  
>>
```

Legend használata



Egy teljesen felszerelt függvény

Egyéb plot beállítás

Lehetséges megadnunk, hogy mettől meddig terjedjen a koordináta rendszer. Ezt az `axis()` paranccsal tehetjük meg, amely kér X-hez és Y-hoz is egy minimum és egy maximum értéket, és ez alapján fogja kiszabni a koordináta rendszert.

Az `axis()` működése:

```
axis([xmin xmax ymin ymax])
```

Ahol

`xmin` = legkisebb X koordináta,

`xmax` = legnagyobb X koordináta

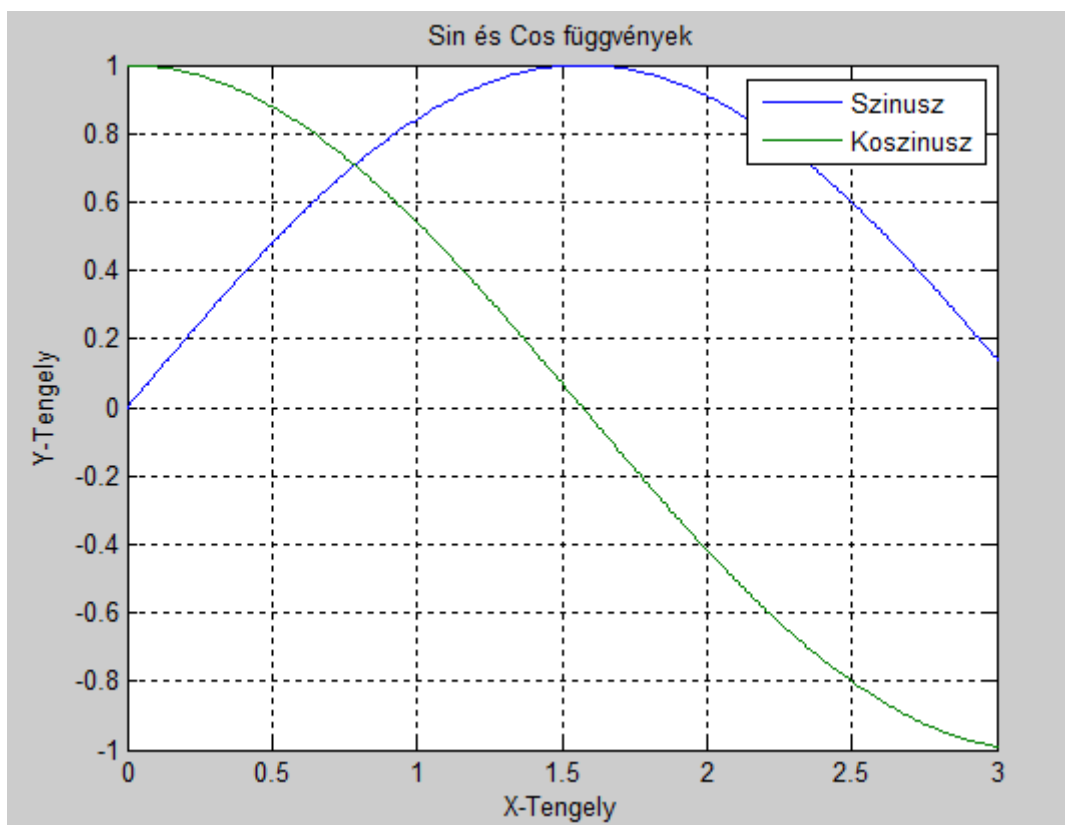
`ymin` = legkisebb Y koordináta,

`ymax` = legnagyobb Y koordináta

Nézzük meg a függvényünk X=0-tól 3-ig tartó szakaszát!

```
>> plot(x,y1,x,y2,'-'), ...  
xlabel('X-Tengely'), ylabel('Y-Tengely'), ...  
grid on, ...  
title('Sin és Cos függvények'), ...  
legend('Szinusz', 'Koszinusz'), ...  
axis([0 3 -1 1])  
>> |
```

Az X és Y tengelyt lelimitáltuk



A lelimitált ábra