

# Bevezetés a MATLAB programozásba

Ekart Csaba 2017

## Tartalom

|                                                                                                            |           |
|------------------------------------------------------------------------------------------------------------|-----------|
| <b>Bevezetés.....</b>                                                                                      | <b>3</b>  |
| <b>1. gyakorlat - Adminisztráció, kalkulátor, beépített függvények, saját szkript és függvény írása ..</b> | <b>4</b>  |
| 1.1. Mi a MATLAB? Mire jó? .....                                                                           | 4         |
| 1.2. A MATLAB környezet.....                                                                               | 4         |
| 1.3. Röviden a MATLAB szkriptekről .....                                                                   | 5         |
| 1.4. Röviden a MATLAB függvényekről .....                                                                  | 5         |
| 1.5. A legfontosabb alapparancsok .....                                                                    | 6         |
| <b>2. gyakorlat - Vektorok, 2D ábrázolás .....</b>                                                         | <b>7</b>  |
| 2.1. Sor- és oszlopvektor létrehozása.....                                                                 | 7         |
| 2.2. Transzponált .....                                                                                    | 7         |
| 2.3. Összefűzés .....                                                                                      | 7         |
| 2.4. Indexelés:.....                                                                                       | 7         |
| 2.5. Csupa egy és nulla tömbök.....                                                                        | 7         |
| 2.6. Random generálás.....                                                                                 | 7         |
| 2.7. Egységmátrix generálás .....                                                                          | 7         |
| 2.8. Diagonális mátrix generálás: .....                                                                    | 7         |
| 2.9. Lineáris vagy logaritmus beosztású vektor generálása: .....                                           | 8         |
| 2.10. Függvények.....                                                                                      | 8         |
| 2.11. Felosztás .....                                                                                      | 8         |
| 2.12. Összeg, átlag, szorzat.....                                                                          | 8         |
| 2.13. Maximum és minimum elem.....                                                                         | 8         |
| 2.14. Logikai indexelés, megkeresés.....                                                                   | 8         |
| 2.15. 2D ábrázolás .....                                                                                   | 9         |
| <b>3. gyakorlat - Formázott kiírás, mátrixok, vezérlő szerkezetek.....</b>                                 | <b>12</b> |
| 3.1. Formázott kiírás.....                                                                                 | 12        |
| 3.2. Stringes átalakítások (szám, mátrix) :-) .....                                                        | 12        |
| 3.3. Mátrixos alapműveletek.....                                                                           | 12        |
| 3.4. Vezérlő szerkezetek - elágazások .....                                                                | 12        |
| 3.5. Vezérlő szerkezetek - ciklusok.....                                                                   | 13        |
| 3.6. Vezérlő szerkezetek - switch .....                                                                    | 13        |
| <b>4. gyakorlat - Lineáris egyenletrendszerek, leképezések.....</b>                                        | <b>14</b> |
| 4.1. Egyenletrendszer megoldása: .....                                                                     | 14        |

|           |                                                                              |           |
|-----------|------------------------------------------------------------------------------|-----------|
| 4.2.      | Egy konkrét példa lineáris egyenletrendszer megoldására.....                 | 14        |
| <b>5.</b> | <b>gyakorlat - Polinomok, deriválás, integrálás, anonim függvények .....</b> | <b>15</b> |
| 5.1.      | Polinomok.....                                                               | 15        |
| 5.2.      | Numerikus deriválás.....                                                     | 15        |
| 5.3.      | Numerikus integrálás.....                                                    | 16        |
| 5.3.1.    | Trapéz szabály .....                                                         | 16        |
| 5.3.2.    | Spin-off .....                                                               | 17        |
| 5.3.3.    | Függvényes integrálás .....                                                  | 17        |
| <b>6.</b> | <b>gyakorlat - Differenciálegyenletek .....</b>                              | <b>18</b> |
| 6.1.      | Differenciálegyenletek .....                                                 | 18        |
| 6.2.      | Szemléletes megközelítés, gyakorlati példa.....                              | 18        |
| 6.3.      | Egy konkrét feladat megoldása .....                                          | 19        |
| <b>7.</b> | <b>gyakorlat - 3D ábrázolás .....</b>                                        | <b>20</b> |
| 7.1.      | 3D ábrázolás .....                                                           | 20        |
| 7.2.      | Pontfelhők .....                                                             | 20        |
| 7.3.      | Felületek ábrázolása.....                                                    | 21        |
| 7.4.      | Szintvonalak ábrázolása .....                                                | 22        |
| 7.5.      | Nézet beállítódása .....                                                     | 23        |
| <b>8.</b> | <b>gyakorlat – Cellatömbök, struktúrák, fájlműveletek .....</b>              | <b>24</b> |
| 8.1.      | Cellatömbök .....                                                            | 24        |
| 8.2.      | Struktúrák.....                                                              | 24        |
| 8.3.      | Fájlkezelés .....                                                            | 25        |
| 8.4.      | Formázott szöveg olvasása és írása.....                                      | 25        |
| 8.5.      | Bináris fájlok olvasása és írása.....                                        | 25        |
| 8.6.      | Fájlok mentése és betöltése.....                                             | 26        |
| <b>9.</b> | <b>gyakorlat – Táblázatok, fájlműveletek, képmentés.....</b>                 | <b>27</b> |
| 9.1.      | Táblázatok alapjai és létrehozásuk.....                                      | 27        |
| 9.2.      | Táblázatok összefűzése, alpműveletek.....                                    | 27        |
| 9.3.      | Fájlba írás és olvasás táblázatként .....                                    | 28        |
| 9.4.      | Grafikon fájlba mentése .....                                                | 28        |

## Bevezetés

Ez a jegyzet a második MATLAB ZH felkészülését hívatott megkönnyíteni, illetve egy teljes összefoglalót ad az egész féléves anyagból. A benne található anyag az órákon elhangzottak, a gyakorlatok diáiból, az internet mély bugyraiból és a laborokon kiadott feladatok alapján készült el. Remélhetőleg mindenkinek segíti a tárgyból való felkészülést. Sok sikert a második ZH-hoz!

# 1. gyakorlat - Adminisztráció, kalkulátor, beépített függvények, saját szkript és függvény írása

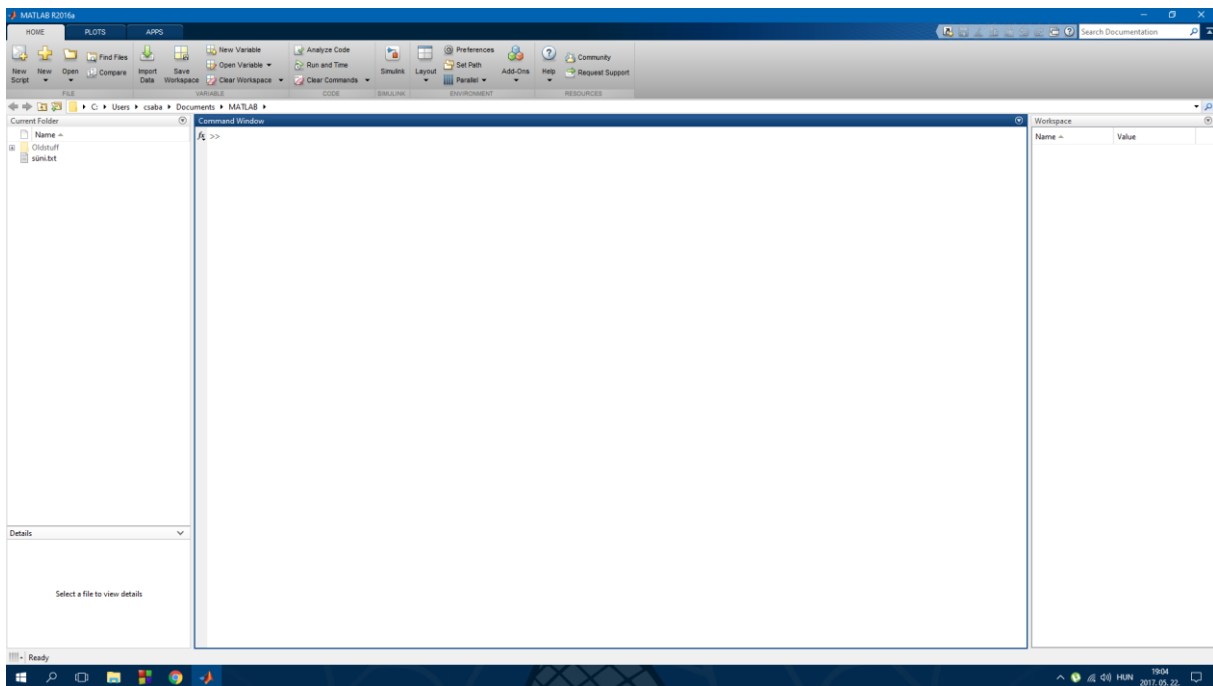
## 1.1. Mi a MATLAB? Mire jó?

A MATLAB szó a Matrix Laboratory rövidítéséből származik. A MATLAB gyakorlatilag egy teljes szoftvercsomag, amely nagyban megkönnyíti és felgyorsítja az algoritmus fejlesztést. Hatékony rengeteg szemléletes, numerikus matematikai probléma megoldására és ábrázolására. A MATLAB egy saját script nyelvvel rendelkezik, amiben implementálhatjuk az adott problémákat.

A MATLAB-ot azért érdemes tanulni, mert később sok tárgy épül rá, nagyban megkönnyíti a munkánkat, de akár matematikai problémákban is segítséget nyújthat. Első program nyelvnek is jó lehet, hiszen a legszükségesebb alap parancsokkal rendelkezik, emellett látványos eredményeket lehet elérni viszonylag kevés energia befektetéssel.

## 1.2. A MATLAB környezet

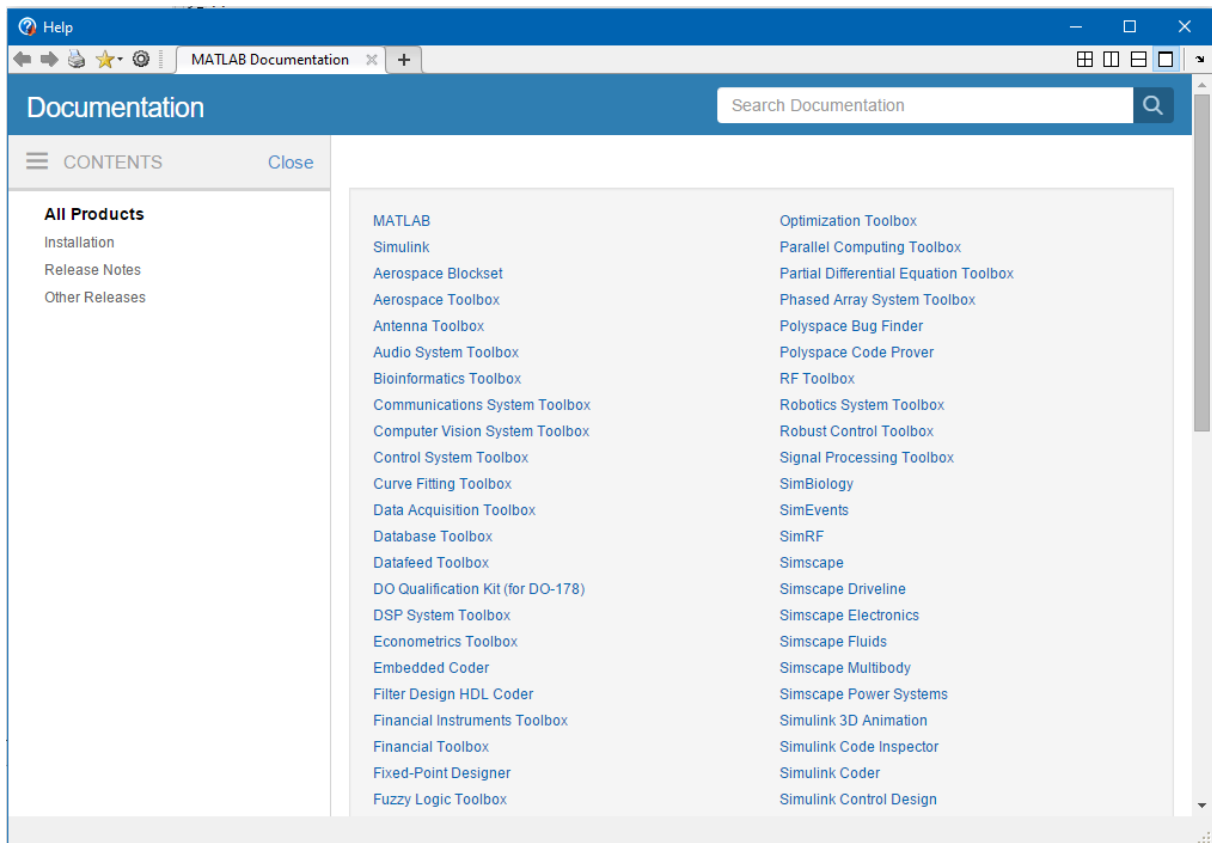
A felhasználói felület rengeteg különböző egységből áll: *Command Window*, *Editor*, *WorkSpace*, *Current Directory*, *Variable Editor*, *Help* - csak, hogy említsük azokat, amelyeket tényleg használni is fogunk az órák során.



A MATLAB kezelőfelülete (2016-os kiadás)

A *Help* használatának fontosságát nehéz kihangsúlyozni. Ez gyakorlatilag egy olyan funkció, amellyel minden egyes parancsnak a pontos ismertetését megtalálhatjuk, példákkal együtt (rendszerint több példa is tartozik egy témakörhöz). Ennek használatát érdemes minden új parancs bevezetésével tanulmányozni, hiszen ZH-n szabadon használható, és feleannyit elég megtanulni, mint azt gondolnád.

A *Help* az egyik legintuitívabb dolog a világon, szépen beírod, amit keresel és meg fogod találni, ez szinte 100%. A használatához természetesen némi előtudás is szükséges, tudnod kell, hogy mire akarsz rákeresni. A későbbiekben bármikor előhívható lesz az F1 parancs megnyomásával, és az épp az input előtt lévő parancshoz tartozó információkat fogja feldobni.

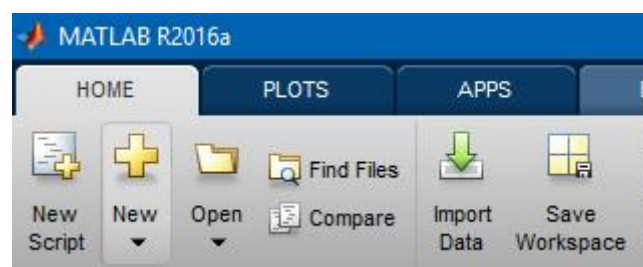


A Help

### 1.3. Röviden a MATLAB szkriptekről

A MATLAB szkriptek gyakorlatilag olyan kódok, amiket megszokhattunk más nyelvekben, pl. C++-ból. Egy egyszerű kódsorozat, ami végrehajtódik a futtatása esetén. Ezek a szkriptek .m kiterjesztéssel rendelkeznek, az egyetlen megkötés, hogy ne kezdődjön számmal, illetve space-szel a neve.

Ilyen szkripteket a bal felső sarokban található „New Script” gomb segítségével hozhatunk létre. Ekkor megnyílik az editor, ahol szerkeszthetjük azt.



Script létrehozása

### 1.4. Röviden a MATLAB függvényekről

A függvények az Editor ablakban készülnek, és hasonlóan a szkriptekhez .m kiterjesztésűek. Főleg ilyenekkel fogunk dolgozni, szóval érdemes megtanulni jól használni. Nagyon fontos, hogy függvény - más nyelvekhez hasonlóan - egy saját munkatérben dolgozik, ezért a változók, amelyeket létrehozott a lefutása után megszűnnek. A fejléc formája a function kóddal kezdődik, és az end parancs zárja le. Ez a C++-os { } megfelelője. A függvényt csak akkor fogod tudni meghívni, ha ugyanazon a néven van elmentve, mint amilyenek deklarálva van.

### 1.5. A legfontosabb alapparancsok

Ezek a parancsok / konstansok / operátorok nem szorulnak külön deklarálásra, mert a MATLAB alapból tartalmazza őket. Ezeket rengeteget fogjuk használni, mindet megjegyezni, hogy hogy működik teljesen felesleges, erre alkalmas a Help, viszont érdemes tisztában lenni, hogy egyes parancsok kb. mit csinálnak, ezzel a saját dolgunk is megkönnyítjük a Help használatában 😊.

Az alábbi alapparancsokhoz nem kívánok kommentárt fűzni a legtöbb egyértelműen következik a névből, a maradék pedig könnyen kitalálható 😊.

**Beépített konstansok:** `i, j, pi, ans, inf, -inf, nan, eps`

**Operátorok:** `+, -, *, /, ^, :, ==`

**Beépített függvények:** `sin, cos, tan, atan, sqrt, exp, power, pow2, log, log10, exp, factorial, factor, primes, round, floor, ceil, abs, datestr(clock), min, max`

**Egyéb:** `., ;, ..., %, clear X, clc, save asdf.mat a, load asdf.mat, format)`

## 2. gyakorlat - Vektorok, 2D ábrázolás

### 2.1. Sor- és oszlopvektor létrehozása

A MATLABos pályafutásunk alatt rengeteget fogunk foglalkozni vektorokkal és mátrixokkal. Ezek működésére térek itt ki.

```
sorv = [2 1 4 3];  
oszlopv = [6; 5; 7; 9];
```

Ahogy erre alaptól számítottunk, ennek a kódnak a kimenetele egy sorvektor és egy oszlopvektor lesz, a megfelelő elemekkel feltöltve.

|   |   |   |   |
|---|---|---|---|
| 2 | 1 | 4 | 3 |
|---|---|---|---|

|   |
|---|
| 6 |
| 5 |
| 7 |
| 9 |

### 2.2. Transzponált

Az egyik legalapvetőbb mátrix művelet. A legegyszerűbb módja, hogy sorvektorból oszlopvektort csináljunk.

```
sorv = sorv';
```

### 2.3. Összefűzés

Amennyiben két vektort össze kívánunk fűzni az alábbi szintaxis alapján tehetjük meg.

```
ofv = [sorv, oszlopv'];
```

### 2.4. Indexelés:

Semmi extra, de különbség minden más normális nyelvhez képest, hogy az indexelés egytől indul. Az utolsó elem indexére az `end` paranccsal hivatkozhatunk, a méret lekérdezésére `size` és `length` parancs alkalmasak, de az egyik a hosszt, a másik a dimenziószámot adja vissza.

```
elem = sorv(3);  
sorv(3) = 44;
```

### 2.5. Csupa egy és nulla tömbök

Ha olyan tömböt akarsz létrehozni, ami csupa egyesből vagy nullásból áll, akkor tudod használni ezeket. Ha egy paramétert adsz négyzetes mátrixot csinál, ha kettőt akkor 1. szám x 2. szám méretűt.

```
ones(2);  
zeros(1,3);
```

### 2.6. Random generálás

Ha véletlen számokból álló tömböt akarsz létrehozni azt a `rand` paranccsal lehet. A tömb mérete az előzőekhez hasonlóan adható meg. Az alap intervallum 0;1, lehet szorzással turbózní.

```
rand(5);
```

### 2.7. Egység mátrix generálás

Ha egység mátrixot szeretnél csinálni, akármilyen bármilyen szuper méretben, akkor az előző dolgokat használd, csak az `eye` paranccsal felturbózní. pl.:

```
eye(5);
```

### 2.8. Diagonális mátrix generálás:

Ha van egy sorvektorod, amiben olyan elemek vannak, amiket egy mátrix főátlójába akarsz behupantani, akkor használd a `diag` parancsot. pl.:

```
v = [2 1 -1 -2 -5];  
D = diag(v);
```

Egyébként használható úgyis, ha egy mátrix a bemente, hogy a diagonálisában lévő cuccok értékét visszaköpi egy oszlopvektorba.

## 2.9. Lineáris vagy logaritmikus beosztású vektor generálása:

A parancsot általában akkor fogjuk használni, ha a feladatban szükségünk van egy intervallumra, amiben két szám között egyenlő eloszlásban szerepel  $n$  darab szám. Pl.: 1 és 2 közötti intervallum 150 elemmel.

Első paraméter: mettől, második: meddig, harmadik: hány elemmel. Részletek: lásd: Help.

```
linspace(2,10,150);
```

## 2.10. Függvények

Az első órán felsorolt beépített függvények használhatók a vektorokon is, de az elemenkénti művelet végzéshez pont karaktereket kell elhelyezni a megfelelő helyen. pl.:  $-e \cdot f$

## 2.11. Felosztás

Ha nem az elemek száma a fontos, hanem a felosztás pontos meghatározása, akkor a kettőspont operátort érdemes használni. Első szám mettől, második lépésköz, harmadik meddig. Jó cucc tud lenni a legtöbb feladatnál. pl.:

```
dio = 1:0.25:2;
```

## 2.12. Összeg, átlag, szorzat

Ha az elemek összegére vagy kíváncsi egy adott tömbben, akkor a `sum` parancs a barátod. Ez az OSZLOPOKBAN lévő értékeket dobja, tehát a végén egy sorvektort kapsz, amiben benne vannak az összegek oszloponként. Ez nagyon fontos a használat logikájának szempontjából, de a Helpben lévő példák rendkívül szemléletesek. A `prod` szorzáshoz használható, ugyanúgy működik, mint a `sum`, de összeszorozza az elemeket oszloponként, ugyanígy a `mean`, ami meg átlagol.

```
A = [0 1 1; 2 3 2; 1 3 2; 4 2 2];  
M = mean(A);  
S = sum(A);  
P = prod(A);
```

## 2.13. Maximum és minimum elem

Pontosan úgy működik mechanikailag mint az előzők, csak annyi a különbség, hogy ha két bemeneti paramétert adsz meg, akkor az egyik azt dobja vissza, hogy mi a max érték a másik, meg hogy melyik indexen van. Rendkívül hasznos maximum érték és maximum hely meghatározás eseténél.

```
MI = min(A);  
MA = max(A);
```

## 2.14. Logikai indexelés, meg keresgélés

Az egyik legszuperebb parancs, amit használni fogunk a `find`, ez azt csinálja, hogy a paraméterben megadott logikai feltétel szerint keresgél egy adott vektorban, és azokkal az INDEXEKKEL tér vissza, amely indexű elemek eleget tesznek a feltételnek. A hagyományos logikai műveletek, mint pl. a `find` és a logikai indexelés abban különbözik, hogy míg előbbi logikai feltétel alapján azt adja vissza, hogy mely indexek tesznek eleget a feltételnek, addig a logikai indexelés az eredeti vektor hosszával

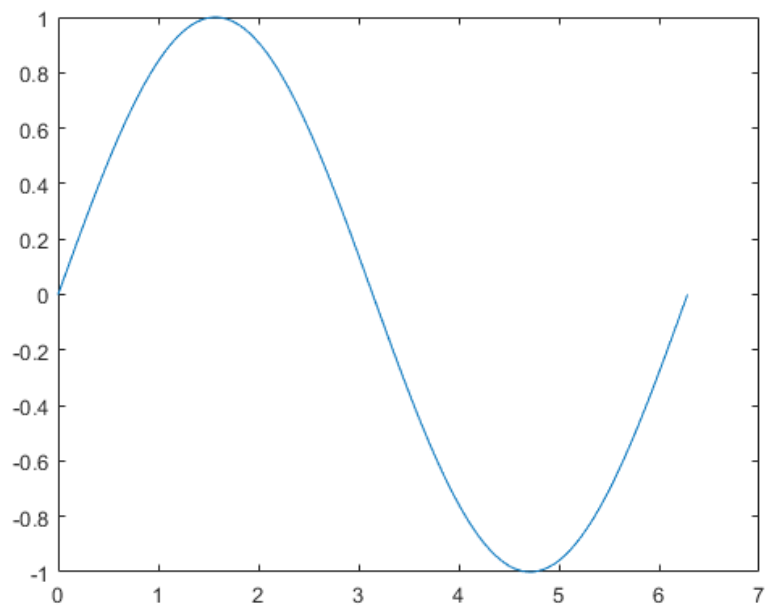
meggyező logikai vektort ad vissza, ami a 0 és 1 számok valamilyen sorozata, ahol az 1 azt jelenti, hogy teljesült az adott indexen szereplő elemre a feltétel a 0 pedig azt, hogy nem. A logikai kifejezések az & és | jelek, illetve ezek többszörösített változataival && és ||-vel írhatók le. Ez a logikus következtetés szerint az és és a vagy művelet megfelelője.

```
alma = [23 14 31 17 42];
indexek1 = find(alma>24); % → [3, 5]
alma(indexek1); % → [31, 42]

indexek2 = alma>24 % → [0, 0, 1, 0, 1]
alma(indexek2) % → [31, 42]
```

## 2.15. 2D ábrázolás

A MATLAB-ban szuperül lehet függvényeket ábrázolni, igazából a mi szintünken főleg ez a képessége az, ami jól kihasználható. Alkalmazzuk hát ezt a funkciót! A legfontosabb parancsok, amik itt szóba kerülnek, a `figure`, ami létrehoz egy új ábrát, és a `plot`, ami kirajzol egy adatsort szépen grafikusán, körülbelül úgy, mint ahogy mi rajzolnánk függvényt a papírra. Két bemeneti paramétere van az `x` értékek és az `y` értékek, és ebből szuperül összehoz egy ábrát. Ha csak egy argumentummal adod meg, akkor a vektor elemeit az indexük, tehát a helyük szerint ábrázolja.



Színusz függvény részletének ábrázolása

Érdemes megemlíteni a `stem`-et is, az is egy frankó dolog, az hasonlóan működik tőkre, mint a `plot`, viszont minden értékhez húz egy vonalat, ilyen szép csíkos lesz a grafikon. (hivatalosan ez a „pálcika diagram”). Fontos lehet még a `stairs` parancs, ami meg tőkre hasonlít ezekhez de ott meg ilyen lépcső-szerű diagrammot kapsz. Összességében ezeket elég ritkán fogjuk használni, csak a `plot` szót kell jól az agyunkba vésni ☺.

```
alma = [23 14 31 17 42];
plot(alma);

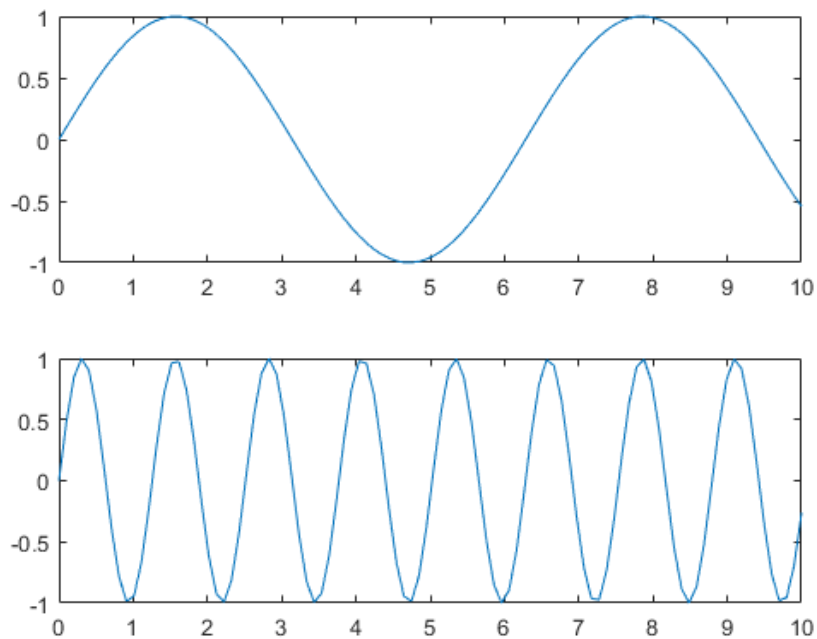
korte = [15, 19, 20, 26, 28];
plot(korte, alma);

x = 0:pi/100:2*pi;
y = sin(x);
plot(x, y);
```

Az ábrázolások kapcsán még érdemes említést tenni a `subplot`-ról, amivel ábránkat alábrákra oszthatjuk. Magyarul egy táblázatba kerül több diagram. Ez úgy működik, hogy három argumentumot fog kérni a `subplot`, az első az, hogy hány sor lesz összesen, a második hogy hány oszlop lesz

összesen, a harmadik pedig az index számot sorfolytonosan. Ha több cellába akarsz rajzolni, akkor tudod azt is jelölni, a szokásos „intervallum” jelölés segítségével.

```
x = linspace(0,10);  
y1 = sin(x);  
y2 = sin(5*x);  
  
figure  
subplot(2,1,1);  
plot(x,y1)  
  
subplot(2,1,2);  
plot(x,y2)
```

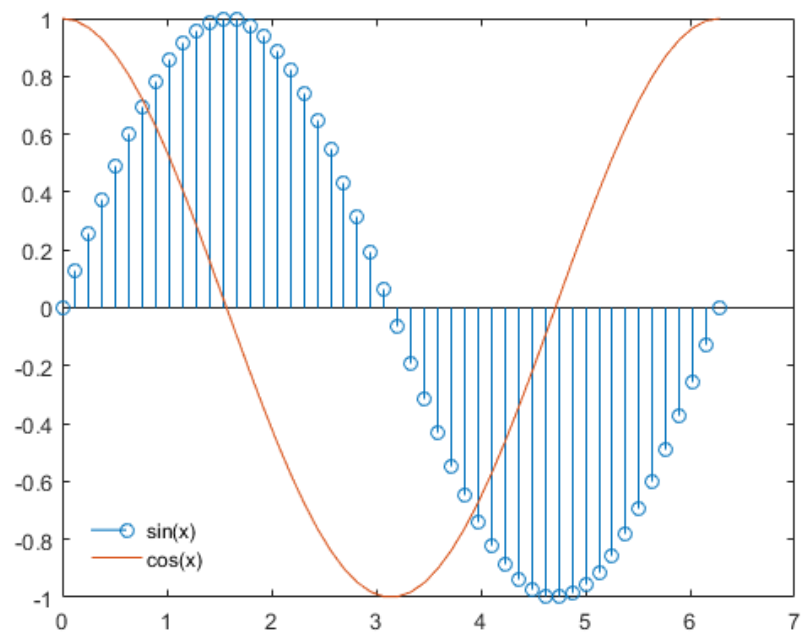


*A fenti példafüggvény kimeneti értéke*

Ha akarsz amolyan négyzethálót az ábrára, akkor van erre lehetőség, bekapcsolásra a `grid on` parancs, a kikapcsolására a `grid off` az alkalmas.

A `legend` parancs arra jó, hogy az adatsorokhoz adj jelmagyarázó szövegdobozt. A függvények grafikai megjelenését a `linespec` parancs helpre hívásával tudod megtekinteni. Az x és y tengely menti „határokat” az `xlim` és az `ylim` parancssal szabályozhatjuk. A tengely címek megadására az `xlabel`, az `ylabel`, és a főcím megadására `title` parancsok alkalmasak.

```
x = linspace(0,2*pi,50);  
ysin = sin(x);  
stem(x,ysin)  
hold on  
ycos = cos(x);  
plot(x,ycos)  
hold off  
legend('sin(x)', 'cos(x)', 'Location', 'southwest')  
legend('boxoff')
```



*A fenti példafüggvény kimeneti értéke, példa a stem-re és a legendre*

### 3. gyakorlat - Formázott kiíratás, mátrixok, vezérlőszervezetek

#### 3.1. Formázott kiíratás

A kiíratás egyik legalapvetőbb példája, ha ki akarunk írni egy szöveg vagy változó értéket a konzolra. Erre jó a `disp` parancs. A gyakorlatban ez pl. így néz ki:

```
disp(ones(2)./[2 3; 4 5]);  
% → 0.5000 0.3333  
%    0.2500 0.2000
```

A `disp` mellett megtalálhatjuk még az `fprintf` parancsot. Ez különböző szövegek kiíratására alkalmas, formázott-beágyazott mezőkkel, vagy fájlba vagy konzolra. Amit itt jó tudni, hogy a `formatSpec` részről a lényeg benne van a helpben.

```
fprintf('Teszt: \n\tEgész: %2.0f\n\tTizedes: ...  
%2.4f\n\tNormál: %e\n', 10*pi, 10*pi, 10*pi)
```

A `sprintf` tők olyan amúgy, mint az `fprintf` csak nem fájlba vagy konzolra, hanem stringbe „nyomtat”.

A `format` parancs a lebegő pontos számok kijelzésének a pontosságát állítja. Példa:

```
format short; disp(pi/100) % → 0.0314  
format long; disp(pi/100) % → 0.031415926535898  
format shortE; disp(pi/100) % → 3.1416e-02
```

#### 3.2. Stringes átalakítások (szám, mátrix) :-)

A `num2str` a „számból stringgé” konverziót hajtja végre. (tehát átalakítja a számot stringgé)

```
strPi=num2str(pi,3);  
disp(strPi); % → 3.14
```

A `mat2str` parancs egy mátrixot alakít stringgé.

```
strVector=mat2str(1:5);  
disp(strVector); % → [1 2 3 4 5]
```

#### 3.3. Mátrixos alpműveletek

Amivel még nem találkoztunk az a `squeeze`, ami egy fölösleges dimenziót távolít el, tehát azokat, amik leginkább 1-es kiterjedésűek. A `reshape` parancs is szuper, ez átméretezésre van. A `numel` megadja egy mátrix összes elemének a számát, a szokásos összegzős dolgok dimenzióként működnek. A logikai műveletek, logikai indexelés, és elemenkénti műveletvégzés az eddigiek szerint.

```
alma = [1 2 3; 4 5 6]; % soronkénti  
max(alma, [], 2) % maxkereses
```

#### 3.4. Vezérlő szervezetek - elágazások

A vártak szerint működnek, azzal a kikötéssel, hogy nincs zárójelzés, a blokk lezáródását egy `end` jelzi és a feltétel logikai értékét vizsgálja (ami nem nulla, ha igaz). Relációs operátorok az `==`, `~=`, `<`, `>`, `<=`, `>=`. Logikai operátorok már voltak korábban is, azon túl még megemlíthető a `~`, a `xor`, az `all`, és az `any`.

|                                          |                                                               |                                                                                                |
|------------------------------------------|---------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| <pre>if feltetel     parancsok end</pre> | <pre>if feltetel     parancsok1 else     parancsok2 end</pre> | <pre>if feltetel1     parancsok1 elseif feltetel2     parancsok2 else     parancsok3 end</pre> |
|------------------------------------------|---------------------------------------------------------------|------------------------------------------------------------------------------------------------|

### 3.5. Vezérlő szerkezetek - ciklusok

Nem térnék ki a részletekre, szerintem pontosan érted, hogyan működik.

|                                           |                                             |
|-------------------------------------------|---------------------------------------------|
| <pre>for n = 1:10     parancsok end</pre> | <pre>while feltetel     parancsok end</pre> |
|-------------------------------------------|---------------------------------------------|

### 3.6. Vezérlő szerkezetek - switch

Pont mint c++-ban.

```
in = input('Írj be egy számot: ');
switch in
    case 0
        disp('Nullat irtal be');
    case 1
        megoldas = 2*pi*exp(3.2);
        disp(megoldas);
    case {2, 5}
        disp('A bemenet 2 vagy 5');
    otherwise
        disp('Nem jó értéket adtal meg...');
end
```

## 4. gyakorlat - Lineáris egyenletrendszerek, leképezések

### 4.1. Egyenletrendszer megoldása:

A mátrixoknál a szorzás nem kommutatív, ezért két féle osztást is definiálunk, a bal osztást és a jobb osztást, amely implementációját tekintve  $A/B$  és  $A \setminus B$ . Ha egyenletrendszereket akarunk megoldani akkor célszerű a bal osztás operátort használni. Lineáris Algebrából tanultuk, hogy minden mátrix egy lineáris leképezésnek tekinthető. A gyakorlatokon 2D leképezésekről lesz csak szó, pl. nagyítás, forgatás stb. 2D-s esetben forgatásról általában az origó körüli forgásról beszélünk, melyet forgatási mátrix segítségével végezhetünk el. A lenti példában az

$$R(\varphi) = \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix}$$

mátrix segítségével hajtunk végre egy forgatást.

```
ang = 0.3;  
B = [cos(ang), -sin(ang);  
sin(ang), cos(ang)];  
vonal3 = vonal*B;
```

### 4.2. Egy konkrét példa lineáris egyenletrendszer megoldására

A feladat az alábbi: Anna, Béla és Cili Münchenbe utaznak a hétvégére vonattal. Amint leszállnak a RailJet-ről, elhatározzák, hogy gyümölcsöt vesznek. Be is térnek az első kisboltba, ahol:

- Anna vásárol  $a_1$  almát,  $a_2$  banánt és  $a_3$  narancsot, összesen  $d_1$  EUR-ért;
- Béla  $b_1$  almát és  $b_3$  narancsot vesz  $d_2$  EUR-ért;
- Cili  $c_2$  banánt és  $c_3$  narancsot vesz  $d_3$  EUR-ért.

Számoljuk ki, hogy mennyibe került az egyes gyümölcsök darabja, ez legyen a visszatérési vektor három értéke ([alma ára, banán ára, narancs ára]). Megoldás fv:

```
function gyumolcsok_ara = gyak4_f41_megoldas(a,b,c,d)  
    A=[a;b;c];  
    gyumolcsok_ara=A\d;  
end
```

## 5. gyakorlat - Polinomok, deriválás, integrálás, anonim függvények

### 5.1. Polinomok

A MATLAB-ban a polinomokat az együttható vektorokkal reprezentálhatjuk:

```
P1 (x) = x2 + 2x + 3 → P1 = [1 2 3];
P2 (x) = 10x3 + 4x2 + 5x - 7 → P2 = [10 4 5 -7];
P3 (x) = 3x4 + 5x2 - 12 → P3 = [3 0 5 0 -12];
```

Ezekben a  $P_n$  vektorok egyszerű sorvektorok. Az általunk megadott vektorokat a MATLAB megfelelő beépített függvényei szuper módon polinomként fogják eleve kezelni. Ezekkel nagyon nagyot könnyíthetünk a dolgunkon, ezekben az esetekben:

- Kiszámolhatjuk a polinom gyökeit a `roots(P)` paranccsal
- Kiértékelhetjük egy adott pontban a `polyval` utasítás segítségével. Parasztosabban ez azt jelenti, hogy egy megadott pontban kiszámolja a függvény értéket. Ha ez a megadott pont vektor, akkor egy olyan vektor a kimenet, amely ugyanakkora, mint a bemeneti és az adott indexekhez tartozó értékek vannak benne.
- Létrehozhatunk egy polinomot a gyökeinek ismeretével, erre a `poly(r)` parancs a legjobb, amiben az `r` egy bemeneti vektor a gyökökkel. Pl. `r = [-1, 1]`.
- Az egyik legérdekesebb dolog, hogy illeszthetünk polinomot egy tetszőleges adatsorra is. Ezt a `polyfit` paranccsal tehetjük meg. Ennek első bemenete az `x` értékek, a második az `y`, a harmadik pedig a fokszám, azaz, hogy hányadrendű polinomot akarunk ráilleszteni.

```
roots(P);
y0 = polyval(P, x0);
x = -1:0.01:1;
y = polyval(P, x);
r = [-1 1];
P = poly(r);
P = polyfit(x_t, y_t, fokszam);
```

### 5.2. Numerikus deriválás

Mivel a MATLAB a hagyományos értelemben nem képes a deriválásra, tehát diszkrét értékeken történik, éppen ezért rendkívül fontos a jelek felbontása. Általánosságban elmondhatjuk, hogy minél sűrűbb felbontást adunk meg, annál pontosabb derivált függvényt kapunk. Analízisből tudjuk, hogy a differenciálhatóság legfontosabb feltétele általában a függvény folytonossága. A MATLAB-bal ilyenről nem beszélünk, lévén, hogy nem „végtelen sűrűek” a beosztások, de megfelelő felbontást választva jó közelítéssel tudunk numerikus deriváltakat számolni.

Az alap parancs amire szükségünk van a `diff`, ami egy vektorban kiszámolja az egymást követő elemek különbségét. Ezzel létrehozuk az értelmezési tartomány és az értékkészlet adatsorainak különbségét, és vesszük ezek hányadosát, így megkapjuk a differenciahányadost.

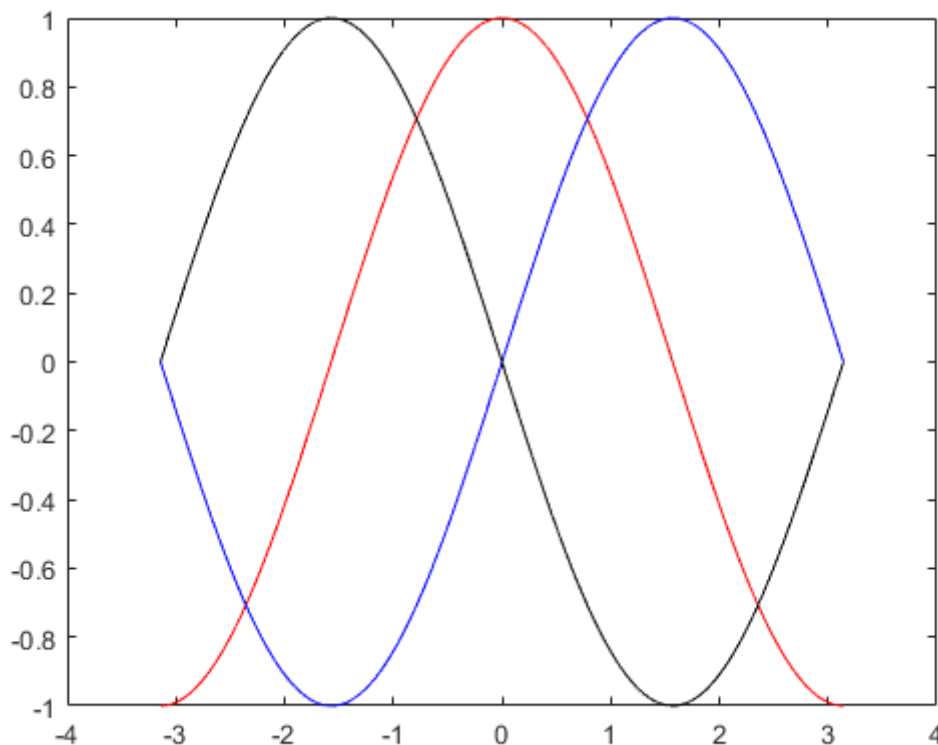
$$\lim_{x \rightarrow x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

Érdemes nagyon odafigyelni, hogy a különbség vektornak eggyel kevesebb eleme lesz, mint az eredetinek, így amikor ábrázolni próbáljuk a `plot` parancs segítségével nagyon oda kell figyelni az indexelésre.

Ha a vizsgált függvény képlettel felírható, tehát polinom vagy szögfüggvény például, akkor adott

hosszúságú és felbontású mintavétel nélkül is megadható egy adott pontban a derivált, csak egy tetszőlegesen kicsi környezetet vizsgálunk.

```
h = 0.001;           % lépésköz
X = -pi:h:pi;         % értelmezési tartomány
f = sin(X);           % a függvény értékek
Y = diff(f)/h;         % első derivált
Z = diff(Y)/h;         % második derivált
plot(X(:,1:length(Y)),Y,'r',X,f,'b', X(:,1:length(Z)),Z,'k')
```



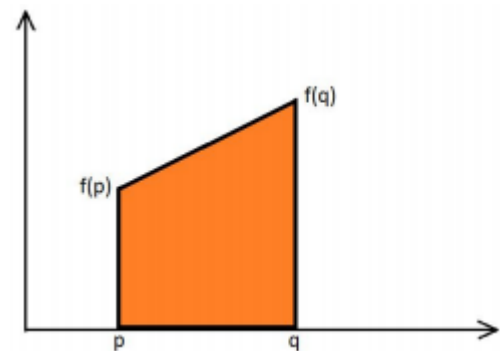
*Az ábrán a kék függvény az eredeti függvény, a piros függvény az első derivált, a fekete pedig a második derivált*

### 5.3. Numerikus integrálás

A deriváláshoz hasonlóan tudunk integrálni is, amit szemlélet jelentéséből fakadóan a függvényértékek és az x tengely közötti területrészek előjeles összegeként számolhatunk. Az integráláshoz alapvetően három módszert tudunk alkalmazni, de ebből csak kettőt fogunk ténylegesen használni, ezért az „Egyszerű összeadás módszerét” most nem fogom bemutatni, de az előadás diákban megtalálható.

#### 5.3.1. Trapéz szabály

Ennek szemléletesen az a lényege, hogy a függvény alatti területet trapézok területeire bontja fel, és azokat számolja ki, majd összegezi. Ez elég szuper és logikus. Parancsa a `trapz(x, y)`, ahol `x` az értelmezési tartomány, `y` az értékkészlet. Főleg akkor fogjuk alkalmazni, ha a konkrét görbét (tehát az egyenletét) nem ismerjük, viszont az értelmezési tartománnyal és az értékkészlettel tisztában vagyunk.



*A trapéz módszer szemléletes ábrázolása*

### 5.3.2. Spin-off

A MATLAB-nak egy szuper tulajdonsága, hogy tudsz benne függvényeket változóban tárolni, ha azok nem túlzottan bonyolultak. Ennek a szintaktikája:

```
fv = @(x) x+3;
```

A @ után megadod a bemeneti paramétereket, utána leírod a függvényt teljes valójában. Ezután az „fv” változó az  $f(x) = x + 3$  függvényt takarja majd.

### 5.3.3. Függvényes integrálás

A függvényes integrálás lényegében arról szól, hogy a MATLAB klasszikus integrálási parancsát meghívjuk egy függvényre, amit az előzőekben ismertetett módszerrel pl. egy változóban tárolhatunk. Az `integral` parancs első paramétere maga a függvény lesz, a másik kettő pedig a Newton-Leibniz tételben található a és b megfelelője - tehát az intervallum alsó és felső határa lesz, melyen integrálni kívánunk. Például:

```
fun = @(x) exp(-x.^2).*log(x).^2;  
q = integral(fun,0,Inf)
```

Ekkor a q változóba egy konstans érték kerül, amely a fun-ban definiált

$$f(x) = e^{-x^2} \cdot (\ln x)^2$$

függvény integrálja a 0-tól végtelenig terjedő intervallumon.

## 6. gyakorlat - Differenciálegyenletek

### 6.1. Differenciálegyenletek

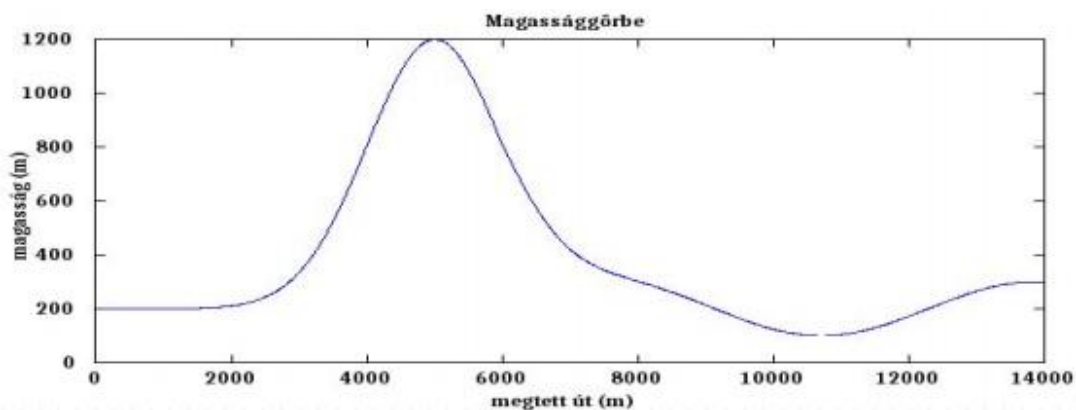
Analízisből tanultuk, hogy a differenciálegyenletek olyan egyenletek, amelyekben az ismeretlen egy függvény, és az egyenletben szerepel a deriváltjuk is. A MATLAB alkalmas lehet differenciálegyenletek megoldására is, amihez szintén numerikus integrálást alkalmaz a program.

### 6.2. Szemléletes megközelítés, gyakorlati példa

Szemléletesen egy példát bemutatva, kb. az a lényeg, hogyha például egy hegyi úton sétálunk akkor ugye a magasságunk az előrehaladás függvényében változik. A magasságunkat felírhatjuk pl. az idő a hosszúsági szélességi kör, vagy a megtett út függvényében is. A megtett út függvényében a magasságra a következő összefüggés lesz felírható. (ahol  $x$  az út  $y$  meg a magasság)

$$y = y(x)$$

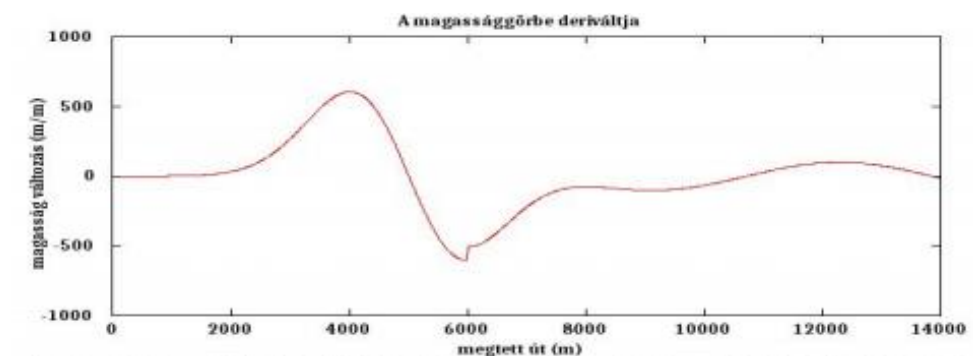
Ha ilyenkor van nálunk például egy GPS vevő, ami méri a magasságunk is, akkor az előrehaladás közben elegendő ponton felírva a magasság értékeket akkor megkapjuk a fenti függvény értékeit.



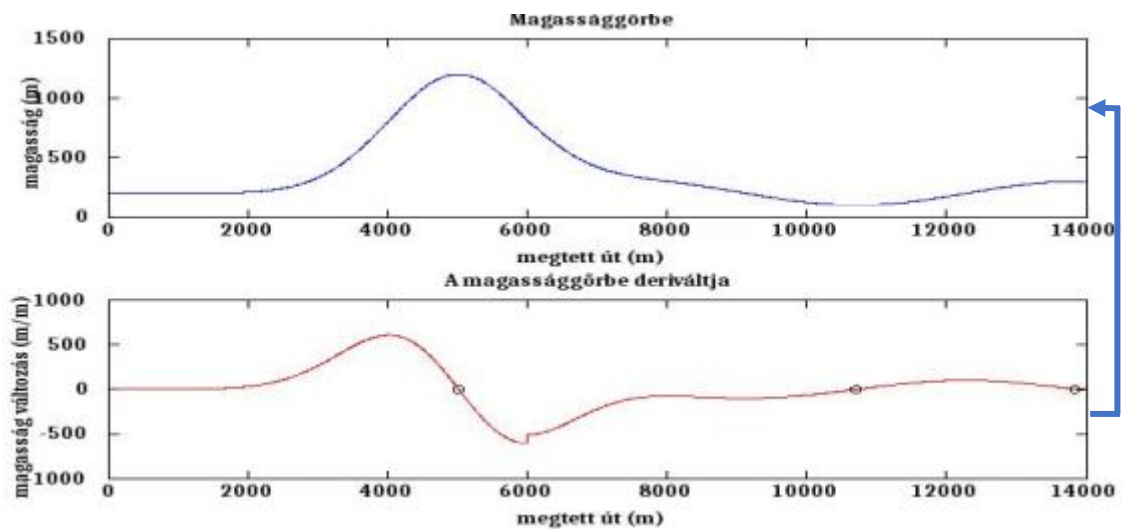
Ez így egy viszonylag egyszerű megoldás lenne, de tegyük fel, hogy nincsen nálunk megfelelő mérőeszköz, viszont látjuk az útmenti emelkedést jelző táblát, amely például 5%-os emelkedőt jelez. Ekkor a tábla megfelelően kicsi környezetében egy tetszőleges  $h = 100$  -ra az alább összefüggést írhatjuk fel: (itt a baloldal az út meredeksége az  $x$  és az  $x+h$  között)

$$\frac{y(x+h) - y(x)}{h} = 0.05$$

Tegyük fel, hogy az út mentén pár méterenként találunk egy ilyen táblát, melyek a magasság változásának közelítő értékeit adják meg a megtett út függvényében. Ha ezeket az értékeket felírjuk, abból megkapjuk a  $\frac{dy}{dx}$  értéket, amely a függvény deriváltja lesz.



Az így kapott görbénket numerikusan integrálva megkapjuk az eredeti  $y = y(x)$  értékeket.



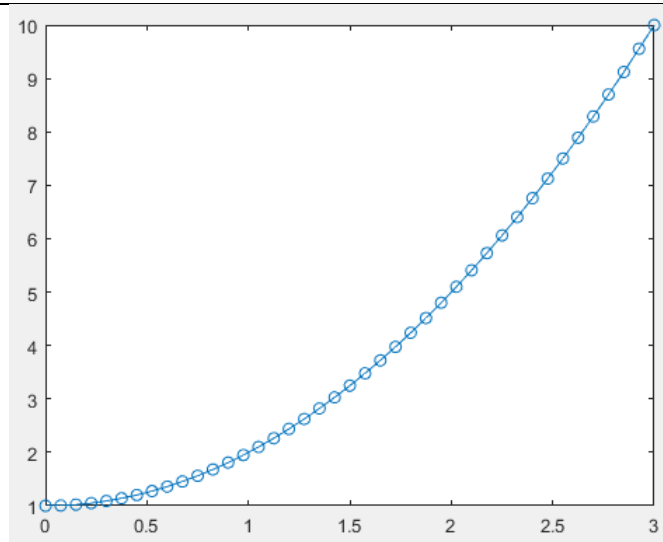
### 6.3. Egy konkrét feladat megoldása

Teszteljük a tudásunk egy konkrét példában! A differenciálegyenleteket a MATLAB-ban praktikusán két módszerrel oldhatjuk meg: az egyik az explicit Euler módszer, de ez relatíve bonyolult, sok plusz munkát igényel, ezért mi a MATLAB beépített megoldó függvényét fogjuk használni, az `ode45`-öt. Ennek használatát, az alábbi feladat és megoldása jól szemlélteti:

- Legyen adott a következő egyszerű, elsőrendű differenciálegyenlet:  

$$y'(t) = 2y(t)$$
- Adjuk meg az  $y(t)$  értékeit a  $t = [0, 3]$  intervallumon az  $y(0) = 1$  kezdeti érték esetén!

```
f = @(y,t) 2*y;
tspan = [0 3];
y0 = 1;
[t45, y45] = ode45(f, tspan, y0);
plot(t45, y45);
```



A kód eredménye: a differenciálegyenlet megoldás függvénye

A diákban számos egyéb példa is található, célszerű átnézni, hogy minél inkább „beidegződjön” ez a tudás.

## 7. gyakorlat - 3D ábrázolás

### 7.1. 3D ábrázolás

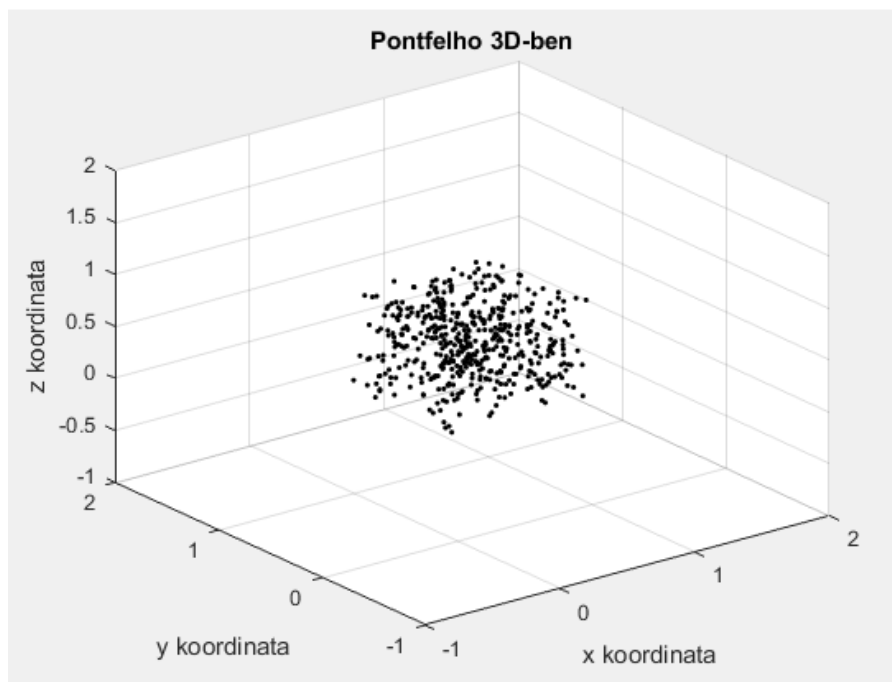
Mint azt már korábban említettem a MATLAB egyik leghasznosabb általunk is alkalmazott tulajdonsága a különböző függvények ábrázolása. Habár 2D-ben csak egyszerűen leegyszerűsíti a dolgunk azzal, hogy nem kell túl sokat gondolkodnunk minden probléma előtt, 3D-ben már sokkal komolyabb potenciál rejlik benne. 3D-s függvényeket szinte lehetetlen ábrázolni papíron. Néhány egyszerűbbet még csak-csak, de a bonyolultabbak ábrázolás számítógépes eszközök nélkül szinte lehetetlen. Hálstennek erre van már megoldás! 😊

A 3D-s függvények ábrázolása a plothoz nagyon hasonlóan működik: szintén az értelmezéstartomány - értékkészlet megadásának kombóján alapszik, csupán azzal a különbséggel, hogy más tartományokról van szó. Jellemzően, amiket rajzolgatni fogunk így, azok pontfelhők, felületek, illetve vektormezők lesznek.

### 7.2. Pontfelhők

A pontfelhők ábrázolására alkalmas parancs a `plot3`, Működéséhez 3 paramétert fog várni, az X az Y és Z koordinátákat tartalmazó vektorokat, melyeknek azonos indexhez tartozó értékeit egy adott pont koordinátáinak fog tekinteni, amit kirajzol a térben.

```
Pontfelho = rand(500,3);  
figure(1)  
plot3(Pontfelho(:,1),Pontfelho(:,2),Pontfelho(:,3), 'k.')  
title('Pontfelho 3D-ben')  
xlabel('x koordinata')  
ylabel('y koordinata')  
zlabel('z koordinata')  
xlim([-1 2]);ylim([-1 2]); zlim([-1 2])  
grid on
```



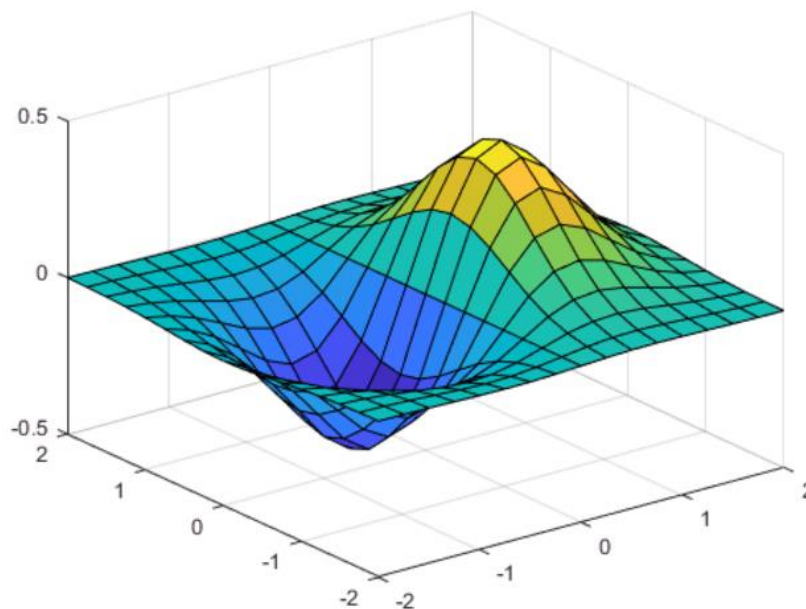
3D-s pontfelhőre példa - a kód kiemenete

### 7.3. Felületek ábrázolása

Konkrét felületek, valamint függvények kirajzolása nagyon hasonló módon történik. Itt szükségünk lesz a `meshgrid` parancsra, ezzel tudunk létrehozni egy térhálót, „amire” majd készül az ábra, pl. az `[X,Y] = meshgrid(-2:0.1:2)` parancs egy olyan hálót feszít ki, ami x és y irányban is -2-től megy 2-ig 0,1-es lépésközzel. Ha megadsz első és második intervallumot is, akkor az első intervallum az x-es kiterjedés a második meg az y tengely irányú ☺.

Az ábra kirajzolásához a `plot3` parancs helyett a `surf` lesz szükség, amely paramétereit tekintve teljesen megegyezik a `plot3`-mal. Pl.:

```
x = -2:0.25:2;
y = x;
[X,Y] = meshgrid(x);
F = X.*exp(-X.^2-Y.^2);
surf(X,Y,F)
```



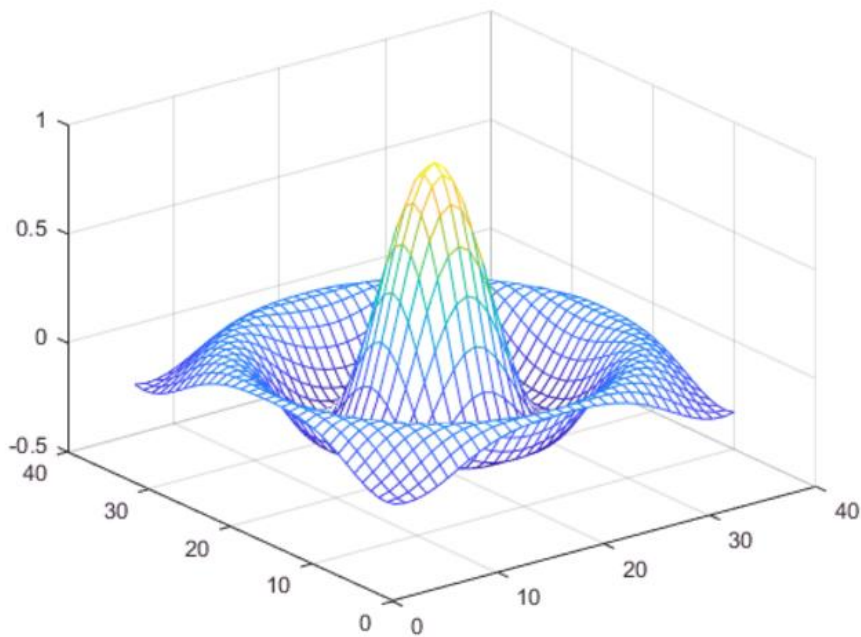
*Példa felület - a fenti kód kimenete*

Az itt megadott függvény a  $F(x) = x e^{-x^2-y^2}$ , melyet a  $[-2, 2]$  intervallumon ábrázoltunk mind x mind y irányban, 0.25-ös felbontással.

A `surf`-ön kívül hasznos parancs még a `mesh`. A két parancs teljesen hasonlóan működik, de a grafikus megjelenésük nagyban eltér egymástól. Mikor a lentihez hasonló ábrát várunk, akkor érdemes alkalmazni.

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;

figure
mesh(Z)
```



A mesh parancs eredménye

#### 7.4. Szintvonalak ábrázolása

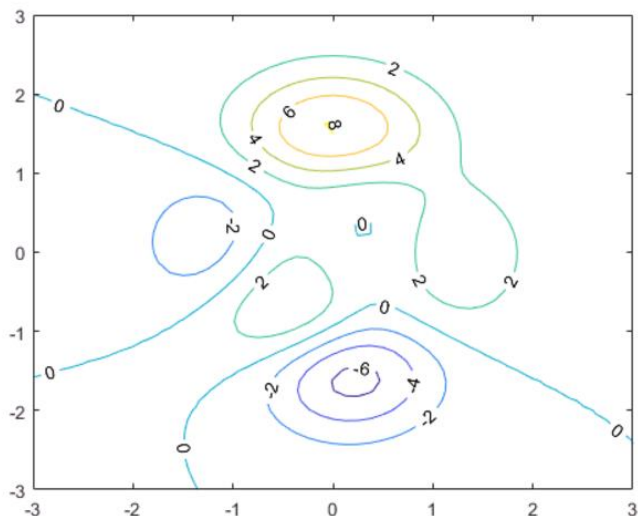
A szintvonalak a függvények ábrázolását segítik nekünk, egy adott szintvonalon a függvény „magassága”, tehát a  $z$  értékei megegyeznek. Ez térképészetben egy nagyon gyakran használt eszköz. A szintvonalak ábrázolására a `contour` a legalkalmasabb parancs.

Példának okáért hozzunk létre egy térhálót a `meshgrid`-del, majd használjuk a `peaks(X,Y)`-t (egy hasznos parancs, ami szemléletesen mutat nekünk egy bizonyos függvényt).

Amennyiben egy feladat a szintvonalak felcímkzését is kéri, erre jó a `clabel` parancs, melynek paramétereként megadjuk a  $C$ -t, tehát a `contour` értékét.

A parancs arra alkalmas, hogy a különböző szintvonalakhoz tartozó értékeket az ábrán grafikusán is megjelenítse.

Előfordulhat, hogy a függvényünket egy ábrán kívánjuk ábrázolni a szintvonalakkal, ebben az esetben a `surf` parancs segíthet.



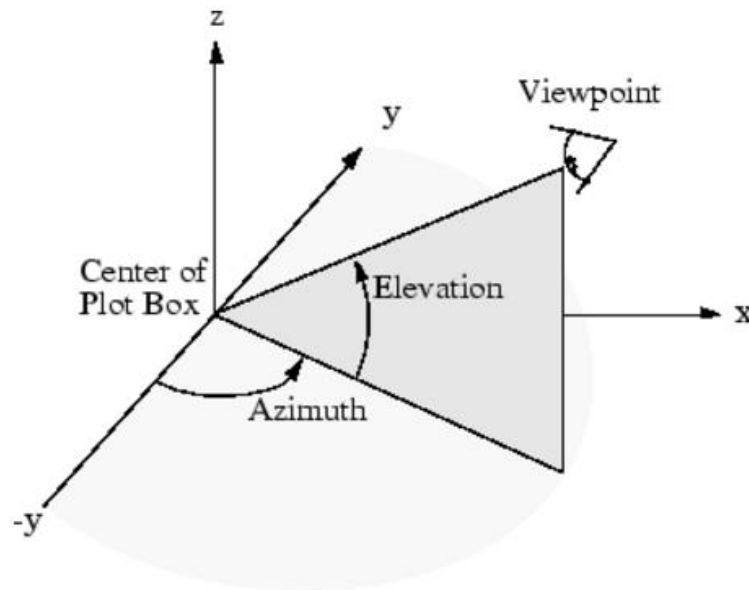
A szintvonalak ábrázolása gyakorlatban

```
[x,y,z] = peaks;
[C,h] = contour(x,y,z);
clabel(C,h)
```

Az ilyen szitukban egyébként ugye a Matlab úgy rajzolja a függvényt, hogy a színe a magasság szerint változik. Ha azt akarjuk, hogy látszódjon a magasság mértéke egy jobb oldali színsávban csak tegyük oda egy `colorbar` parancsot.

### 7.5. Nézet beállítódása

A feladatok gyakran fogják kérni, hogy állítsad egy bizonyos nézőpontra az ábrát. Ez a `view` paranccsal lehetséges. Két bemeneti paramétere meghatározza a szögbeli eltérést az origótól. A parancs első bemenete az *azimuth* eltérést, a második az *elevation* adja meg. Ezek szemléletes jelentése az alábbi ábrán jól megfigyelhetőek. ☺



A szögelfordulás szemléletes jelentése a `view` parancs megvalósításával

## 8. gyakorlat – Cellatömbök, struktúrák, fájlműveletek

### 8.1. Cellatömbök

A cellatömb egy olyan adattípus, ami különböző típusú vagy méretű változók tárolására alkalmas. Létrehozhatunk új üres cellatömböt a `cell` paranccsal. Pl.:

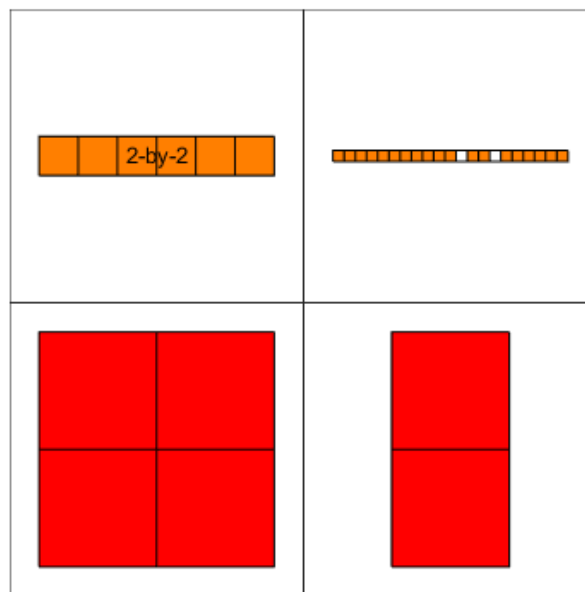
```
C = cell(2,2);
```

A cellatömb a `{}` zárójelekkel a rész cellatömb elemeinek indexelése pedig `()`-kel történik. Pl.: ha ismer elemek vannak egy cellatömbben:

```
C = {1, 2; 3, 4};
```

Egyes cellákban igazából teljesen eltérő elemeket is tárolhatunk, akár további cellatömböket is. A cellatömböket megjeleníthetjük a `cellplot` paranccsal. A cellatömbök értékadása több módon is történhet: meg lehet tölteni elemenként, és egy sorban is.

```
c{1,1} = '2-by-2';  
c{1,2} = 'eigenvalues of eye(2)';  
c{2,1} = eye(2);  
c{2,2} = eig(eye(2));
```



*Példa a cellatömb ábrázolására a `cellplot` parancs segítségével*

### 8.2. Struktúrák

A cellatömbökhöz hasonlóan különböző típusú és/vagy méretű változók tárolására használható adattípus, azzal a különbséggel, hogy a tárolt adatok névvel ellátott mezőkben vannak. Nagyon hasonlít a C, és C++ nyelvek struct típusához. Az adatok címzése a „változónév.mezőnév” minta alapján történik. Egy példa, hogy hogyan működik ez a gyakorlatban:

```
hallgato.nev = 'Kis Pista';  
hallgato.szul_datum = '1992.03.24.';  
hallgato.evfolyam = 3;  
hallgato.osztalyzatok = [2 4 3 5 3 4 4 3];
```

A struktúra értékeinek a lekérdezése pl.

```
>> hallgato
hallgato =
      nev: 'Kis Pista'
    szul_datum: '1992.03.24.'
    evfolyam: 3 osztalyzatok: [2 4 3 5 3 4 4 3]
>> hallgato.nev
ans = Kis Pista
```

### 8.3. Fájlkezelés

A MATLAB A MATLAB-bal való munkánk során nem egyszer fogunk találkozni olyan adatsorokkal, melyek felhasználáshoz fájlbeolvasást kell létrehozni. A fájlokban tárolt adatok használatához először meg kell nyitnunk a file-t a „fopen” paranccsal, majd a legvégén zárjuk be az „fclose” paranccsal. A megnyitás létfontosságú, a bezárás azonban csak formalitás, nem nagyon fogunk olyan helyzettel találkozni, ahol nagy hibát követünk el azzal, ha elfelejtjük a bezárást.

A fájlok két fő fajtáját fogjuk megkülönböztetni: a formázott szövegeket, illetve a bináris fájlokat. A formázott szövegek használhatók logok készítésére, vagy egyszerű adatok olvashatóak le belőle tárolásra. A bináris forma esetében az adatok kompakt tárolása a cél, általában csak programmal olvashatóak, és a formátumok nem feltétlenül derülnek ki. A MATLAB beépített tárolási módja is bináris fájlokat generál és olvas be (.mat).

### 8.4. Formázott szöveg olvasása és írása

A beolvasott szöveg kiíratásához az `fprintf` parancs használható. Itt megadod paraméterként a file változót, aztán a szöveget, amit bele akarsz írni. A 'w' paraméter azt jelzi, hogy írni is akarjuk ☺. pl.:

```
fid = fopen('süni.txt', 'w');
fprintf(fid, 'szöveg');
```

Olvasni is tudunk a fájlokból. Ha formázott szövegről van szó, akkor az `fscanf` parancsot illik használni. Ennek egy példája:

```
% fajl megnyitasa olvasasra
fid=fopen('file.txt');
% eloszor a header beolvasasa stringkent
Sheader=fscanf(fid,'%s',6);
% majd az adatok beolvasasa lebegopontos szamkent
Sdata=fscanf(fid,'%f %f',[2 inf]);
% fajl bezarasa
fclose(fid);
% beolvasott adatok kirajzolasa
figure;
plot(Sdata');
legend('t','sin');
```

### 8.5. Bináris fájlok olvasása és írása

Bináris fájlba írás esetén az `fprintf` parancs helyett az `fwrite`-t használjuk. Ennek első bemenete a file változó, a második egy tetszőleges számsorozat, a harmadik meg az adatok típusa idézőjelben, pl 'double'. Láthatóan teljesen hasonlóan működik a `sim` printes parancshoz, csak a végeredmény kódolása lesz különböző ☺.

A fileből olvasás is hasonlóan történik a bináris formában, mint formázott szövegekből, de az `fscanf` helyett, az `freadf` lesz az általunk alkalmazott parancs. A parancs bemenetei hasonlóan, mint az előző esetben teljesen megegyeznek a „formázott párjával”, egyéb részletek a Helpben találhatóak.

## 8.6. Fájlok mentése és betöltése

Amennyiben egy kirajzolt `plot`ot, egy bizonyos ábrát kívánunk képként elmenteni a `saveas` parancsot használhatjuk. Ennek részletes bemutatása praktikusán le van írva a *Help* megfelelő részében, de gyakorlati használatának bemutatására az alábbi kód szemléletes lehet.

```
x = [2 4 7 2 4 5 2 5 1 4];  
bar(x);  
saveas(gcf, 'Barchart.png')
```

Speciális eset, ha a MATLAB saját formátumában szeretnénk „menteni és olvasni”. Ez körülbelül azt jelenti, hogy a változóneveket értékeket, és egyéb használt parancsokat, történetet stb. kimentünk egy fájlba, illetve betöltünk egy másikkól. Erre a `save` és a `load` parancs alkalmasak.

## 9. gyakorlat – Táblázatok, fájlműveletek, képmentés

### 9.1. Táblázatok alapjai és létrehozásuk

A táblázat egy olyan adattípus, amely különböző típusú és méretű változók rendszerezett tárolására alkalmas. Az oszlopokba sorolt adatok tárolására a legmegfelelőbb, amire jó példák a vesszővel tagolt fájlok (csv), vagy a klasszikus excel táblák (xls,xlsx) is.

Táblákat a legegyszerűbben a `table` paranccsal hozhatunk létre. Az elemeik felsorolása után a „változóneveket”, azaz a megfelelő oszlopok nevét is meg kell adnunk a `'VariableNames'` parancs segítségével.

```
t = table(1,1,1,'VariableNames',{'o1','o2','o3'})

t =   o1 o2 o3
      -- -- --
      1  1  1
```

Táblázatokat létrehozhatunk tömbökből az `array2table`, a `cell2table` és a `struct2table` parancsok megfelelő helyzetben való felhasználásával. Természetesen odavissza működő parancsokról van szó, ezeknek a fordított „párjai” a `table2array`, a `table2cell` és a `table2struct`.

Az adatokat a táblázatban háromféle képpen indexelhetjük.

- `()` - táblázatot ad vissza
- `{}` - tömböt ad vissza
- oszlopnévvel - vektort ad vissza

### 9.2. Táblázatok összefűzése, alapl műveletek

|                                               |                                                                                                                                   |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| <code>istable</code>                          | logikai értékkel tér vissza: 1, ha a beadott paraméter táblázat, 0, ha nem az.                                                    |
| <code>height, width</code>                    | táblázat sor- és oszlopszámának lekérésére alkalmas parancsok                                                                     |
| <code>summary</code>                          | a tábla tartalmának statisztikai összefoglalója rendezett formában.                                                               |
| <code>ismember, ismissing</code>              | hasonlóan az <code>istable</code> hez, le ellenőrzi, hogy valamilyen változó megtalálható egy másik struktúrában                  |
| <code>sortrows</code>                         | az első oszlop szerint sorba rendezi a mátrix sorait                                                                              |
| <code>varfun(func, A)</code>                  | létrehoz egy olyan táblát, ami az A táblának elemeinek <code>func</code> függvény meghívására létrejött képeit tartalmazza        |
| <code>rowfun()</code>                         | hasonlóan a <code>varfun</code> hoz, de sorokra van meghívva                                                                      |
| <code>standardizeMissing(A, indicator)</code> | bizonyos elemeket kicserél a táblában az adott hiányzó elem értékre, ami általában a NaN lesz.                                    |
| <code>unique</code>                           | ugyanazon elemekkel tér vissza, mint az eredeti tábla, de minden elem csak egyszer szerepel benne                                 |
| <code>intersect</code>                        | két bemeneti paraméterből azokat az elemeket adja vissza, amely mindkét szerkezetben szerepel, de mindegyik csak egyszer szerepel |
| <code>union</code>                            | klasszikus unió művelet ismétlődés nélkül                                                                                         |

|                                   |                                                                                                                |
|-----------------------------------|----------------------------------------------------------------------------------------------------------------|
| <code>setdiff(A,B)</code>         | A két halmaz különbségével tér vissza: azok az elemek vannak benne, amelyek csak A-ban vannak ismétlés nélkül. |
| <code>setxor(A,B)</code>          | kizáró vagy alkalmazása. azok az elemek, amelyek maximum az egyik halmazban szerepelnek                        |
| <code>innerjoin, outerjoin</code> | táblázatok összefűzésére alkalmas, különböző logika mentén: lásd a Helpben ☺                                   |

### 9.3. Fájlba írás és olvasás táblázatként

Amennyiben konkrét táblázat típusból olvasunk be (pl. xls, xlsx, csv, stb.) célszerű a szokásos beolvasási parancsok (`fopen`, `freadf`, `fscanf` stb.) helyett a `table` saját függvényeit használnunk.

A `readtable('filename.csv')`, a `filename.csv` nevű fájlban lévő táblázatot beolvassa egy MATLAB által kezelt táblázat változóba ☺ Hasonlóan működik a `writetable` is. Ezek a parancsok több bemeneti paraméterrel is rendelkezhetnek, ezeket meg lehet találni a Helpben.

### 9.4. Grafikon fájlba mentése

Korábban már volt szó egy `plot` fájlba mentéséről, de ennek kicsit „kőkorszakibb” módját is be kell mutatni, mert a gyakorlatokon ez szerepelt, bizonyosan jó okkal.

A `print` parancsot fogjuk használni ehhez, erre egy ékes példát kimenettel együtt itt találhattok.

```
f1=figure;
surf(peaks)
title('Peaks felület')
xlabel('x')
ylabel('y')
zlabel('z')
colorbar
set(gca)
print(f1, '-dpng', '-zbuffer', '-r300', 'surf1.png')
```

