2. MATLAB ELÁGAZÁSOK, CIKLUSOK, FÁJLMŰVELETEK

LOGIKAI MŰVELETEK

A logikai műveletek (1-igaz/0-hamis) ismerete is nagyon fontos, különösen, ami a vektorok/mátrixok elemeinek módosítását, lekérdezését illeti. Sok olyan feladatot is meg tudunk oldani a mátrixokkal és logikai változókkal, amihez különben valamilyen ciklus kellene.

```
> % egyenlo ==, nem egyenlo ~=
> a = 3==4 % hamis - 0
> whos a
> b = 5~=6 % igaz - 1
> vs = [1 2 3 4 5 6] % sorvektor
> vs(5)>5
> vs(5)>=5
> % vagy (or) ||, es (and) &&
> a || b % igaz, mert a ket feltetel kozott van igaz
> a && b % hamis, mert csak az egyik feltetel igaz
```

Nézzünk egy példát, ahol egy vektor adott tulajdonságú elemeit kérdezzük le logikai változóval. Legyen egy műegyetemi tanár, aki véletlenszerűen osztogatja a jegyeket a diákoknak a vizsgán. Most éppen 6 vizsgázója van (a,b,c,d,e,f). Kapjon mindenki egy jegyet 1-5 között és utána kérdezzük le hányan buktak és kik?

```
> vizsgazok = ['a';'b';'c';'d';'e';'f']
> vizsga = randi([1,5],1,6) % 1-5 között véletlen egész számok, 1 sor,
  6 oszlop, vagy:
> vizsga = randi(5,1,6) % ha a legkisebb érték 1, azt nem kell kiírni
> bukott = vizsga<2
> vizsgazok(bukott)
```

A bukott=vizsga<2 eredménye egy 6 elemű vektor lesz, ahol 1-es áll azokon a helyeken, amire igaz volt a feltétel, 0 a többin. Ha a vizsgázók nevei közül le akarjuk kérdezni azokat, akik megbuktak, akkor nem kell mást tennünk, mint meghívni a vizsgazok(bukott) parancsot, ez csak azokat a neveket fogja visszaadni, ahol 1-es állt a bukott vektorban. Egy ilyen lekérdezést matlab-ban ciklus nélkül is meg lehet oldani, logikai változókat használva. Megjegyzés: kerekítési parancsok: round, ceil, floor, fix (bővebben, lásd: help).

ELÁGAZÁSOK, CIKLUSOK

KÉTIRÁNYÚ FELTÉTELES ELÁGAZÁS - IF, ELSEIF, ELSE

Az **if**, **elseif**, **else, end** típusú szerkezet kétirányú elágazásokat használ. Az alap **if** utasítás szerkezete: **if** feltétel; mit tegyen ha igaz a feltétel (lehet egy vagy több sorban is), **end**; Az end előtt lehetnek elágazások benne az **elseif** (egyébként, ha…), illetve az **else** (egyébként) használatával. Az if-else szerkezete Matlab-ban:

if (feltételes kifejezés)

(Matlab utasítások)

elseif (feltételes kifejezés)

(Matlab utasítások)

else

(Matlab utasítások)

end

Ábrázoljuk, majd oldjuk meg a következő másodfokú egyenleteket, adjuk meg, hogy hány valós gyöke (zérushelye) van, és melyek azok!

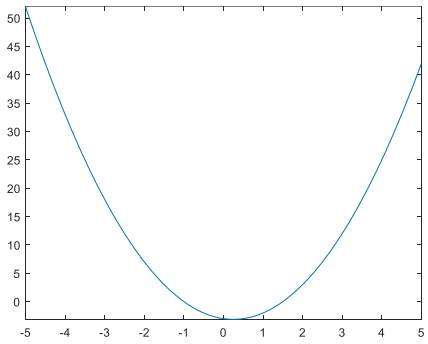
$$2\,x^2 - x - 3 = 0$$

$$x^2 + 2\,x + 3 = 0$$

$$2\,x^2 + 4\,x + 2 = 0$$

A másodfokú egyenlet általános alakja: $ax^2 + bx + c = 0$. A megoldása pedig:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



Ábrázoljuk az első egyenletet függvényként az **fplot** használatával! Hozzuk létre a gyak2.m script fájlt az aktuális könyvtárba!

```
clc; clear all; close all;
a = 2, b = -1, c = -3
f = @(x) a*x.^2+b*x+c;
figure; fplot(f);
```

A megoldásnál figyelembe kell vennünk, hogy hány megoldásunk van! Nézzük meg a következő saját függvényt (masodfoku.m), ami megvizsgálja az $a\,x^2 + b\,x + c = 0$ alakú másodfokú egyenlet megoldhatóságát, ábrázolja a függvényt és ha van megoldás, akkor megadja azokat! Ehhez a $D = b^2 - 4ac$ diszkrimináns lehetséges eseteit kell megvizsgálni. Mentsük el a fájlt az aktuális könyvtárba.

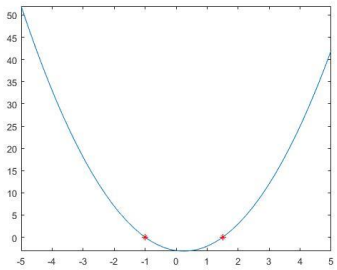
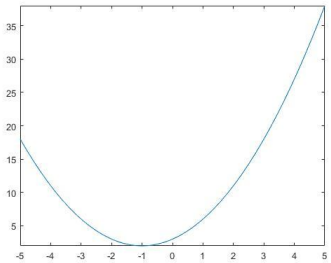
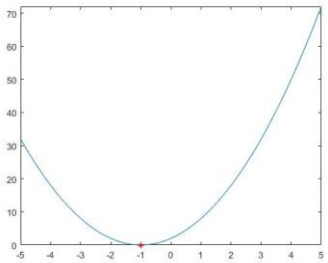
```
function x = masodfoku(a,b,c)
% Az a*x^2+b*x+c=0 egyenlet megoldása, % input: a,b,c
    f = @(x) a*x.^2+b*x+c;
    figure; fplot(f);
    D = b^2-4*a*c; % diszkrimináns
    if D>0
        disp('Az egyenletnek 2 megoldása van')
        x(1) = (-b+sqrt(D))/(2*a);
        x(2) = (-b-sqrt(D))/(2*a);
        hold on; plot(x,[0,0],'r*')
    elseif D==0
        disp('Az egyenletnek csak 1 megoldása van')
        x = -b/(2*a);
        hold on; plot(x,0,'r*')
    else
        disp('Az egyenletnek nincs megoldása')
        x = [];
    end
end
```

A **disp** utasítás csak kiír egy szöveget a képernyőre. A függvényeket nem tudjuk önmagukban futtatni, hanem parancssorból, vagy script fájlból hívhatjuk meg őket. Oldjuk meg az első egyenletet parancssorból meghívva a masodfoku függvényt! (Ezt akkor tehetjük meg, ha a fájl abban a könyvtárban van ahová dolgozunk.)

```
> masodfoku(2,-1,-3)
```

Célszerűbb azonban script fájlba dolgozni, amit utólag is könnyen lehet módosítani. Folytassuk a gyak02.script fájlt!

```
> %% Elágazások - IF
> disp('Másodfokú egyenlet megoldása: ax^2+bx+c=0')
> a = 2, b = -1, c = -3,
> x = masodfoku(a,b,c)
> a = 1, b = 2, c = 3,
> x = masodfoku(a,b,c)
> a = 2, b = 4, c =2,
> x = masodfoku(a,b,c)
```

Megjegyzés: a legújabb Matlab *.m fájlok esetében a használt függvényeket bemásolhatjuk a script fájl végére is, nem csak külön függvényként fájlba elmentve hívhatóak meg. Ebben az esetben viszont egy szakasz futtatásakor nem találja meg a függvényt a Matlab, csak a teljes script futtatásakor.

TÖBBIRÁNYÚ ELÁGAZÁS - SWITCH, CASE

Nem csak kétirányú elágazásokat használhatunk, hanem többirányúakat is bonyolultabb esetekben. Írjunk egy programot, ami segít egy tanárnak véletlenszerűen osztályozni! A **randi(n)** paranccsal véletlen egész számokat generálhatunk 1 és n között. Dupla %% jeleket használva megjegyzésként, amit utána írtunk külön szakaszba fog kerülni, és CTRL+Enter megnyomásával többször is lefuttatható. Lett jeles 3 futtatásból?

```
> %% Elágazások - SWITCH
> disp('Vizsgajegy:')
> jegy = randi(5);
> switch jegy
>   case 1
>     disp('Elégtelen')
>   case 2
>     disp('Elégséges')
>   case 3
>     disp('Közepes')
>   case 4
>     disp('Jó')
>   case 5
```

```
>       disp('Jeles')
>   end
```

SZÁMLÁLÁSSAL VEZÉRELT CIKLUS - FOR

Egy ciklusban parancsok csoportját hajtjuk végre ismétlődően. A **for** számlálással vezérelt ciklusban előre meghatározott számban történik az ismétlés. Szerkezete Matlabban:

for *i = f:s:t*

(Matlab utasítások)

end

Ahol *i* a ciklus index változója, *f* az index változó értéke az első ismétléskor, *s* az index növekménye minden ismétlés után, *t* pedig az utolsó értéke. Nézzük meg hogyan oldhatjuk meg az első példában leírt egyenleteket ciklus használatával, ha az együtthatókat egy mátrixban tároljuk!

```
>   %% Ciklusok - FOR
>   close all; clc; clear all;
>   disp('Oldja meg a 2x^2-x-3=0, x^2+2x+3=0, 2x^2+4x+2=0 egyenleteket!')
>   M = [2,-1,-3;
>       1,2,3;
>       2,4,2]
>   n = size(M,1) % sorok száma
>
>   for i = 1:n
>       a = M(i,1), b = M(i,2), c = M(i,3),
>       masodfoku(a,b,c)
>   end
```

A **size(M)** függvény megadja egy mátrix méretét, két kimenettel: sorok és oszlopok száma, a **size(M,1)** a sorok számát adja vissza, a **size(M,2)** pedig az oszlopok számát. Van még két hasonló függvény, a **length** egy vektor elmeinek a számát adja vissza, vagy a mátrix nagyobbik méretét, a **numel** pedig az összes elem számát a mátrixban/vektorban.

FELTÉTELLEL VEZÉRELT CIKLUS - WHILE

A while ciklus egy feltétellel vezérelt ciklus, ahol a ciklus törzsében lévő parancsok addig fognak ismétlődni, amíg a megadott feltétel igaz. Szerkezete Matlabban:

while (feltételes kifejezés)

(matlab utasítások)

end

Nézzük a következő kitalált példát: egy tantárgyból véletlenszerűen történik a pontozás a vizsgán 0-100 között. 88 % felett jeles az osztályzat. Addig próbálkozunk a vizsgával, amíg ötöst nem kapunk. Írjunk egy programot, ami véletlenszerűen előállítja az egyes vizsgákra kapott osztályzatainkat. Hányadik vizsgára kaptunk ötöst?

```
> %% Ciklusok - WHILE
> disp('Véletlenszerű pontozás esetén hányadik vizsga lesz 88%
  felett?')
> i = 0; pont = 0;
> while pont<=88
>     i=i+1;
>     pont = rand()*100
> end
> i
```

FORMÁZOTT SZÖVEGEK (FPRINTF, SPRINTF)

Gyakran van szükség arra, hogy az eredményeinket egy adott formátumban jelenítsük meg. Nézzük például a szögekkel való műveleteket. A legtöbb matematikai művelet végzésére alkalmas szoftver (pl. Matlab, Octave, Excel…) a szögeknél a radiánt tekinti alapértelmezettnek, ha ettől eltérő formátumban szeretnénk látni az eredményeket, akkor nekünk kell erről gondoskodni. Matlab és Octave alatt a radiánban értelmezett trigonometriai függvényeknek (pl. sin, cos, tan, atan, atan2…) megvannak a fokokkal számoló változatai (pl. sind, cosd, tand, atand, atan2d…), de ha már fok-perc-másodpercben szeretnénk megjeleníteni az eredményeket, mondjuk a geodéziában használt formátumban (pl 302-06-23), esetleg adott számú tizedesjegyre (23-03-48.5831), akkor ezt csak a formázott szövegekkel tehetjük meg. Ugyanígy, ha pl. képeket szeretnénk automatikusan elnevezni egy ciklusban pl. IMG0001.jpg, IMG0002.jpg stb. akkor ehhez is a formázott szövegeket hívhatjuk segítségül, ahol pl. a számokat vehetjük egy változóból és megadhatjuk hozzá a megjelenítést.

Az **`fprintf`** paranccsal fájlba és képernyőre is írhatunk formázott szövegeket, az **`sprintf`** használatával pedig egy stringbe (szöveges változóba)/képernyőre. Mindig egy **%** jel jelzi, hogy most egy formázott változó következik a szövegben, ahány **%** jel szerepel a szövegben, annyi formázott változónk lesz. A szöveg után vesszővel kell megadni a változókat, olyan sorrendben, ahogy a szövegben szerepeltek.

A következő formátumjelölőket alkalmazhatjuk:

- **%d** – egész szám, **%s** – szöveg, **%f** – valós szám (lebegőpontos), **%c** – karakter, **%u** – nem előjeles egész
- **%e** – normál alak pl. 3.14e+00, **%E** – 3.14E+00
- **%g** – kompakt forma, **%f** vagy **%e** közül a rövidebbik, fölös 0-k nélkül

A típust jelző betű előtt szerepelhet még pl. **+** jel, akkor előjelesen írja ki a számot; mező szélesség; tizedesjegyek száma; **0**, akkor 0-kal tölti fel elől az üres helyeket a mező szélességéig. Próbáljuk ki a következőket!

```
> clc; disp('Hány éves a kapitány?')
> % Octave-ban nincs datetime, between parancs!
> % in Octave use this: ev = 35; ho = 5; nap = 2;
> szul = datetime(1984,02,28)
> most = datetime('now')   % (2019.07.30-án)
> kor = between(szul, most) % 35y 5mo 2d 13h 58m 47.086s
> [ev,ho,nap] = datevec(kor) % ev = 35  ho = 5  nap =  2
> evt = ev + ho/12 + nap/365;
> fprintf('A kapitány 35 éves')    % nem rak be sortörést a végére
> fprintf('A kapitány 35 éves\r\n')% \r\n - sortörés
```

```
> sprintf('A kapitány 35 éves')   % szöveges változóba teszi az
  eredményt (ans)
> sprintf('A kapitány %d éves, %d hónapos és %d napos',ev,ho,nap) %  'A
  kapitány 35 éves, 5 hónapos és 2 napos'
> sprintf('A kapitány %f éves', evt) % 'A kapitány 35.422146 éves'
> sprintf('A kapitány %.2f éves', evt) % 'A kapitány 35.42 éves'
> sprintf('A kapitány %8.2f éves', evt) % 'A kapitány    35.42 éves'
> sprintf('A kapitány %08.2f éves', evt) % 'A kapitány 00035.42 éves'
> sprintf('A kapitány %+6.2f éves', evt) % 'A kapitány +35.42 éves'
```

A `%+6.2f` kifejezés azt jelzi, hogy 6 karakteren írja ki a változót (tizedespontot, előjelet is beleértve!), egy valós számot (`f`), ahol a tizedesjegyek száma 2 (`.2`), és kiírja az előjelet is (`+`). Ha `0` is szerepel benne, akkor az üres helyeket 0-kkal tölti ki. (`0`). Ha az eredmény hosszabb lenne, mint a mező szélessége, akkor nem veszi figyelembe a megadott mező szélességet.

Nézzük meg a következő függvényt, ami a tizedfokban megadott mérési/számítási eredményeket a geodéziában szokásos fok-perc-másodperc formában írja ki. A perc, másodperc értékét mindig két számjeggyel kell kiírni pl. 192-03-12!

```
> function str = fpm(x);
> % A függvény tizedfokból fok-perc-másodperc értékekbe számol át.
> % A kimenet egy formázott szöveg (ddd-mm-ss)
>   f = fix(x);
>   p = fix((x-f) .* 60);
>   m = round(((x-f).*60-p).*60);
>   str = sprintf('%3d-%02d-%02d', f, abs(p), abs(m));
> end
```

A **fix** függvény mindig a 0 felé kerekít (ez a negatív szögek miatt fontos), a **round** matematikailag kerekít, a **floor** lefelé, a **ceil** pedig felfelé kerekítene. A végén a perceknek és a másodperceknek az abszolút értékét vesszük, hogy a negatív előjelet oda már ne írja ki. Próbáljuk ki!

```
> a = 123.123, b = -123.123
> fpm(a)  % '123-07-23'
> fpm(b)  % '-123-07-23'
```

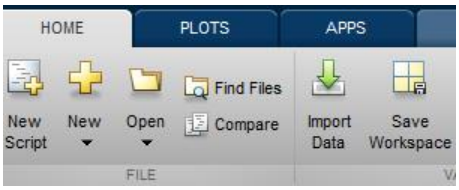
Cseréljük le a fokok (f), percek (p) számításánál a **fix** parancsot **floor**-ra, mentsük el és futtassuk le újra az előbbieket. Mi történik?

FÁJLMŰVELETEK

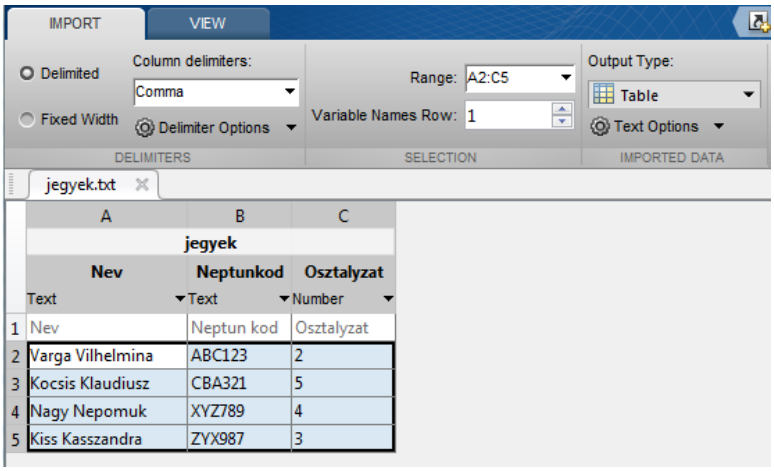
A mérnöki munkák során gyakran előfordul, hogy egy műszeres mérés eredményeit kell feldolgozni és ezeket például egy szöveges fájlban kapjuk meg valamilyen formátumban. Ha Matlabbal szeretnénk a méréseket feldolgozni, akkor be kell tudjuk olvasni ezeket az adatokat. A beolvasás a formátum függvényében többféleképpen történhet. A feldolgozás eredményét sokszor szintén egy adott formában kell elmenteni a további használathoz. Erre nézünk most példákat, hogy kicsit ismerkedjünk a fájlműveletekkel.

IMPORT DATA TOOL, TABLE, STRUCTURE, CELL ARRAY ADATTÍPUSOK

Az egyik eszköz, amit fájlok importálására használhatunk, az a Matlab saját import eszköze, ami a Home fül 'Import Data' gombjára kattintva érhető el. Olvassuk be a jegyek.txt fájl tartalmát ezzel az eszközzel!



A használata elég egyszerű, csak a beállításokra kell odafigyelni. Megadhatjuk, hogy mekkora tartományban vannak az adataink, fix szélességű oszlopokban vannak-e, vagy adott karakterrel elválasztva. Amire még fontos figyelni, az a kimenet típusa (Output type), ami alapértelmezett esetben Table. Lehet más típusokat is választani, pl. Cell array, Numeric matrix. Hagyjuk most az alapértelmezett Table típuson és olvassuk be az adatokat a zöld pipára kattintva. Utána bezárhatjuk az import ablakot.




```
> %% 'Import Data' tool
> % jegyek.txt -> table forma
> clc; jegyek
> %   4x3 table
> %            Nev              Neptunkod    Osztalyzat
> %     ____________________    _________    __________
> %     'Varga Vilhelmina'     ' ABC123'    2
> %     'Kocsis Klaudiusz'     ' CBA321'    5
> %     'Nagy Nepomuk'         ' XYZ789'    4
> %     'Kiss Kasszandra'      ' ZYX987'    3
```

Ezek az adatok 'Table', táblázat típusúak lesznek, amiben egyszerre lehet különböző típusú adatokat tárolni, szöveget is, számokat is (akárcsak a Structure és a Cell array típusnál). Az egyes oszlopokra változó nevekkel lehet hivatkozni a táblázat neve és egy pont után megadva a változó nevét.

```
> jegyek(1:2,1:3)
> nev = jegyek.Nev % cella tömb
> oszt = jegyek.Osztalyzat % szám vektor
```

Ehhez hasonló forma még a 'structure' adattípus, ott is nevekkel hivatkozhatunk egy-egy mezőre. A struktúra típus esetén nem kötelező, hogy minden változó (mező) esetén ugyanannyi sor/adat legyen, mint a táblázatnál. A cellatömbben (Cell array) is különböző típusú adatokat tárolhatunk, de ott nincs névvel ellátva semmi, hivatkozni az egyes elemire hasonlóan lehet, mint a mátrixoknál, csak kapcsos zárójeleket

használva {}. Például a nevek egy cellatömbbe kerültek tárolásra. Kérdezzük le a másodikat!

```
> nev2 = nev{2}  % 'Kocsis Klaudiusz'
```

EGYSZERŰ ADATBEOLVASÁS/KIÍRÁS (LOAD, SAVE)

Nézzünk most egy mérnöki példát, ismét a betonacél fajlagos alakváltozásához (ε) tartozó feszültségekről (σ)! A feszültség-fajlagos alakváltozás (σ-ε) diagram adatait a következő táblázat tartalmazza:

ε [%]	0	0.2	2	20	25
σ [N/mm^2=Mpa]	0	300	285	450	350

1. TÁBLÁZAT BETONACÉL FESZÜLTSÉG-FAJLAGOS ALAKVÁLTOZÁS DIAGRAMJA

Feladatunk egy táblázat előállítása, ami 0-tól 25%-ig 0.1 százalékonkénti alakváltozásokhoz tartalmazza a feszültségeket. Most nem kézzel fogjuk bevinni az adatokat, hanem beolvassuk a betonacel.txt fájlból:

```
0     0
0.2   300
2     285
20    450
25    350
```

Természetesen itt is beolvashatnánk az adatokat az import data eszköz segítségével, csak oda kell figyelni a beállításokra, hogy szóközzel tagolt fájl legyen és a kimenet pedig egy mátrix (Numeric matrix).

Ez az állomány 5 sorban és két oszlopban tartalmaz számokat. Szabályos, minden sorban ugyanannyi elemet tartalmazó, csak számokból álló szöveges fájlokat nagyon egyszerű Matlab-ba beolvasni a **load** parancsot használva. Persze az esetek egy nagy részében nem ilyen egyszerű formában kapjuk a mérési eredményeket, lehet, hogy minden sor más hosszúságú, szöveges elemek is lehetnek benne, ezekhez más parancsokat kell használni. Bonyolultabb formátum esetén sokszor érdemes lehet soronként beolvasni az adatokat és úgy feldolgozni a sorokat.

Nézzük most a legegyszerűbb adatbeolvasásra/kiírásra szolgáló **load** illetve **save** parancsokat. Az aktuális könyvtárba másoljuk be a betonacel.txt fájlt, majd töltsük be a tartalmát. Ezt kétféleképpen tehetjük meg, parancsként meghívva (ekkor nem kell idézőjeleket használni):

```
> load betonacel.txt
```

vagy függvényként meghívva a load parancsot:

```
> adat = load('betonacel.txt')
```

Az első esetben létrejön egy 'betonacel' nevű változó és oda menti a tartalmat. A második változatban a **load**-t függvényként meghívva hozzárendelhetjük az eredményt egy változóhoz. Használjuk most ezt a módszert. Nézzük meg az eredményül kapott adat nevű változó méretét! Megnézhetjük a workspace-ben a méretét (**size**), típusát stb. Le is kérdezhetjük a **whos**… paranccsal a változó főbb tulajdonságait, vagy használhatjuk a **size** parancsot is.

```
> whos adat
> size(adat)
```

Egy 5x2 méretű, vagyis 5 sorból és 2 oszlopból álló mátrixot kaptunk.

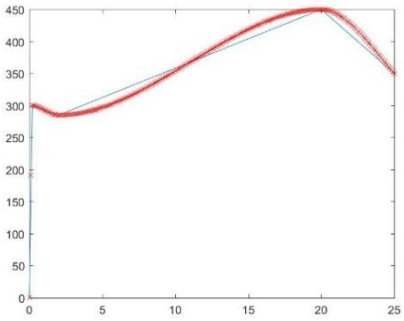
Ábrázoljuk (σ-ε) diagramon a beolvasott értékeket, ehhez válasszuk szét a változókat (most legyen x az alakváltozás, y a feszültség)!

```
> x = adat(:,1); % első oszlop
> y = adat(:,2); % második oszlop
> plot(x,y);
> xlabel('\epsilon');ylabel('\sigma');
```

Oldjuk meg az eredeti feladatot, vagyis 0.1 százalékonként minden fajlagos alakváltozáshoz adjuk meg a hozzá tartozó feszültség értékeket! Ehhez interpolációra lesz szükségünk. Most köbös, elsőrendű spline interpolációt fogunk használni, ahol a pontokban az illesztett görbéknek az első deriváltjai is megegyeznek. Először sűrítsük be a pontokat 0-25% (maximális alakváltozás) között 0.1%-ként, majd számoljuk ki interpolációval ezekben a pontokban a feszültség értékeket, az **interp1** parancs 'pchip' (köbös, elsőrendű) módszerét használva!

```
> % köbös elsőrendű spline interpoláció
> xi = 0:0.1:max(x); % pontok sűrítése
> yi = interp1(x,y,xi,'pchip'); % interpoláció
```

Rajzoljuk be az előző ábrába az interpolációval besűrített pontokat is! Ha ugyanabba az ábrába akarunk rajzolni, mint az előbb, akkor ki kell adni a '**hold on**;' parancsot (különben felülírja az előző ábránkat). Ezt elég ábránként egyszer kiadni, ha mégis később felül akarjuk írni az ábrát, akkor 'hold off;' paranccsal megszüntethetjük a hatását.



1. ÁBRA BETONACÉL FAJLAGOS ALAKVÁLTOZÁS-FESZÜLTSÉG ÁBRÁJA

```
> hold on;
> plot(xi,yi,'rx'); % 'rx' - piros x
```

Ha el akarjuk menteni esetleg illusztráció céljára az eredményt, akkor megtehetjük akár a megnyílt grafikus ablakból, akár a matlab **print** parancsát használva, megadva a kép nevét és a típusát is:

```
> print('betonacel.jpg','-djpeg')
```

Ha megnézzük xi és yi változókat, akkor látjuk, hogy ezek 1x271 méretű sorvektorok. Szeretnénk ezeket egy táblázatban elmenteni ahol az első oszlopban az elmozdulás a másodikban pedig az alakváltozás van. Ehhez transzponálni kell a sorvektorokat (') és utána egy egyszerű mátrix művelettel összefűzhetjük őket, mivel megegyező a méretük:

```
> adat2 = [xi' yi'];
```

Az elmentéshez használhatjuk a **save** parancsot. Ez alapértelmezésben a matlab saját bináris *.mat kiterjesztésbe menti a fájlokat, amit más programba nem tudunk beolvasni, de matlabba a **load** paranccsal a későbbiekben bármikor betölthető az állomány.

```
> save('betonacel2.mat','adat2')
```

Ha szöveges fájlba szeretnénk menteni, akkor meg kell adni még az **'-ascii'** opciót is.

```
> save('betonacel2.txt','adat2','-ascii');
```

Megjegyzés: a **save** is kiadható függvény és parancs formátumban is. Parancsként egyszerűbb (lásd lent), nem kell idézőjeleket használni, viszont kevésbé rugalmas a meghívás, nem vehetjük pl. a fájlnevet egy szöveges változóból.

```
> save betonacel2 adat2
> save betonacel2.txt adat2 -ascii
```

Nyissuk meg az elmentett szöveges állományt!

```
0.0000000e+00   0.0000000e+00
1.0000000e-01   5.2521666e+01
2.0000000e-01   1.0943166e+02
3.0000000e-01   1.6158083e+02
4.0000000e-01   1.9982000e+02
…
```

Sima **save** parancsot használva normál alakban menti a számokat, ha hagyományos formában szeretnénk megjeleníteni, pl. előre megadott 1 vagy 2 tizedesjegyre, akkor más módon kell mentenünk, formázott szövegként.

FORMÁZOTT KIÍRÁS FÁJLBA (FPRINTF)

Írjuk ki a tized százalékonként megadott fajlagos alakváltozás - feszültség értékeket hagyományos szám formátumba, egy tizedesre az alakváltozást és kettőre a hozzájuk tartozó feszültségeket. Ehhez szükségünk lesz az alap fájlkezelő utasításokra, mint fájl megnyitás, írás, lezárás és a korábban áttekintett formázott szövegek írására. Az alap fájlkezelő utasítások általánosan a következőképp néznek ki:

- fájl megnyitása (**fopen**)
- beolvasás, írás, hozzáfűzés a fájlhoz
- fájl bezárása (**fclose**)

Az **fopen** használata során megadhatjuk, hogyan kívánjuk megnyitni a fájlt, 'r'-csak olvasásra (alapértelmezett, ha nem adunk meg semmit), 'w'-írásra, 'a'-hozzáfűzéshez:

`fileID = fopen(filename,'w')` – fájl megnyitásra írásra

A fájlokat bezárhatjuk egyenként: **fclose**(fileID), vagy egyszerre az összeset: **fclose**('all').

Írjuk ki az adatokat fájlba egy számlálással vezérelt **for** ciklussal! 4 karakteren egy tizedesre az alakváltozást és 6 karakteren 2 tizedesre a feszültséget, köztük egy szóközzel! A **length** parancs egy adott vektor hosszát adja vissza.

```
> n = length(xi); % vektor hossza
> fid = fopen('tablazat.txt','w');
> for i=1:n
>     fprintf(fid,'%4.1f %6.2f\r\n',xi(i),yi(i));
> end
> fclose(fid);
```

A feladat egyébként megoldható ciklus nélkül is az adat2 változót használva:

```
> fid = fopen('tablazat2.txt','w');
> fprintf(fid,'%4.1f %6.2f\r\n',adat2');
> fclose(fid);
```

```
> type tablazat2.txt % kiírja a képernyőre a fájl tartalmát
```

Az adat2 változó 2 oszlopban 271 sorban tárolta az összetartozó adatokat, a formázott kiíráshoz azonban a transzponáltját vettük (2 sor 271 oszlop), mivel az fprintf oszloponként olvassa be az összetartozó értékeket.

SORONKÉNTI BEOLVASÁS (FGETL, FGETS)[1]

A mérnöki munkák során előfordul, hogy egy adott műszer méréseit kell feldolgozni, amelyekben nem csak számok, hanem szövegek is előfordulnak. A feldolgozáshoz be kell tudjuk olvasni ezeket az adatokat és ki kell tudni válogatni belőle a minket érdeklő részt. Nézzünk most erre egy navigációs példát!

Következő példában egy GPS-szel rögzített útvonal adatait kaptuk meg, amit a navigációban használt NMEA 0183 formátumban rögzítettek (hb_nmea.txt). Olvassuk be az adatokat és ábrázoljuk az útvonalat. Vajon milyen járművel rögzíthették az adatokat?

```
$GPGLL,5156.9051,N,00117.1178,E*69
$GPGLL,5156.9194,N,00117.1482,E*61
…
```

Az NMEA szabványban a sor elején a $GPGLL azt jelenti, hogy GPS Geographic Latitude, Longitude, vagyis csak földrajzi szélesség és hosszúság információkat tartalmazó adat (sokféle NMEA üzenet létezik). Meghatározott hosszúságú mezők vannak vesszővel elválasztva, elég könnyen beolvasható, feldolgozható fájl. A földrajzi szélességnél az első két karakter a fok érték, utána perc, a hosszúságnál az első 3 karakter a fok érték, utána perc (mivel előbbi ±90°, utóbbi ±180°-ig terjed). Szélességnél N (Észak), S (Dél), hosszúságnál E (Kelet), W (Nyugat). Pl. 5156.9051,N azt jelenti, hogy északi szélesség 51°56.9051'.

Ez már egy bonyolultabb szerkezetű fájl, nem tudjuk sima **load** paranccsal beolvasni. A beolvasás ebben az esetben hasonló az előbbi fájlba íráshoz, először meg kell nyitni a fájlt (**fopen**), utána jöhetnek a beolvasási műveletek és utána le kell zárni a fájlt (**fclose**). A beolvasáshoz ebben az esetében hasznos lehet ismerni a soronkénti fájlbeolvasás parancsait: **fgetl**, **fgets**. Az **fgetl** beolvas egy sort és levágja belőle a sorvége karaktert, míg az **fgets** megtartja. A beolvasás eredménye egy string változóba kerül. Az egész fájl tartalom beolvasásához egy feltételes ciklusra lesz szükség (**`while`**), hogy addig olvasson, amíg el nem érünk a fájl vége jelhez (**`feof`** - end-of-file).

Először csak olvassunk be egy sort és próbáljuk meg kinyerni belőle a minket érdeklő adatokat, hogy ábrázolni tudjuk. Megj.: A fájl megnyitása után egy fájl pointer figyeli, hogy épp hányadik bájtig olvastuk be a fájlt, amit akár le is kérdezhetünk az **ftell**(fid) paranccsal.

```
> fid=fopen('hb_nmea.txt');
> line=fgetl(fid) % egy sor beolvasása
> % $GPGLL,5156.9051,N,00117.1178,E*69
```

Az eredmény egy szöveges változó lesz, ami az első sort tartalmazza. Szűrjük ki belőle a minket érdeklő információkat, a földrajzi szélesség (fi) és hosszúság adatokat

[1] Kiegészítő anyag

(lambda)! Ehhez tudni kell, hogy a 8-9. karakter a fi fok értéke, a 10-16. karakter a percé, 20-22. a lambda fok, 23-29 lambda perc értéke. Adott karakterek egy szövegből ugyanúgy válogathatóak le, mint a vektorok elemei, ugyanis ezek is karakter vektorok a Matlabban!

```
> fi_fok = line(8:9);   fi_perc = line(10:16);
> lambda_fok = line(20:22);  lambda_perc = line(23:29);
```

Számoljuk át az értékeket tizedfokba! Ehhez először szövegből számmá kell alakítanunk fi-t, lambdát a **str2num** paranccsal.

```
> fi = str2num(fi_fok)+str2num(fi_perc)/60  % 51.9484
> lambda = str2num(lambda_fok)+str2num(lambda_perc)/60  % 1.2853
```

Olvassuk be, hogy melyik féltekén van a koordináta: É-i vagy D-i szélesség (18. karakter), K-i vagy Ny-i hosszúság (31. karakter). A D-i szélesség vagy Ny-i hosszúság értékeket negatív koordinátákkal adhatjuk meg. Egy **if** feltételes művelettel változtassuk meg az előjelet, ha szükséges!

```
> NS = line(18);   if NS=='S'; fi=fi*-1; end;
> EW = line(31);   if EW=='W'; lambda=lambda*-1; end;
```

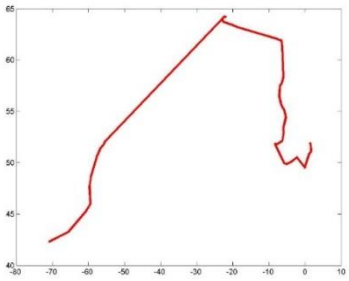
Így lehet például egy sorból kinyerni egy bonyolultabb szerkezet esetén a minket érdeklő információt. Persze rengeteg egyéb kiíró, beolvasó művelet létezik még matlabban (input/output functions), ezeket részletesebben megnézhetjük a **help iofun** parancs kiadásával.

Most olvassuk be az egész fájlt egyben. Ehhez feltétellel vezérelt ciklusra lesz szükség (**while** ciklus). Jelen esetben a feltétel az lesz, hogy elértük-e már a fájl végét? Vagyis feof(fid)==0-e (**feof** = end of file). Szükség lesz még két vektor változóra (lat, long), amikbe a beolvasott koordinátákat tesszük. Ezeket az elején inicializálni kell, vagyis üres vektorokként megadni, és a ciklusban egyszerű vektor művelettel mindig hozzáfűzzük az újabb értékeket. A sorok végére tegyünk pontosvesszőket, hogy ne írja ki mindig az összes részeredményt! Az egész egyben a következőképpen néz ki:

```
> lat = []; long = [];
> fid=fopen('hb_nmea.txt');
> while feof(fid)==0
>   line=fgetl(fid); % egy sor beolvasása
>   % fi, lambda kiszűrése
>   fi_fok = line(8:9);   fi_perc = line(10:16);
>   lambda_fok = line(20:22);  lambda_perc = line(23:29);
>   % Átszámítás tizedfokba
>   fi = str2num(fi_fok)+str2num(fi_perc)/60;
>   lambda = str2num(lambda_fok)+str2num(lambda_perc)/60;
>   % előjelek
>   NS = line(18);   if NS=='S'; fi=fi*-1; end;
>   EW = line(31);   if EW=='W'; lambda=lambda*-1; end;
>   % eredmények összefűzése
>   lat = [lat; fi]; long = [long; lambda];
> end
> fclose(fid);
```

Ábrázoljuk az útvonalat egy új ábrán vastag piros vonallal!

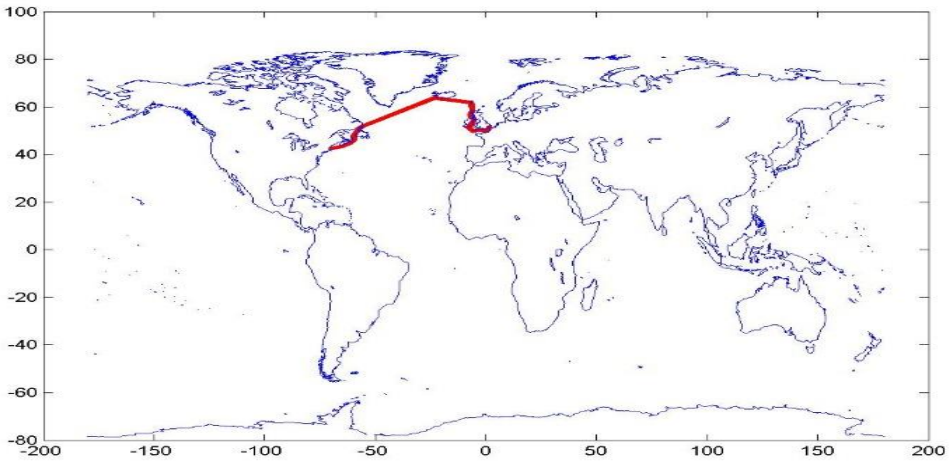
```
> figure(2)
> plot(long, lat,'r','LineWidth',3)
```



Az ábra alapján még nehéz lenne eldönteni, hogy merre is ment a jármű, a könnyebbség kedvéért ábrázoljuk a partvonalakat is kékkel. Ehhez töltsük be a partvonal.txt állományt!

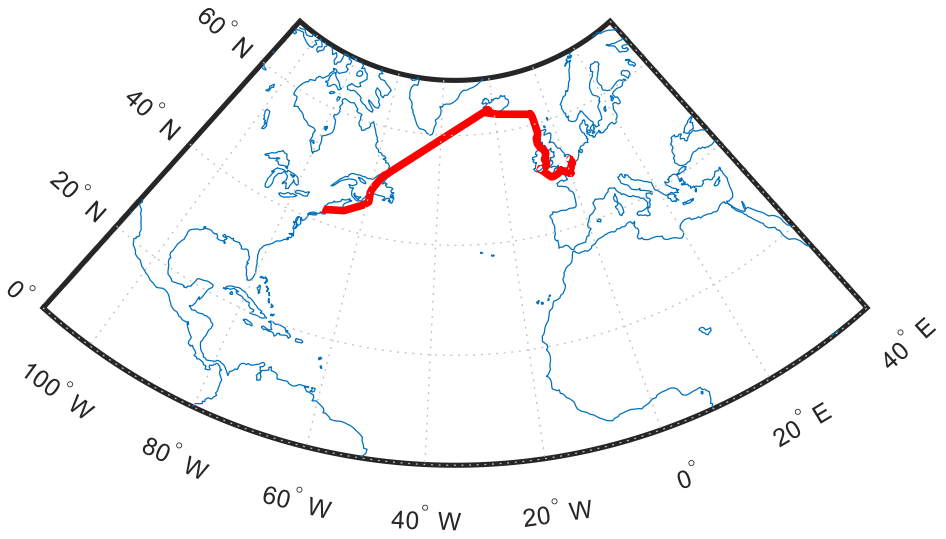
```
> part = load('partvonal.txt');
> hold on; plot(part(:,1),part(:,2),'b')
```

Milyen járműről lehetett szó?



Ábrázolás másképp Matlabban:

```
> figure
> worldmap([0 70],[-110 40])  % worldmap('World')
> load coastlines
> plotm(coastlat,coastlon)
> hold on
> plotm(lat, long,'r','LineWidth',3)
```



A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

==	-	Logikai egyenlőség
~=	-	Logikai ’nem egyenlő’
&&	-	Logikai ’és’
||	-	Logikai ’vagy’
disp	-	Szöveg, változók tartalmának kiírása a Command Window-ba
if, elseif, else, end	-	Kétirányú feltételes elágazás
switch, case	-	Többirányú elágazás
for	-	Számlálással vezérelt ciklus
while	-	Feltétellel vezérelt ciklus
size	-	Mátrix sorainak, oszlopainak száma
length	-	Vektor elemeinek száma, vagy mátrix nagyobbik mérete
numel	-	Mátrix/vektor összes elemszáma
randi	-	Véletlen egész számok generálása
fprintf	-	Fájlba és képernyőre is írhatunk formázott szövegeket
sprintf	-	String típusú (szöveges) változóba/képernyőre írhatunk formázott szövegeket
\r\n	-	Sorvége jel a formázott szövegeknél
fix	-	Kerekítés mindig a 0 felé
round	-	Kerekítés matematikai értelemben
floor	-	Kerekítés lefelé
ceil	-	Kerekítés felfelé
load	-	Adatok betöltése (Matlab adatállományból (*.mat), egyszerű szöveges fájlból)
save	-	Adatok elmentése (Matlab adatállományba (*.mat), egyszerű szöveges fájlba)
print	-	Ábra elmentése fájlba
interp1	-	Egyváltozós interpoláció
fopen	-	Fájl megnyitása
fclose	-	Fájl bezárása
type	-	Szöveges fájl tartalmának kilistázása a Command window-ba
fgetl	-	Beolvas egy sort és levágja belőle a sorvége karaktert.
fgets	-	Beolvas egy sort, megtartja a sorvége karaktert is.
feof	-	Fájl vége jel (end-of-file)
ftell	-	Pointer, hogy hol tart a fájl beolvasása
str2num	-	Szövegből számmá alakít
atan, atan2	-	Inverz tangens függvény, eredmény radiánban.
sind, cosd, tand, atand, atan2d	-	Trigonometrikus függvények fok mértékegységgel