ADATOK BEOLVASÁSA/KIÍRÁSA FÁJLBA

Az előző órán láttuk, hogy load és save használatával be lehet olvasni, illetve el lehet menteni szöveges állományokat. A szöveges fájl beolvasásánál azonos típusú adatok és azonos sorhosszak szükségesek. Az eredmény egy mátrixba kerül, amiben csak azonos típusú elemek lehetnek. A **save** paranccsal elmenthetjük a munkakörnyezet összes vagy néhány változóját *.mat kiterjesztésű MATLAB bináris fájlba, amiből utána **load**-dal ezek betölthetőek, de ez más program számára nem igazán értelmezhető formátum. Ha szöveges állományba kívánjuk menteni egy mátrix tartalmát, akkor a **save filename variables -ascii** paranccsal tehetjük meg. Ha egy létező fájlhoz akarunk hozzáírni valamit, akkor a **save filename variables -ascii -append** parancsot használhatjuk. Formázott szövegeket az **fprintf** használatával írhatunk ki. Mi helyzet azonban akkor, amikor ennél bonyolultabb szerkezetű fájlt kell beolvasni? Nem azonos típusú adatok szerepelnek benne, nem azonos az egy sorban lévő adatok száma? Ilyenkor más megoldást kell keresnünk a beolvasásra.

SORONKÉNTI BEOLVASÁS (FGETL, FGETS)

Az **fgetl** és **fgets** parancsokkal soronként lehet beolvasni egy fájl tartalmát. Az **fgetl** levágja belőle a sorvége karaktert (\n vagy \r\n)[1], míg az **fgets** megtartja. A beolvasás eredménye egy string változóba kerül. Az egész fájl tartalom beolvasásához egy feltételes ciklusra van szükség (**while**), hogy addig olvasson, amíg el nem érünk a fájl vége jelhez (**feof** - end-of-file).

Olvassuk be a következő állományt, amiben betűk és számok is vannak, szambetu.dat:

```
5.3 a
2.2 b
3.3 a
4.4 a
1.1 b
```

Először csak nyissuk meg az állományt és olvassunk be két sort. 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.

```
> clear all;clc;
> type szambetu.dat
> fid=fopen('szambetu.dat');
> line=fgetl(fid) % egy sor beolvasása
> line=fgetl(fid) % egy sor beolvasása
> fclose(fid);
```

Jó lenne ezt egy ciklusba tenni, ami addig futna, amíg a fájl végére nem érünk, akkor nem kell tudni előre hány sort olvassunk be! Használjunk ehhez egy feltétel vezérelt ciklus, amíg a fájl végére nem érünk (feof igaz nem lesz).

```
> fid=fopen('szambetu.dat');
```

[1] A sor vége jel Windows esetében: \r\n, Mac (OS 9-) esetében \r, Unix/Linux esetében: \n.

```
> while feof(fid)==0
>   line=fgetl(fid) % egy sor beolvasása
> end
> fclose(fid);
```

Most minden új sor beolvasásakor felülírjuk az előzőt. Jó lenne elmenteni külön egy mátrixba a számokat és külön a betűket! Nézzük meg hogyan tudjuk szétválasztani őket!

```
> szam = str2num(line(1:3))
> betu = line(5)
```

Ez az egyik lehetőség, ha ismerjük, hogy hány karakter a szám az elején és hol van a betű, a másik lehetőség, hogy az első szóköz mentén szétvágjuk a szöveget. Van még sok szöveges állomány kezelő parancs, amiket érdemes lehet nézegetni, ha ilyen feladata van az embernek (lásd pl. az Octave Documentation fülnél a string menü, vagy https://www.mathworks.com/help/matlab/characters-and-strings.html).

```
> [szam betu] = strtok(line)
> szam = str2num(szam)
```

Válasszuk ki az egyik megoldást és a ciklusban fűzzük össze egy-egy tömbbe a számokat, betűket!

```
> szamok=[];betuk=''; % számok és betűk tömbök létrehozása
> fid=fopen('szambetu.dat');
> % Sorok beolvasása, számok, betűk szétválasztása
> while feof(fid)==0 % ciklus, amíg a fájl végére érünk
>    line=fgetl(fid); % egy sor beolvasása
>    % szöveg szétválasztása számra, betűre
>    szam = str2num(line(1:3))
>    betu = line(5)
>    szamok=[szamok;szam]; % szam hozzáfűzése a szamok tömbhöz
>    betuk=[betuk;betu]; % betű hozzáfűzése a betuk tömbhöz
> end
> fclose(fid);
> szamok, betuk
> % Írassuk ki a számok összegét a képernyőre!
> szumma=sum(szamok);
> fprintf('A szamok osszege: %.2f\n',szumma)
```

Az eredmény: A számok összege: 16.30

PÉLDA INTERFEROMÉTERES ADATOK BEOLVASÁSÁRA (FSCANF)

Nézzünk egy kicsit bonyolultabb adatbeolvasást. A következő fájl interferométerrel készült méréseket tartalmaz. Van egy fejléc része (ez mindig ugyanannyi sorban van tárolva, most épp 26-ban), utána jönnek a tényleges mérések (ennél nem tudjuk előre a sorok számát), majd a végén még egyéb adatok jönnek (pl. légnyomás stb.).

teszt_interfero.txt

```
HEADER
File type  : rtl
…
Run Target Data:
```

```
1   1       7.403
1   2    -994.335
2   2   -1008.836
…
ENVIRONMENT::
Air temp   : 22.445311 22.460936 0
…
EOF
```

A lényeges információ 3 oszlopban van, és tetszőleges számú sorban. Ezt lenne jó beolvasni egy mátrixba valamilyen módon. Itt nem használhatjuk az end-of-file opciót a ciklushoz, mert az adatok nem a fájl végéig tartanak, más módot kell keresnünk. Az első 26 sornyi fejlécet könnyen át tudjuk ugrani, ha **fgetl** paranccsal beolvasunk 26 sort. A többit beolvashatjuk formázott szövegként az **fscanf** paranccsal. Ez ugyanazokat a formátumokat használja, mint a korábban már használt sprintf, fprintf. Megadhatjuk neki, hogy 3 szám van egy sorban szóközzel (vagy tabbal \t elválasztva, jelen esetben mindegy melyiket használjuk), és megadhatjuk a méretet is. Az fscanf oszlopokat olvas be, amikor a méretet adjuk meg, először az oszlopok számát kell megadni, és utána a sorokét, de ezt állíthatjuk előre nem meghatározottra (itt végtelen - inf) is. A parancs leáll, ha olyan sort talál, ami nem felel meg a formátumnak.

```
> fid = fopen('teszt_interfero.txt');
> for i=1:26; fgetl(fid); end; % Kihagyjuk az első 26 sornyi fejlécet
> % számok beolvasása 3 oszlopba, az első szövegnél leáll
> interfero=fscanf(fid,'%f %f %f\n',[3 inf])
> fclose(fid);
> % transzponáljuk hogy 3 oszlopban legyenek az adatok, ne 3 sorban
> interfero = interfero'
```

BEOLVASÁS FSCANF, TEXTSCAN HASZNÁLATÁVAL

Az **fscanf** paranccsal egyszerre az egész fájlt be tudjuk olvasni egy megadott formátum szerint. Az eredmény egy mátrixba kerül. Ezzel a paranccsal különböző hosszúságú sorokat nem tudunk beolvasni, azoknál célszerű az **fgetl**, **fgets** parancsot használni. Nézzük meg a számok/betűk szétválasztását az **fscanf** parancsot használva!

A beolvasás oszloponként történik, megadjuk a formátumot, amiben az adatok le vannak tárolva most először egy szám, szóköz, aztán egy betű. Ekkor először az első oszlop kerül beolvasásra, amiben a számok vannak, ez kerül az első sorba, utána a második oszlop a betűkkel, ez kerül a második sorba. Transzponáljuk, hogy a forma megfeleljen az eredetinek.

```
> clear all; clc;
> fid=fopen('szambetu.dat');
> mat=fscanf(fid,'%f %s',[2 inf])'
> fclose(fid);
```

Eredmény:

```
5.3000   97.0000
2.2000   98.0000
3.3000   97.0000
4.4000   97.0000
```

```
1.1000    98.0000
```

Miért lett ilyen furcsa az eredmény? Hol vannak a betűk? Mivel egy mátrixban csak egyféle adatot lehet tárolni (csak szám vagy csak szöveg), a betűk helyett azok ascii kódja került eltárolásra. Válasszuk szét a változókat és alakítsuk vissza szöveggé!

```
> szamok = mat(:,1)
> betuk = char(mat(:,2))
```

Az `fscanf`-hez hasonlóan használható az `sscanf`, csak ez nem fájlból olvas be adott formátum szerint, hanem stringből!

Hogyan tudnánk az adatokat úgy beolvasni, hogy a különböző formátumok ne okozzanak gondot? Van egy másik változó típus, a cellatömb, amiben különböző típusú változókat is eltárolhatunk. A megadása hasonló a mátrixokhoz, csak itt kapcsos zárójelet {} használunk szögletes [] helyett. A **`textscan`** parancs hasonló az **`fscanf`**-hez, de ez cellatömbbe olvas be adatokat.

```
> clear all; clc;
> fid=fopen('szambetu.dat');
> adatok=textscan(fid, '%f %s');
> fclose(fid);
> % Eredmények szétválasztása a cella tömbből
> szamok=adatok{1} % mátrix számokkal
> betuk=adatok{2} % ez egy cellatömb karakterekkel
> betuk{1,1} % ez már maga az eltárolt betű
> betuk = char(betuk) % karaktertömböt (vektort) készít a cellákból
```

Nézzünk még egy példát az fscanf használatára. Olvassuk be az *xypoints.dat* állományt!

```
x2.3y4.56
x7.7y11.11
x12.5y5.5
```

Itt x,y koordináták vannak, csak előttük ott áll, hogy x vagy y. A számok különböző hosszúak, 3-5 karakteren tárolódnak, így a karakterszám szerinti szétválasztás nem megy. Formázott szövegként viszont be tudjuk olvasni. Ha az fscanf vagy textscan részeként konkrét szöveget adunk meg (itt pl. x vagy y), akkor azok nem kerülnek beolvasásra, hanem csak a köztük lévő számok, amit pl. %f alakban adhatunk meg. Most nincs problémánk a különböző típusokkal, hiszen a lényeges információ csak számokból áll.

```
> fid = fopen('xypoints.dat')
> xy = fscanf(fid, 'x%fy%f\r\n', [2 inf])'
> fclose(fid)
```

Érdemes tanulmányozni ezeknek a parancsoknak a help-jét, mivel számtalan opciót lehet még megadni. Pl. **`fscanf`**-nél, ha %*s vagy %*f alakot adok meg, akkor átugrik egy bizonyos karaktert/számot, **`textscan`** parancsnál meg lehet adni fejlécet, kommentstílust, amiket kihagy a beolvasásból, meg lehet adni elválasztót a szövegek között (szóköz, pont, vessző...) stb.

Leggyakrabban azonban az fgetl parancsot használjuk a beolvasásra, amikor nem adott formában vannak az adatok, esetleg változó sorhosszúságúak és egyenként kell minden sort megvizsgálni.

KÜLÖNBÖZŐ HOSSZÚSÁGÚ SOROK BEOLVASÁSA

Nézzünk most egy olyan példát, ahol különböző hosszúságú soraink vannak, pl. a rögzített mérések száma eltér egymástól. Ilyen lehet pl. egy légi lézerszkenner visszaverődéseinek rögzítése.

Legyen például a következő fájlunk:

meresek.dat

3 4 5 23 56 67 43 21 11

1 4 7 15 26 11

4 17 35 78 43 32 21

Hogyan tudjuk ezt beolvasni? Azt szeretnénk, hogy az eredmény egy tömbbe kerüljön, és a tömb üres elemei (az eltérő sorhosszak miatt) töltődjenek fel 0-val.

Érdemes soronként beolvasni az **fgetl**-lel, és utána egy ciklussal beleírni az elemeket a mátrix következő sorába. A hiányzó elemek automatikusan 0-k lesznek. Nézzük a megvalósítást. A fájl megadásakor válasszuk a meresek.dat fájlt!

```
> clear all; close all; clc;
> page_screen_output(0); % laponkénti megjelenítés leállítása Octave-ban
>
> % beolvasni kívánt fájl kiválasztása
> [inputfilename, inputpathname] = uigetfile('*.dat', 'Valaszd ki a
  beolvasando fajlt!');
> fajlnev = [inputpathname inputfilename]
> fid = fopen(fajlnev, 'r'); % fájl megnyitása olvasásra
```

Először olvassunk be egy sort az **fgetl** paranccsal, utána vizsgáljuk meg, hogyan tudnánk a sorban lévő számokat egy mátrixba beírni! Erre több lehetőségünk is van.

Először nézzük meg a szövegfeldolgozó parancsokat! Ezek között nagyon sok olyan van, ami hasznos lehet a számunkra, pl. a **strsplit** utasítás, ami szétdarabolja a szóközök (vagy opcionálisan megadott más határoló karakter mentén, például vessző - **strsplit(str,',')**) a szöveget, és a darabokat egy cellatömbbe teszi. Ilyenkor a cellatömb elemei szövegek, de ezt át lehet könnyen alakítani számmá a **str2double** paranccsal. Most a kimenet egy sorvektor lesz ami a számokat tartalmazza.

```
> a = strsplit(line)
> a = str2double(a)
```

Egy másik megoldás, ha használhatjuk a **sscanf** utasítást, ami ugyanaz, mint az **fscanf** (amit már használtunk korábban például az interferométeres adatok beolvasásakor), csak nem fájlból, hanem string típusú változóból olvas be formázott szöveget.

```
> line = fgetl(fid)
> a = sscanf(line,'%f')
```

A fenti parancsban a %f, azt jelenti, hogy lebegőpontos számokat (floating point number) olvas be a stringből, amíg talál ilyeneket. A kimenet egy oszlopvektor lesz ami a számokat tartalmazza. Ez tökéletesen megfelel a céljainkra. Meg lehet hívni két

kimenettel is, ilyenkor a második kimenetbe az kerül, hogy hány darabot talált az adott formátumból. Hívjuk meg mi is így később még szükség lehet a darabszámra.

```
> [a n]=sscanf(line,'%f')
```

Bármelyik megoldást is használjuk a végeredmény egy (oszlop vagy sor) vektor lesz, ami tartalmazza az adott sorban lévő számokat. Miután több sort beolvastunk ezeket kéne összefűzni egy mátrixba. A gond csak az, hogy ezek a sorok nem egyenlő hosszúak, nem lehet egy [a1; a2] paranccsal összefűzni őket.

Hogyan tudunk eltérő hosszúságú sorokat mátrixba összefűzni? Hozzunk létre egy tetszőleges, de a-tól eltérő darabszámú b sorvektort, és egy M üres mátrixot. Az M mátrixba fűzzük össze az a és b vektort a következő módon (az üres helyekre 0-k fognak kerülni):

```
> b = [1 2 3], M=[]
> M(1,1:length(a))=a
> M(2,1:length(b))=b
```

Itt a length parancsot használtuk, hogy megszámoljuk hány elem van a mátrixban, de ha korábban az sscanf parancsot használtuk, akkor az rögtön megadja a darabszámot is! Töröljünk (vagy kommenteljünk) ki mindent az **fopen** parancs után, és most próbáljuk meg egy ciklusban beolvasni M mátrixba a sorokat, 0-val feltöltve az üres helyeket.

```
> M=[];i=0; % M tömb letrehozasa, i - sorok száma
> while ~feof(fid)
>     line=fgetl(fid); % egy sor beolvasása
>     i=i+1; % sorok szamat noveljuk eggyel
>     [a n]=sscanf(line,'%f'); % a – számok, n - darabszám
>     M(i,1:n)=a;
> end
> fclose(fid);
> M
```

Eredmények:

```
M =

     3     4     5    23    56    67    43    21    11
     1     4     7    15    26    11     0     0     0
     4    17    35    78    43    32    21     0     0
```

FONTOSABB INPUT/OUTPUT PARANCSOK ÖSSZEFOGLALÁSA ANGOLUL

```
load      - Load workspace variables from disk, load filename
save      - Save workspace variables to disk, save filename variables
            -ascii -append
fopen     - Open file, or obtain information about open files, fileID =
            fopen(filename)
fclose    - Close one or all open files, fclose(fileID)
fseek     - Move to specified position in file, fseek(fileID, offset,
            origin)
feof      - Test for end-of-file, feof(fileID)
```

```
fgetl     - Read line from file, removing newline characters,
            fgetl(fileID)
fgets     - Read line from file, keeping newline characters,
            fgets(fileID)
fscanf    - Read formatted data from a text file, converts data into
            array, fscanf(fileID, format)
sscanf    - Read formatted data from string, sscanf(str, format)
textscan  - Read formatted data from text file or string, returns a cell
            array, textscan(fid, 'format')
fprintf   - Write data to text file, fprintf(fileID, format, A, ...)
sprintf   - Format data into string, sprintf(format, A, ...)
dlmread   - Read ASCII-delimited file of numeric data into matrix,
            M = dlmread(filename, delimiter)
dlmwrite  - Write matrix to ASCII-delimited file, dlmwrite(filename, M,
            'D')
fileread  - Read contents of file into string, text = fileread(filename)
fread     - Read data from binary file
```