Matlab bevezető

1. A segédlet használata

Jelen segédlet a Jelfeldolgozás, méréskiértékelés c. tárgyhoz készült abból a célból, hogy az azt feldolgozó hallgatók megismerkedjenek a Matlab programnyelv alapjaival és megszerezzék azokat a képességeket, melyekkel teljesíthetők a tárgy laborgyakorlatainak számítógépes részei.

A segédletben a programnyelvet a Matlab fejlesztői környezeten (*IDE: Integrated Development Environment*) keresztül mutatjuk be. Ez egy olyan szoftverkörnyezet, amiben meg tudjuk írni a programkódot, le tudjuk fordítani és rögtön futtathatjuk is, azaz egy új programot az első lépésektől az utolsóig el tudunk benne készíteni.

A segédletben *dőlt* betűvel szedett részek – az előtanulmányoktól függően – további magyarázatra szorulhatnak, de nem feltétlenül szükségesek a későbbiek megértéséhez. Javasoljuk, hogy ezekhez a hallgató először önállóan próbáljon magyarázatot találni, és csak további kérdés esetén forduljon a gyakorlatvezető oktatóhoz!

Javasoljuk továbbá, hogy a Matlab IDE és a jelen dokumentum egyidőben legyen nyitva a számítógépen, kiegészítve valamely internetes keresőprogrammal (pl. Google).

- Ezekben a szövegdobozokban gyakorló feladatok lesznek, melyek esetenként plusz pontokat érnek.
- (0 pont) A megszerezhető pluszpontokat minden ilyen feladatnál jelezzük.

```
% Ezekben a szövegdobozokban programkód lesz.
% A sorokat bemásolhatjuk a MatLab parancssorába és lefordíthatjuk.
```

2. Alapok és ismétlés

A Matlab helye a programnyelvek között

A Matlab egy ún. *magas szintű*, *röpfordítású* programnyelv, mely alapvetően a mérnöki munkát hivatott segíteni. Mind a fejlesztés, mind a kutatás területén elterjedt, mivel a nagyszámú beépített *szubrutinnak* köszönhetően gyorsan kipróbálhatók vele az új ötletek. Továbbá támogatja a két legelterjedtebb programozási paradigmát (*procedurális* és *objektum-orientált*), így más nyelvek ismerői számára gyorsan tanulható. Különösen nagy a hasonlóság a Python programnyelvvel, egyes parancsoknál a funkcionalitás mellett sokszor a parancs neve is egyezik.

A kurzus során a Matlab alábbi erősségeit fogjuk megtapasztalni:

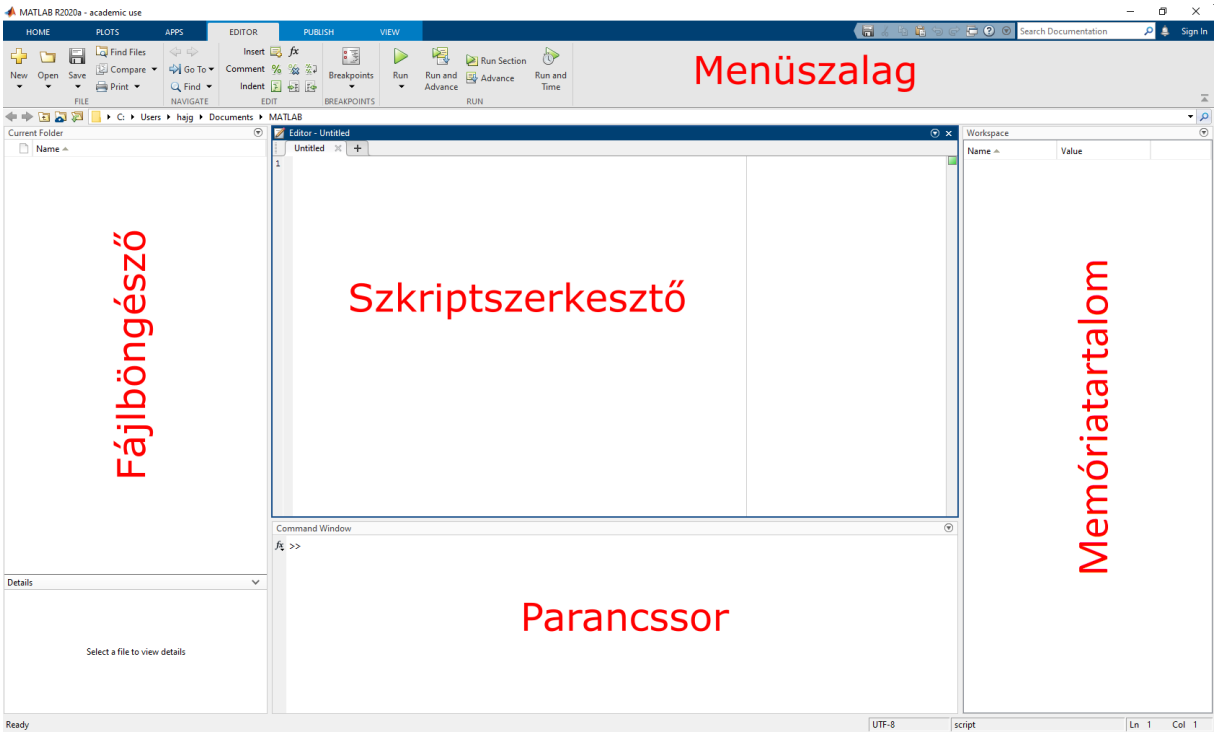
- a fordító képes különböző I/O eszközökkel kommunikálni (adatgyűjtés mérőerősítővel);

- a nyelvben sok algoritmust előre megírtak, melyek egyszerűen meghívhatók (mérési adatok feldolgozása);
- a diagramok rajzolásához szükséges eszközök könnyen elérhetők, a felhasználónak elég a matematikára koncentrálnia (mérési adatok kiértékelése).

A Matlab fejlesztői környezet

Az integrált fejlesztői környezetek kifejezetten a programkód megírásához nyújtanak segítséget. (A programozási feladat megoldásmenetének kigondolásához és felvázolásához továbbra is a papír és a ceruza ajánlott.)

Nyissuk meg a Matlab IDE-t, és ismerkedjünk meg a legalapvetőbb részeivel! Az alapértelmezett nézetet látjuk az 1. ábrán.



1. ábra: Matlab fejlesztői környezet

MENÜSZALAG
Itt érjük el a programkóddal kapcsolatos parancsokat, mint pl. töréspont beszúrása, futtatás, mentés, stb.

FÁJLBÖNGÉSZŐ
A menüszalag alatti címsorban jelzett mappa tartalmát mutatja. A Matlab saját fájltípusaira kattintva a bal alsó sarokban betekintést nyerhetünk a fájlba, pl. szkripteknél a leírásba, mentett memóriatartalomnál a változókba láthatunk bele.

SZKRIPTSZERKESZTŐ
Itt vihetjük be és szerkeszthetjük a programkódot. Néhány előnyös tulajdonsága, hogy

- a bevitelt automatikusan színezi és intendálja;
- a tabulátor lenyomására kiegészíti a félig begépelt parancsokat vagy változókat;

- ha a kurzort egy parancsba állítjuk és lenyomjuk az 1-es funkcióbillentyűt (F1), a Matlab vonatkozó súgója ugrik fel.

PARANCSSOR
Az ide bevitt parancsok azonnal lefordulnak és futnak, hasonlóan pl. a Linux, DOS, Windows parancssorhoz.

MEMÓRIATARTALOM
Itt látjuk a memória tartalmát, azaz a hivatkozható változókat és azok értékeit.

Változók definiálása

A Matlab nyelvben azonnali értékadással is definiálhatunk új változókat, ekkor a típust sem szükséges megadnunk, azt a fordító maga találja ki.

A fontosabb változótípusok a következők (zárójelben a matlabos elnevezéssel):

- előjel nélküli egész szám 32 biten ábrázolva (uint32);
- előjeles egész szám 32 biten ábrázolva (int32);
- előjeles, egyszeres pontosságú, lebegőpontos szám (single);
- előjeles, dupla pontosságú, lebegőpontos szám (double);
- karakter (char).

A Matlabban – mint azt később látni fogjuk – a változók a típusuktól függetlenül egy tömböt jelölnek, ezért nincs külön vektor vagy tömb változótípus.

A következőket írjuk be a parancssorba! A sorvégi pontosvesszőkkel a parancsokat „elnémíthatjuk", ilyenkor nem látjuk a parancsok visszatérési értékét a terminálban. Egy értékadó parancs Matlabban magával az értékkel tér vissza, ezt az egyik pontosvessző elhagyásával ki is próbálhatjuk. Minden parancsot egy Enter leütésével kell nyugtáznunk, ezután azt a fordító értelmezi és futtatja.

```
szam_valtozo = 1;
szoveg_valtozo = 'Milyen típus lehetek?'
sorvektor = [1, 2, 3, 4];
```

Figyeljük meg, hogy a memóriatartalomban megjelentek a fenti változók! Ha duplán rákattintunk valamelyikre, akkor megnézhetjük, hogy milyen adatok vannak benne.

Hozzunk létre pár további változót és végezzünk el velük pár alapvető műveletet!

```
a = 1;
b = 3;
c = pi;
d = b;
```

A `c` változónak a `pi` értéket adtuk, amit nem tettünk idézőjelek közé, ezért a Matlab megpróbálja értelmezni azt. A `pi` egy beépített változó, ami a π duplapontosságú értékét jelöli. Tehát `c` ugyanazt az értéket kapja, mint a beépített `pi` változó, `d` pedig azt, mint a korábban definiált `b` változó. Ezt ellenőrizhetjük, ha csak az adott változó nevét adjuk ki parancsként.

A fenti változókon túl használhatunk ún. struktúrákat is, amik a fenti változótípusokat fogják össze. Pl. ha egy szállítóláda méreteit és tartalmát szeretnénk egy változó alatt nyilvántartani, akkor így megtehetjük.

```
lada.meret_1 = 5;
lada.meret_2 = 2.5;
lada.meret_3 = 1;
lada.meret_mertekegyseg = '10 cm';
lada.tartalom = {'alma', 'körte', 'barack'};
```

- Írassuk ki a `lada` struktúra tartalmát!
- Írassuk ki csak az egyik változójának értékét!

A struktúrában lévő ún. mezők ugyanolyan változók lehetnek, mint amiket előbb láttunk. Ezekre hivatkozni a fent látott módon lehet.

Műveletek változókkal

A változókkal műveleteket is végezhetünk, pl. összeadhatjuk és szorozhatjuk őket.

```
a+b
d = (a+b)*b;
```

Az előbbi esetben az eredményt csak a parancssorba írattuk ki, utóbbi esetben viszont a művelet végeredménye a *d* változóba került.

- Írassuk ki a `lada` első méretének kétszeresét!

Függvények

Egy szám szinuszát a következőképpen határozhatjuk meg.

```
sin(0.5)
```

Programozási szempontból a szinusz egy függvény, aminek az argumentumlistája egyelemű. Függvény nem csak a matematikai előtanulmányainkból ismert függvény lehet, hanem elvontabb utasítássorozatok is. A Matlab a függvény neve utáni gömbölyű zárójelek közötti részt értelmezi argumentumokként. Ha konkrét szám helyett kifejezést írunk be (pl. matematikai műveletet, változónevet, egy másik függvényt a saját argumentumaival), akkor a Matlab azt először lefordítja, és úgy adja át a függvénynek.

Nézzünk meg egy ilyen összetett példát! Fordítsuk meg a számok sorrendjét a korábban létrehozott `sorvektor` változóban és határozzuk meg a legkisebb elem értékét és sorszámát!

```
[min_ertek, min_idx] = min(flip(sorvektor));
```

A `flip` függvény az argumentumban szereplő vektor fordítottjával tér vissza, ezt pedig rögtön átadjuk a `min` függvénynek. Az egyenlőségjel bal oldalán egy kételemű vektor szerepel, azaz a `min` függvénytől azt várjuk, hogy két értékkel térjen vissza. A beszédes változók segítenek, de ha nem vagyunk biztosak abban, hogy mi történik,

akkor a kurzorral álljunk rá a `min` függvényre és nyomjuk meg az F1-et, majd keressük meg a releváns részt a leírásban!

- A fenti műveletsort végezzük el úgy, hogy közben a tükrözött sorvektort elmentjük egy változóba! (Nem kötelező egysoros paranccsal megoldani.)
- Írassuk ki a korábban létrehozott változók típusát a megfelelő függvénnyel! Ennek megtalálásához használjuk a „type of variable matlab" keresőkifejezést!

Műveletek vektorokkal és mátrixokkal

Hozzunk létre vektort! Figyeljük meg, hogy mitől lesz az egyik sor- a másik oszlopvektor!

```
sorvektor = [1, 2, 3];
oszlopvektor = [1; 2; 3];
```

Írassuk ki a vektorokat és vegyük az egyik transzponáltját!

```
sorvektor
oszlopvektor
transpose(sorvektor)
```

A kijelölések és az alapvető műveleteket áttekintéséhez egy mátrixot (a Matlab szemszögéből egy kétdimenziós tömböt) is segítségül hívunk.

```
elso_elem = 1;
utso_elem = 9;
mx = transpose(reshape(linspace(elso_elem, utso_elem, 9), [3, 3]))
```

Az ismeretlen parancsokra keressünk rá a súgóban! Gyakran fordul elő, hogy egy tömbnek csak egy adott részét szeretnénk feldolgozni, ekkor ki kell jelölnünk a megfelelő elemeit. Ilyenkor minden dimenzióban meg kell adnunk, hogy az adott dimenzió mentén mely elem(ek)et szeretnénk kijelölni. Matlabban a mátrixoknál az első koordináta a sorokra, a második az oszlopokra vonatkozik.

Nézzük meg, hogy mit adnak vissza a következő parancsok!

```
mx(2, 2) % a 2. sor 2. eleme
mx(2:3, 2:3) % a 2-3. sorok 2-3. elemei
mx(2:end, 3) % a 2. sortól a dimenzió végéig az összes sor 3. eleme
mx(1, :) % az első sor összes eleme
```

- Jelöljük ki rendre az első, az utolsó, illetve az első és az utolsó elemeket a korábbi `sorvektor` és `oszlopvektor` változóból!
- A korábban létrehozott struktúránkban a `tartalom` változó is egy tömb, aminek az elemei `char` típusúak. Ezeket a már ismert módon tudjuk indexelni. Írassuk ki a parancssorba a láda tartalmának második elemét!
- (0,5 pont) Rendezzük át az `mx` változót sorvektorrá, majd jelöljük ki minden elemét a 3. elemtől kezdve!

Vegyük át az alapvető mátrixműveleteket!

```
mx + mx % összeadás
mx + sorvektor % összeadás*
mx * 2 % szorzás skalárral
mx * transpose(sorvektor) % mátrixszorzás
mx .* transpose(sorvektor) % mátrixszorzás*
```

A csillaggal jelölt műveletek nem fedik a lineáris algebrából tanultakat, mégis sok programnyelv elfogadja ezt a jelölést. A háttérben ún. *broadcasting* történik, azaz a fordító megpróbálja úgy elvégezni a műveletet, hogy az a műveletben résztvevő változók dimenzióival összeegyeztethető legyen.

Az **összeadás**nál így a fordító a sorvektort a mátrix minden sorához elemenként adja hozzá. A **mátrixszorzás**nál a fordító a mátrix oszlopvektorait elemenként szorozza össze a sorvektor transzponáltjával.

- Gondoljuk végig, hogy mely műveletek működtek volna a fenti sorvektorral és egy 2x3-as mátrixszal! Ha bizonytalanok vagyunk, próbáljuk is ki!

Próbálkozzunk meg egy érvénytelen művelettel is!

```
mx*sorvektor
```

```
Error using  *
Incorrect dimensions for matrix multiplication. Check that the number of columns in the first matrix matches the
number of rows in the second matrix. To perform elementwise multiplication, use '.*'.
```

Válaszul egy hibaüzenetet kapunk, a Matlab pedig nem végzi el a műveletet. Figyeljük meg, hogy a hibaüzenet segít a hiba megoldásában, ezért a későbbiekben is érdemes mindig elolvasni azt!

Végül tekintsük meg a mátrixszorzás programkódját egy alacsony szintű programnyelvben! Ha ezt összehasonlítjuk a fenti egysoros paranccsal, akkor ízelítőt kaphatunk abból, hogy miért terjedtek el a magas szintű programnyelvek a kutatásban és a fejlesztésben.

3. Diagramok rajzolása

Ebben a fejezetben a diagramok rajzolásáról lesz szó. Kezdésnek hozzunk létre két, pszeudo-véletlenszámokkal feltöltött vektort!

```
xx = rand([5000, 1]); % pszeudo-véletlenszámok egyenletes eloszlásból
yy = randn([5000, 1]); % pszeudo-véletlenszámok normális eloszlásból
```

Diagram rajzolás

Ábrázoljuk az imént generált számokat! Ezt megtehetjük a `plot` paranccsal, ami az argumentumlistától függően kétféleképpen ábrázol:

- 1 db egydimenziós vektornál az x-tengelyen az elem sorszáma, az y-tengelyen az értéke látszik;

- 2 db egydimenziós vektor esetén az x-tengelyen az első vektor értékei, míg az y-tengelyen a második vektor értékei látszanak.

```
plot(xx, 'k.') % független változó: sorszám
plot(xx, yy, 'r.') % független változó: xx, függő változó: yy
```

A parancsnak több argumentumot is megadhatunk, mi most a jelölő alakját és színét is megadtuk. A megjelenítés után zárjuk be a diagramok ablakait.

Ábrázoljuk egy diagramban a szinusz és a koszinusz függvényt! Ehhez először létrehozunk egy új rajzterületet, majd „megfogjuk" azt. Amíg tartjuk a vásznat, addig minden rajzparancs a fogott vásznon érvényes. (A korábbi görbék nem törlődnek.)

```
x = linspace(0, 2*pi, 100);
x_sin = sin(x);
x_cos = cos(x);

figure(); % új, üres rajzvászon
hold on; % vászon megfogása
plot(x, x_sin);
plot(x, x_cos);

xlabel('x') % tengelyfelirat a vízszintes tengelyen
ylabel('f (x)') % tengelyfelirat a függőleges tengelyen
legend('sin(x)', 'cos(x)') % jelmagyarázat
```

Az ábrákat el is menthetjük, amihez először be kell állítanunk a méreteit.

```
set(gcf, 'PaperPositionMode', 'manual'); % kézi papírméret beállítás
set(gcf, 'PaperUnits', 'centimeter'); % mértékegység megadása
set(gcf, 'PaperPosition', [0 0 7.5 5]); % ábra papíron belüli elhelyezkedése
set(gcf, 'PaperSize', [7.5 5]); % papírméret

% ábra mentése adott fájlnévvel és felbontással png formátumba
print('teszt_abra', '-dpng', '-r600')
```

Figyeljük meg, hogy a fájlböngészőben megjelent a `teszt_abra.png`!

- Ábrázoljuk az x^2 függvényt a [-1, +1] tartományon!

4. Algoritmusok leírása

Áttekintjük a korábbi tanulmányainkból ismert algoritmikus egységeket, úgymint elágazás, tesztelő ciklus, számláló ciklus. Innentől célszerű a szkriptszerkesztőt használni a kód beírásához, majd minden kiegészítésnél lefuttatni a szkriptet a végeredmény ellenőrzéséhez. Egyúttal törölhetjük a memória tartalmát a `clear all` paranccsal.

Elsőként inicializáljuk az `xx` és `yy` változókat pszeudo-véletlenszámokkal, mint fent! Az ötös funkcióbillentyű lenyomásával egyidejűleg mentjük és futtatjuk a szkript tartalmát.

Elágazás

Nézzük meg, hogy `yy` utolsó eleme pozitív vagy negatív!

```
if yy(end) > 0
    fprintf('yy utolsó eleme pozitív.')
else
    fprintf('yy utolsó eleme negatív.')
end
```

Itt kijelöltük az `yy` vektor utolsó elemét és egy elágazásba tettük. Most álljunk meg egy kicsit! Valóban jól működik ez a kódrészlet?

- (1 pont) A súgó segítségével egészítsük ki a fenti kódot úgy, hogy 0 esetén is jó eredményt írjon ki!

Tesztelő ciklus

A MatLabban a tesztelő ciklus addig fut, amíg a ciklusfeltétel igaz. Keressük meg `yy` első negatív elemét és írjuk ki a sorszámával együtt!

```
idx = 1;
while yy(idx) >= 0
    idx = idx+1;
end
```

A ciklus futása után a sorszám az `idx` változóban van. Írjuk ki a kimenetre az elemmel együtt!

```
fprintf('yy első negatív eleme a/az %d. helyen van, értéke %f', idx, yy(idx))
```

A fenti kifejezésben a %-kal kezdődő kifejezések helyére az `fprintf` parancs argumentumlistájában szereplő értékek kerülnek be. A % utáni betű jelzi, hogy milyen formátumban szeretnénk kiíratni a számokat (d – egész értékes, f – lebegőpontos).

- (1 pont) Írjuk át a fenti kódrészletet úgy, hogy a legutolsó negatív elemet találja meg, majd írassuk ki az értékét és a sorszámát! Segítségül: egy tömb legnagyobb méretét a `length` paranccsal lehet lekérdezni.

Számláló ciklus

A számláló ciklus egy speciális tesztelő ciklus, ahol egy ún. ciklusváltozóval nyilvántartjuk, hogy hányadszor ismétlődik a ciklusmag. A leállító feltétel ebben az esetben az ismétlések száma.

Pl. az `yy` vektor első 10 elemét kiírathatjuk a következőképpen.

```
for idx = 1:10
    fprintf('yy %d. eleme %f\n', idx, yy(idx))
end
```

A `\n` itt egy *vezérlőkaraktert* jelöl, nevezetesen a *soremelés* vezérlőt.

Kitérő: logikai operátorok

Logikai operátorokkal a fentieknél összetettebb feltételeket is előírhatunk. A legalapvetőbb operátorok:

- és (&),
- vagy (|),
- negáció (~),
- kizáró vagy (xor).

Ezek használatára a későbbiekben látunk példát.

5. Saját függvények létrehozása

A gyakorlatban egy-egy algoritmust akár több programban vagy egy programon belül többször is használunk. Ezekben az esetekben időpazarló és potenciális hibaforrás lenne újra és újra megírni azt a kódot, amit korábban már megcsinálunk és teszteltünk. Emiatt a gyakorlatban gyakran alkalmazunk saját függvényeket, melyeknek a hívása egyezik a korábban már megismert beépített MatLab függvényekkel (pl. `linspace`).

Útvonalak

A MatLab egy-egy parancs értelmezésekor keresést indít a parancs nevével egyező `.m` fájlok után az ún. keresési útvonalakon. Ezek az útvonalak a fájlrendszer több mappáját is magukba foglalják, de pl. a fájlböngésző mindenkori útvonala is a keresési útvonalhoz tartozik.

Függvény definíció és függvényhívás

Az alapismereteket tekintsük át egy példán: hozzunk létre egy saját függvényt a Fibonacci-sorozat kiszámítására, amit később más programokban is meghívhatunk! A függvény bemenete (argumentuma) egy egész szám lesz, eddig a Fibonacci-számig számoljuk ki a sorozatot. A függvény kimenete pedig a sorozat lesz, sorvektorként.

Nyissuk meg a kódszerkesztőt (a menüszalagon a New gomb alatt kattintsunk a Script gombra), majd rögtön mentsünk is el a még üres szövegfájlt az alapértelmezett útvonalon `fibonacci_sorozat.m` néven! A fájl ezek után megjelenik a fájlböngészőben (ld. 1. ábra), ami azt jelenti, hogy a parancssorban már hivatkozhatunk a `fibonacci_sorozat()` függvényre. A szerkesztőbe másoljuk be az alábbiakat, majd mentsünk!

```
function sorozat = fibonacci_sorozat(n)
    if (n < 0) || (floor(n) ~= n) % végtelenre nem működik!
        fprintf('Érvénytelen paraméter.')
        sorozat = [NaN];
    elseif n == 0
        sorozat = [0];
    else
        sorozat = [0, 1];
        for i = 2:n
            sorozat(end+1) = sorozat(end) + sorozat(end-1);
        end
    end
end
```

A függvény első sorában definiáljuk a függvény

- visszatérési értékét (`sorozat` változó, amit majd a függvényben hozunk létre);
- nevét (célszerű a fájlnevet használni, de nem kötelező);
- argumentumlistáját (`n` változó, amit majd a függvényhívásnál használunk).

Figyeljük meg, hogy a függvény az elágazás mindegyik ágában kap visszatérési értéket! Próbáljuk is ki a parancssorban!

```
>> fibonacci_sorozat(6)

ans =

     0     1     1     2     3     5     8
```

Függvények több argumentummal és visszatérési értékkel

A fenti mintára olyan függvényeket is írhatunk, melyek több bemeneti és kimeneti változóval dolgoznak. Lássunk erre egy példát!

```
function [osszeg, szorzat] = kismatek(a, b, c)
    osszeg = a+b+c;
    szorzat = a*b*c;
end
```

A visszatérési értékeket ilyenkor egy sorvektorba rendezzük és a függvényhívásnál is így kérjük az eredményt:

```
>> x = 1;

>> [ossz, szorz] = kismatek(1,x,3);
```

Figyeljük meg, hogy a változóneveknek nem kell egyezniük a függvény definíciójában és a függvényhívásban! A definícióban használt változók csak a függvény névterében léteznek, azon kívül nem tudunk hivatkozni rájuk: a fenti hívás után a memóriatartalomban csak az `x`, `ossz` és a `szorz` változók szerepelnek.

Kitérő: rekurzió (érdekességként, a félévben nem lesz rá szükség)

Függvények segítségével rekurzív (önmagukra hivatkozó) algoritmusokat is megvalósíthatunk. Lássunk erre egy példát az n-edik Fibonacci-szám (a sorozat 0-tól

indul) kiszámításán keresztül! A szkriptet célszerű egy új, `fibonacci_szam.m` fájlba menteni.

```
function fibo = fibonacci_szam(n)
    if (n < 0) || (floor(n) ~= n) % végtelenre nem működik!
        fprintf('Érvénytelen paraméter.')
        fibo = NaN;
    else
        fibo = rekurziv_mag(n);
    end
end

function ertek = rekurziv_mag(n)
    if (n == 0) || (n == 1)
        ertek = n;
    else
        ertek = rekurziv_mag(n-1) + rekurziv_mag(n-2);
    end
end
```

Ebben az esetben két függvénydefiníció is szerepel a fájlban. Ilyen helyzetekben a MatLab a legelső függvényt futtatja le. A paraméter ellenőrzését kivettük a rekurzióból, így az csak egyszer, az önismétlő hívás előtt történik meg.

6. Záró feladat az alapismeretekhez

Végezetül nézzük meg az egyik legegyszerűbb rendezési algoritmus, a buborékrendezés pszeudokódját! Sikerülne megírni MatLabban?

A tömb elemeinek cseréjéhez használhatunk egy segédváltozót, mint pl. itt az 5. és 6. elemek cseréjéhez.

```
tomb = linspace(0, 1, 11)
seged = tomb(5);
tomb(5) = tomb(6);
tomb(6) = seged;
tomb
```

- (2 pont) Rendezzük növekvő sorba az xx vektor elemeit az általunk megírt buborékrendezési algoritmussal, majd jelenítsük meg a rendezett vektor tartalmát sorszám szerint!

7. Adatgyűjtés National Instruments mérőerősítővel

E fejezet során összeillesztjük a mérőerősítőt a számítógéppel és áttekintjük, hogy hogyan tudunk vele Matlab környezetben adatokat gyűjteni.

Összeállítás

1. Az USB-kábelt dugjuk be a mérőerősítőbe.
2. Az USB-kábelt dugjuk be a számítógépbe.
3. Ellenőrizzük, hogy a mérőerősítőn a kábelkivezetés melletti LED kéken világít-e. Ez jelenti azt, hogy a mérőerősítő mintavételezésre kész.

Ha a LED nem világít, akkor elsősorban azt ellenőrizzük, hogy a megfelelő szoftvereket telepítettük-e a számítógépünkre, ld. Laborhoz tudnivalók c. segédlet.

Mintavételezés előkészítése

Mielőtt mérhetnénk, a mérőerősítőt fel kell ismertetnünk a Matlabbal, valamint be kell állítanunk a mintavételezés paramétereit. A főbb lépések a következők.

1. Elérhető mérőerősítők listázása.
2. Új mérési munkafolyamat indítása egy adott erősítővel.
3. Csatornák hozzárendelése a munkafolyamathoz, amiken mérni szeretnénk.
4. Csatornák és munkafolyamat részleteinek beállítása (pl. mintavételezési frekvencia).

```
% Lekérjük a csatlakoztatott eszközök tulajdonságai egy struktúrába
devices = daqlist


% Egy új, a gyártmánynak megfelelő munkafolyamatot definiálunk
sess_analog = daq('ni');

% A munkafolyamathoz hozzáadjuk a mérőerősítő ai0 csatornáját, amin feszültség
jelet várunk
ch_1 = addinput(sess_analog, devices.DeviceID(1), "ai0", "voltage")

% a munkafolyamathoz hozzáadjuk a mérőerősítő ai1 csatornáját is
ch_2 = addinput(sess_analog, devices.DeviceID(1), "ai1", "voltage")

% Ha a ch_2 csatornán a földhöz képest mérjük a potenciált, akkor ezt be kell
állítanunk, mivel alapértelmezetten 'Differential' értékre van beállítva.
ch_2.TerminalConfig = 'SingleEnded'

% Beállítjuk a mintavételezési frekvenciát Hz-ben
sess_analog.Rate = 100;

% Létrehozunk egy változót, amiben megadjuk a mérés időtartamát másodpercben
DurationInSeconds = 5;
```

A mérési munkafolyamathoz hozzá kell adnunk egy mérőeszközt, amit a fent látható módon tudunk azonosítani. A továbbiakban ennek a mérőeszköznek a csatornáit tudjuk hozzáadni a méréshez, azaz ezekről a csatornákról történik majd adatgyűjtés függetlenül attól, hogy be van-e kötve rájuk valami. A csatornaazonosítókat (pl. `ai0`) látjuk az adatgyűjtő eszköz oldalán is.

Mérés

Miután előkészítettük a mintavételezést, a következőképpen indíthatjuk el a mérést.

```
% A korábbi beállításoknak megfelelően mintavételezünk.
% Az öt másodperc alatt mért adatok a sensor_data változóba kerülnek, míg a
timestamps változóba a mérésekhez tartozó időbélyegek.
[sensor_data,timestamps] = read(sess_analog,seconds(DurationInSeconds),"OutputFor-
mat","Matrix");


% első csatornán mért értékek
sensor_data(:, 1);

% az időbélyegek a mérési eredményekhez
timestamps
```

- Számoljuk ki a mért értékek átlagát és szórását mindkét csatornára!

Egyéb beállítások

Bizonyos méréseknél digitális csatornákat (is) használunk. Ezek többségére a laborokon nem lesz szükségünk (pl. LED-villogtatás, kommunikáció), viszont digitális számlálóra igen. A laborban rendelkezésre álló mérőerősítőkön egyszerre csak az analóg vagy csak a digitális csatornákat lehet használni, ezért a számláláshoz egy új munkafolyamatot kell definiálnunk.

```
% új munkafolyamat
sess_digital = daq("ni");

% számláló hozzáadása a munkafolyamathoz
% A számlálást a mérőeszköz 'ctr0' csatornáján végezzük.
addinput(sess_digital, devices.DeviceID(1),"ctr0","EdgeCount");


% a számláló pillanatnyi állapota
snapshot = read(sess_digital,1,"OutputFormat","Matrix");
```

Az `EdgeCount` paraméterrel arra utasítjuk a számlálót, hogy az ún. felfutó éleket (amikor a jel 0-ról 1-re vált) számolja.

Néhány kiegészítő parancs következik, amikre ritkán, de szükségünk lehet.

```
% n. csatorna törlése a sess_analog munkafolyamatból
removechannel(sess_analog,n);

% eszközök újraolvasása
daqreset;
```

Egy csatornát akkor érdemes törölni a munkafolyamatból, ha egy másik munkafolyamathoz szeretnénk hozzáadni, vagy egyszerűen csak nem vagyunk kíváncsiak az ott mért értékekre (mert pl. a műszert lekötöttük róla).

Az eszközöket akkor érdemes újraolvasni, ha az első eszközfelismerés óta új eszközöket csatlakoztattunk.

Végül, egy másik Matlab fájlban, MatlabAlapvetoParancsok.m, a legtöbb hasznos Matlab parancsot is megtaláljátok.

Gyakorló feladat

A két feladat együtt 0,5 pontot ér.

- Gyakorlásképpen hozzunk létre egy olyan munkamenetet, ami sess_practice parancsra az alábbiakat adja vissza, azaz az ai0-s és ai2-es csatornán mér feszültséget 10 másodpercig 50 Hz-es mintavételezési frekvenciával.
- Ábrázoljuk a mért értékeket az idő függvényében!

```
Data acquisition session using National Instruments(TM) hardware:
   Will run for 10 seconds (500 scans) at 50 scans/second.
   Number of channels: 2
      index Type Device Channel MeasurementType      Range        Name
      ----- ---- ------ ------- --------------- ---------------- ----
      1     ai   Dev1   ai0     Voltage (Diff)  -10 to +10 Volts
      2     ai   Dev1   ai2     Voltage (Diff)  -10 to +10 Volts
```