Matlab 2020.-ban használt parancsok, függvények

Használd a Matlab Helpet, ott mindent megtalálsz! ;)

Készítette:

Juhász Kinga, Kiss Réka

A dokumentumban leírt tartalmak helyességéért nem vállalunk felelősséget! Amennyiben hibát találsz, kérlek jelezd nekünk, hogy javíthassuk.

Pázmány Péter Katolikus Egyetem, Információs Technológiai és Bionikai Kar
1083 Budapest, Práter utca 50/a

Tartalomjegyzék

próba/innenmásolj

1. a parancs/funkció

   -
   -
   -
   -
   -
   -
   -
   - xjxfjz

     Példa:

1. Első óra (adminisztráció, kalkulátor, beépített függvények, saját szkript és függvény írása)

1. **Röviden a Matlabról:**

   - MATrix LABoratory (MathWorks)
   - Teljes szoftvercsomag, ami nagyban megkönnyíti és felgyorsítja az algoritmusfejlesztést
   - Hatékony numerikus megoldók mátrixműveletekre optimalizálva, saját szkript nyelv, könnyű adatmegjelenítés, rengeteg toolbox
   - Hogyan állítsuk a betűméretet nagyobbra? →felső szalag felületen a "Home" fülön →"Preferences" →a felugró ablak bal oldali listájában “MATLAB” / “Fonts” menüpont -> jobb oldalon feltűnő panelen “Desktop code font” méretét állítsuk nagyobbra (pl. 10).
   - A Command Windowban dolgozunk

2. **Változók:**

   - változók megadás egyből értékadásnál (*nem kell előre deklarálni a típust*, sőt, a típust nagyon ritkán adjuk meg direkten)

   Példa: a = 1;

3. **Beépített konstansok:**

   - **i** vagy **j**: komplex szám képzetes részének jelöléséhez
   - **pi**
   - **ans**: a legutóbbi olyan művelet eredményét tárolja, amelyet nem rendeltek semmilyen változóhoz

     Példa: 2 + 2 beírása esetén a 4 el lesz tárolva az *ans*-ban a következő ilyen "esetig"
   - **inf**: végtelen a pozitív, növekvő számokra nézve. (A negatív irányba menő végtelen: **-inf**
   - **nan**: "Not a Number", pl. 0/0 vagy ∞/∞ esetén ezt kapjuk vissza
   - **eps**: az a legkisebb lépésköz két nem egész szám között, amelyet a Matlab még érzékel. Ennek értéke 2^{-52}, tehát ha két olyan szám különbségére lennénk kíváncsiak, amelyek kevesebb, mint 2^{-52}-nel térnek el, akkor a Matlab már 0-t dobna vissza, mivel ennél kisebb számot már nem lenne képes felismerni.

4. **Operátorok:**

   - + (összeadás), - (kivonás), * (mátrixszorzás), / (mátrixosztás jobbról), ∧ (hatványozás), : (vektor létrehozásához, indexeléshez, iterációhoz), == (összehasonlítja a két objektumot, és visszatér azzal, hogy egyenlőek-e vagy sem)

     Példa: x = 1:10 egy olyan vektort hoz létre, melyben a számok 1-től 10-ig mennek.

5. **Beépített függvények:**
   - **sin(X), cos(X)**: radiánban megadott szög szinusza, koszinusza
   - **sind(X), cosd(X)**: fokban megadott szög szinusza, koszinusza
   - **tan(X), cotan(X)**: radiánban megadott szög tangense, kotangense
   - **tand(X)**: fokban megadott szög tangense (kotangensre nem találtam ilyet(?))
   - **sqrt(X)**: négyzetgyökét veszi az argumentumnak
   - **exp(X)**: e^x -t jelenti, komplex X-ekre is működik
   - **power(X,Y)**: hatványfüggvény, X az alap, Y a kitevő
   - **pow2(X)**: 2 X-edik hatványa
   - **log(X)**: természetes alapú logaritmus, ez is működik komplex számokra, és negatívokra is (negatív esetén komplexel tér vissza)
   - **log10(X)**: 10-es alapú logaritmus
   - **factorial(n)**: n faktoriálisa
   - **factor(X)**: a függvény az X egész szám prímtényezős felbontását adja meg sorvektorként
   - **primes(X)**: a függvény az n-nél nem nagyobb prímeket tartalmazó vektorral tér vissza
   - **round(X)**: A round függvényt ha egy paraméterrel hívjuk, akkor egész értékre kerekít, ha két paraméterrel, akkor az első paraméterként megadott számot a második paraméterként megadott számú tizedesjegyre kerekíti.
   - **floor(X), ceil(X)**: X alsó ill. fölső egészrészével tér vissza
   - **abs(X)**: X abszolútértéke
   - **datestr(X)**: stringgé konvertálja a dátumokat nap-hónap-év óra:perc:másodperc formátumban (X-et sorvektorban lehet megadni)
   - **min(X), max(X)**: ha X vektor, visszatér a legkisebb/legnagyobb elemével, ha X mátrix, visszatér egy sorvektorral, amely oszloponkénti legkisebb/legnagyobb elemét tartalmazza. Be lehet írni max(X, [], dim) formában is, így dim-mel megmondhatjuk melyik dimenzió mentén szeretnénk lekérni a legkisebb/legnagyobb elemeket. (Tehát ha dim=2-t írunk be, akkor egy oszlopvektort kaptunk a sorok minimumaival/maximumaival egy 2D-s mátrix esetében.)
6. **Szkriptek:**
   - az Editor ablakban készülnek;
   - .m kiterjesztés, megkötések: NE kezdődjön számmal, NE legyen benne space;
   - parancsok szekvenciális sora (ugyanúgy értelmezi a MATLAB, mintha a Command Window-ba gépelnénk a sorokat)
   - **clear variables**: eddigi változók törlése a memóriából
   - **disp('X')**: kiírás a konzolra

7. **Saját függvények**:

    - Szkriptekhez hasonlóan:
      az Editor ablakban készülnek, .m kiterjesztésűek,
      ugyanúgy nem kezdődhet számmal, és nem lehet space a nevében,
      ne egyezzen meg beépített függvény nevével (pl. plot.m);
    - Szkriptektől eltérően:
      saját munkatérbe dolgozik - a függvényben használt változók itt jönnek létre a függvény futása során, majd a függvény terminálásakor meg is szűnnek;
      fejléc function kulcsszóval kezdődik; az egészet end zárja le;
      opcionálisan lehetnek bemenő paraméterek és visszatérési értékek,
      Példa: function [visszateresi_p1, visszateresi_p2] = pelda_fv(bemeneti_p1, bemeneti_p2)

2. Második óra (vektorok, mátrixok)

1. **Sor/oszlopvektor létrehozása:**

    - Mindig szögletes zárójelbe [ ] kell tenni az elemeket.
    - Space vagy vessző használatakor sorvektor jön létre,
    - pontosvessző használatakor oszlopvektor.
      Példa: [2 1 4 3]/[6; 5; 7; 9]

2. **Transzponálás, vektorok szorzása:**

    - **Transzponálás:** A ′ jel használatával tehető meg.
      Példa: [1 2 3 4]′
    - Két vektor szorzásakor a .* műveletet kell használni, ez jelenti az elemenként való szorzást a matlabban.
      *Megjegyzés: A "szimpla csillagos" szorzás a mátrixszorzást jelenti, azaz egy $k \times l$-es és egy $l \times m$-es mátrix szorzásakor egy $l \times l$-es mátrix fog létrejönni. Amennyiben $k \neq m$, $A*B \neq B*A$. Az elemenkénti szorzás esetében minden elem meg fog szorzódni minden elemmel, és a mátrix mérete az A*B, B*A mátrixok szorzásakor létrejött mátrixok közül a nagyobbikkal fog megegyezni. (Ez sor és oszlopvektor szorzásakor igaz, nagyobb mártixok esetén a Matlab nem is engedi, hogy * műveletet végezzünk.)*
      Példa: A.*B
      Megjegyzés:A és B két tetszőleges, a kritériumoknak megfelelő vektort jelölnek.

3. **Vektorok összefűzése:**

    - Hasonlóan a vektorok létrehozásához, itt is csak vesszővel/pontosvesszővel elválasztva kell belőlük új vektort létrehozni.
      Példa: [sorvektor, oszlopvektor′]

4. **Indexelés:**

- 1-től indul, nem úgy mint C++ban.
- Az indexelés ()-el történik, a zárójelben a keresett elem indexét adjuk meg.
  Példa: sorvektor(3)
- Elem lekérése a vektor indexelésével történik.
  Példa: elem = sorvektor(10)
- Elem értékének megváltoztatása esetén elég lekérni az elemet és egyenlővé tenni új értékével.
  Példa: sorvektor(3)= 23
- Vektor hosszának lekérdezése a length() függvénnyel történik.
  Példa: length(vektor)
- A méret lekérdezése a size() függvénnyel lehetséges. Ez a függvény alapvetően mátrixok esetében használatos, azonban - mivel a vektor is egy egydimenziós mátrix - használható vektorok esetében is. Egy sorvektort ad vissza, ami tartalmazza a adott mátrix/vektor méretét. Vektor esetében tehát 2 visszatérési értéke lesz: a dimenziók száma, ami vektor esetében 1 és a vektor hossza.
  Példa: size(sorvektor)
- Speciális index: end (vége).
  Példa: sorvektor(end)

5. **Számokkal teli vektorok létrehozása:**

- Egy paraméterrel meghívva a függvényeket $n \times n$-es négyzetes mátrixot kapunk, kettővel $n \times m$-es mátrixot.
- **zeros:** Csupa nullával teli vektor létrehozása.
  Példa: zeros(3)
- **ones:** Egyesekkel teli vektor létrehozása.
  Példa: ones(5,2)
- **rand:** Random számokkal teli vektort hoz létre. (A számok nulla és egy közötti, double típusú értékek lesznek.)
  Példa: rand(9,4)
- **eye:** Egységmátrix generálására alkalmas.
  Példa: eye(7)
- **diag:** Diagonális mátrix létrehozására alkalmas, illetve -ha egy mátrixot kap bemenetül - akkor a mátrix főátlójában (diagonálisában) álló elemeivel tér vissza, mint oszlopvektor.
  Példa: diag(8), diag(peldamatrix)

6. **Adott felosztással vektor létrehozása:**
   Függvénnyel:

- 3 bemeneti értéket kell megadjunk, az első a mettől, a második a meddig érték lesz, a harmadik (legyen x) pedig azt adja meg, hogy hány részre ossza a tartományt a függvény. Ekkor egy x oszlopból álló sorvektort fogunk kapni.

Megjegyzés: a tartomány megadásánál fontos a sorrend, ha a mettől érték kisebb, mint a meddig érték - tehát tulajdonképpen növekvő sorrendben vannak megadva a határok - akkor növekvő sorrendben fogjuk megkapni a számainkat, ellenkező esetben csökkenő sorrendben. Ha a két tartományhatár megegyezik egymással, ugyanolyan értékekkel lesz tele a sorvektor, ez az érték pedig a tartományhatár lesz.

- **linspace:** Lineárisan elosztva generálja le a számokat, tehát azonos távolságra lesznek egymástól a kapott értékek.
  Példa: linspace(3,2,4)
- **logspace:** Logaritmikus beosztású vektort hoz létre.
  Példa: logspace(5,3,2)

Kettőspont operátorral:

- Abban az esetben, ha nem az elemek száma a fontos, hanem a felosztás pontos meghatározása, jó szolgálatot tehet nekünk a kettőspont operátor. Használata 3 érték megadásával történik, kettősponttal elválasztva őket. Az első és a harmadik érték adja meg a tartományt, a középső pedig a felosztást (lépésközt). textitMegjegyzés: Amennyiben a felosztás nem illeszkedik pontosan a tartományhoz, azaz a megadott felosztással a kezdőértéktől elindulva nem tudunk eljutni PONTOSAN a tartomány végéhez, akkor a tartomány vége nem lesz benne a sorvektorban. Lásd a példát.
  Példa: 1:3:9, erre a matlab a következőt adja ki: 1 4 7. Látható, hogy a 9-es elveszett a végéről.
  A kettőspont operátor használatos még tartományok bejárására is, pl az A mátrix(:,:,2:4) esetében a 2.,3.,4.-k dimenzión tudunk így végigmenni. Amennyiben csak egy kettőspontot írunk a tartomány helyére, az egészen végig fog menni a Matlab.

7. **Beépített függvények vektorokra:**

- A vektor minden elemére elvégzi a mehívott függvényt. Így a végén egy ugyanolyan méretű vektorral tér vissza, mint az argumentumban megadott mátrix.
- **sin, cos, tan, cot:** Az értékek szinuszának, koszinuszának, tangensének, kotangensének kiszámítása.
  Példa: sin(vektor)
- **abs:** Az egyes értékek abszolút értékét veszi.
  Példa: abs(vektor)
- **exp:** A vektorban szereplő adatok e^x-ével tér vissza (minden adatot az x jelöl).
  Példa: exp(vektor))
- **round:** Egészre kerekít.
  Példa: round(vektor)
- **sum:** Megadja a vektorban található elemek összegét.
  Példa: sum(vektor))
- **prod:** Ez egy rendkívül sokszínű függvény. Ha argumentumként nem üres vektort kap, akkor visszatér az elemek szorzatával.

Példa: prod(vektor)

- **min, max:** Egy visszatérési értékkel használva a minimummal/maximummal tér vissza, kettővel felhasználva a szélsőértékkel és a megtalált szélsőérték indexével tér vissza.
  Példa: [a, b] = min(vektor), ahol a minimum, b az indexe
- **find:** Az argumentumban logikai feltételt vár, visszatérési értéke azoknak az elemeknek az indexe, amelyek megfeleltek a feltételnek.
  Példa: find(A>2)
- !!! Find segítségével a következőképpen kaphatóak meg azok az elemek, amik megfeleltek: indexek = find(A>2)-vel meghatározzuk a "helyes" elemek indexeit. Ezek után egy következő sorban lekérjük az A azon elemeit, amelyeknek az indexe benne van ebben az indexmátrixban.Tehát: A(indexek). Ez ki fogja írni ekkor a feltétel(ek)nek megfelelő elemeket.

8. **Logikai operátorok, indexelés:**

- A "kacsacsőr" van elöl: <= (helytelen: =<)
  Megjegyzés: Mindig skalárt használjunk az összehasonlításnál, két mátrixot nem tudunk például összehasonlítani ilyen módon!
  Példa: A>=5
- **És/vagy operátor:** Vektorok esetén használható a & (elemenként), egyébként a && jelöli az és-t. (Vagy operátornál hasonlóan, vektoroknál használható az |, máskor a || használandó.)
  Megjegyzés: Ha az &&-el összekötött logikai kifejezés első tagja hamis, a kifejezést nem értékeli ki tovább a Matlab. Ezt hívják "rövidrezáró" működésnek.
  Példa: vektor$_1$ & vektor$_2$
- **Logikai indexelés:** Logikai indexelés során egy, az eredeti vektor hosszával megegyező logikai vektort kapunk vissza. Ebben a 0, 1 számok valamilyen sorozata fog megjelenni, ahol az 1 jelentése az, hogy az adott elemre teljesül a feltétel, a 0-é pedig, hogy nem teljesült rá a feltétel.
  Példa: vektor$_1$ & vektor$_2$
- !!! Logikai indexeléssel elemek kinyerése egy vektorból: indexek = A>4, majd A(indexek).

9. **Mátrix alapműveletek:**

- A **size, ones, zeros, rand, diag, eye** függvények a vektoroknál leírtakkal analóg módon működnek.
- **sum, prod, mean, min, max** Szintén a vektorokhoz hasonlóan lehet alkalmazni őket, de tudni kell róluk, hogy dimenzióként működnek.
- **squeeze:** Eltávolítja az 1 kiterjedésű dimenziókat. (Azaz ha a mátrix mérete mondjuk $4 \times 1 \times 2 \times 3$, akkor az 1-es méretű dimenzió el lesz távolítva, és a mátrix $4 \times 2 \times 3$-as méretű lesz.)
  Példa: squeeze(A(:,:,2))

- **reshape:** Átméretezi a mátrixot. (Fontos: Az újonnan létrejövő mátrix méreteinek szorzata az eredeti mátrix méreteinek szorzatával kell, hogy egyenlő legyen.)
  Példa: Az eredeti A mátrix 5×5-ös. Egy lehetséges átméretezés ekkor:
  B = reshape(A, 1,25)
- **numel:** Megadja a mátrix összes elemének számát.
  Példa: numel(A)

3. Harmadik óra (vezérlőszerkezetek, formázott kiíratás, polinomok, 2D ábrázolás)

1. **Elágazások**:

- IF-fel kezüdnk, ELSEIF, ELSE jöhet utána hasonlóan, mint C++-ban
- Itt nincs zárójelezés, az elágazás végét az END jelzi
- A feltétel logikai értékét vizsgálja (ami nem 0, az igaz)
- **Relációs operátorok**:
  == (igaz, ha egyenlő a két elem, hamis, ha nem),
  = (igazzal tér vissza, ha a két elem nem egyenlő, hamissal, ha egyenlő),
  < (igaz, ha a kacsacsőr bal oldalán álló kifejezés kisebb, mint a jobb oldalon álló),
  > (igaz, ha a kacsacsőr bal oldalán álló kifejezés nagyobb, mint a jobb oldalon álló),
  <= (igaz, ha a baloldali kifejezés kisebb vagy egyenlő, mint a jobboldali),
  >= (igaz, ha a baloldali kifejezés nagyobb vagy egyenlő, mint a jobboldali)
- **Logikai operátorok**:
  – **&** (logikai "és", A & B esetén akkor lesz csak igaz, ha A és B is igaz)
  – **&&** ("szigorúbb és", ha A && B kifejezésnél A hamis, akkor B-t már le sem ellenőrzi)
  – **|** (logikai "vagy", A | B igaz lesz, ha legalább az egyik kifejezés igaz, és akkor lesz hamis, ha mindkettő kifejezés hamis)
  – **||** ("szigorúbb vagy", ha A || B kifejezésnél A igaz, akkor B-t már meg sem nézi)
  – **~** (tagadás, ~A akkor lesz igaz, ha A hamis, és akkor hamis, ha A igaz)
  – **xor(A,B)** (-> akkor lesz igaz, ha A és B logikai értéke különbözik, és akkor lesz hamis, ha logikai értékük megegyezik)
  – **all** (az all(A) akkor lesz igaz, ha A minden eleme igaz (vagyis nem nulla), és hamis lesz, ha legalább egy eleme nulla/hamis)
  – **any** (any(A) igaz lesz, ha A legalább egy eleme igaz/nemnulla, hamis lesz, ha minden eleme hamis/nulla)
- **logical(A)**: átkonvertálja az A mátrix/vektor elemeit logikai kifejezésekbe, azaz 1-esekbe és 0-kba

2. **For-ciklus**:

   - Ismert számú iteráció elvégzésére
   - Nincs zárójelezés, a blokk végét az end jelzi
   - A ciklusváltozót sorvektorként definiáljuk, a ciklus törzsében értéke az aktuális elemnek megfelelő skalár
   - A ciklusváltozó tetszőleges sorvektor lehet
   - **Példa:**

   ```
   for n = 1:10
       [parancsok]
   end
   ```

3. **While-ciklus:**

   - Ismeretlen számú iteráció elvégzésére, egy feltétel teljesüléséig
   - Nincs zárójelezés, a blokk végét az end jelzi
   - Nincs explicit módon megadott ciklusváltozó
   - Figyeljünk a végtelen ciklus elkerülésére!
   - **Példa:**

   ```
   while [feltetel]
       [parancsok]
   end
   ```

4. **Minimumkeresés mátrixban:**

   - min(m,[],'all'); (m a mátrixunk, 'all' jelenti azt, hogy az összes elem közül keresük a minimumot)

5. **Switch:**

   - Megadhatjuk, hogy különböző esetekben milyen különböző parancsokat hajtson végre a program
   - Hasonló az if-else if-else if... struktúrához
   - **Példa:**

   ```
   in = input('Irj be egy szamot: ');
   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 jo erteket adtal meg...');
end

6. **Formázott kiíratás:**

- (Ismétlés:)**disp():** szöveg vagy változótartalom egyszerű kiíratása
- **fprintf** - szöveg kiiratása formázott-beágyazott mezőkkel (vagy fájlba, vagy a konzolra); különösen fontos: formatSpec rész a help-ben
- **FormatSpec:** Egy szöveg kiíratásánál meg lehet mondani a formátumot amiben a számot ki szeretnéd írni: pl. %2.4f\n\t esetén a % jelenti a mező nyitását, 2 a minimum mező szélességet, . a decimális pont, 4 a tizedesjegyek száma, f a fixpontos típust jelenti, \n és \t pedig a sortörést illetve tabulálást.
  Ez a kódban így néz ki:
  fprintf('Teszt: Tizedes: %2.4f\n', 10*pi)
  Output: Teszt: Tizedes: 31.4159
- **sprintf** - mint fprintf, csak string-változóba "nyomtat"
- **rats** - racionális számmal közelít, bizonyos/adott tolerancia mellett
- **format** - lebegőpontos számok kijelzésének pontossága
  format short; - 4 tizedesjegyig írja ki a számot
  format long; - 15 tizedesjegyig is kiírja
  format shortE; - 3.1416e-02 alakban írja ki (normálalak)
- **num2str** - szám → string konverzió
  num2str(szám1,szám2) alakban megadva a szám1-et szám2 darab számjegyes pontossággal konvertálja
- **mat2str** - mátrix → string konverzió

7. **Polinomok:**

- sok függvény és valós folyamat leírható magasabb rendű polinomokkal
- MATLAB-ban a polinomokat az együtthatóvektorukkal reprezentáljuk:
  $P_1(x) = x^2 + 2x + 3- > P_1 = \begin{bmatrix}1 & 2 & 3\end{bmatrix}$;
  $P_2(x) = 10x^3 + 4x^2 + 5x - 7- > P_2 = \begin{bmatrix}10 & 4 & 5 & -7\end{bmatrix}$;
  $P_3(x) = 3x^4 + 5x^2 - 12- > P3 = \begin{bmatrix}3 & 0 & 5 & 0 & -12\end{bmatrix}$;
  P_1, P_2, P_3 itt egy egyszerű sorvektor!

8. **Polinomokhoz tartozó függvények:**

- Az általunk megadott vektorokat a MATLAB megfelelő beépített függvényei fogják polinomként értelmezni
- Gyökök kiszámítása: roots(P);
- A P polinom kiértékelése adott x_0 pontban: y_0 = polyval(P, x_0);

- Kiértékelés sok pontban:
  x = -1:0.01:1; (itt x tartalmazni fogja a számokat -1-től 1-ig, 0.01-es lépésközzel)
  y = polyval(P, x);
- Polinom létrehozása a gyökeiből:
  (gyökök "vektora":) r = [-1 1];
  P = poly(r);
- Polinom illesztése adatsorra:
  P = polyfit(x_t, y_t, fokszam);

9. **2D-s ábrázolás:**

- **figure** - új ábra létrehozása
- **plot(x), plot(x, y)** - adatsor kirajzolása
  egyargumentumú esetben: a vektor értékei a vektorindexek függvényében
  kétargumentumú esetben: x -> helyek, y -> értékek
- **stem(x), stem(x, y)** - pálcikadiagram, paraméterezés az előzőhöz hasonlóan
- **stairs** - lépcsődiagram
- **hold on/off** - egy rajzfelületre több görbét is kirajzolhatunk (felülírás az alapértelmezett)
- **subplot** - az ábránkat alábrákra oszthatjuk szabályos rácsos felosztással
  pl. subplot(2, 3, 4), ahol 2 a sorok száma, 3 az oszlopok száma, 4 annak az alábrának az indexe, amire rajzoltatni akarunk (sorfolytonosan megy az index)
- rajzolás több cellába: subplot(2, 3, [1,4])
  (így az 1-es és 4-es cella "össze lesz olvasztva")
- **grid on/off** - a grafikonhoz hátsó hálóbeosztás kérhető
- **legend** - adatsorok feliratozása, legend utáni zárójelben felsorolhatjuk az ábravonalaink neveit, illetve elhelyezkedését is megadhatjuk 'Location'-el
  Példa: legend('Lower limit', 'Upper limit', 'Location', 'northeastoutside');
- linespec: plot parancsnál meg lehet mondani a vonal stílusát
  Példa: plot(x,y,'-xg') -> sima összefüggő zöld vonallal, x alakú markerekkel legyen a vonal (nagyon sok féle van, matlab helpben bent van az összes)
  Ha a plotot elmentjük egy változóban, hogy később adatot lehessen lekérni (**p1 = plot(x, y);**), akkor a set() függvénnyel is be lehet állítani a vonal stílusát: **példa: set(p1,"Color",'blue'); set(p1,"Marker",'.');**
- **xlim([hatar1, hatar2]), ylim([hatar1,hatar2])** - tengelyhatárok beállítása (box on/off beállítással még meg is vastagítja nekünk a tengelyeket - 3D-ben használják inkább)
- **xticks, yticks:** ezeket ez értékeket írja ki a tengelyeken
  Példa: xiticks(0:5:30) - 0-tól 30-ig 5-ösével lépkedve kiírja az értékeket
- **text, xlabel, ylabel, title** - csak úgy szöveg, x- és y-tengely felirat, ábracím

Ezt a szöveget utána még tudjuk formázni (megadni a méretét, félkövérré tenni stb.) pl. 'FontSize', 14, 'FontWeight', 'bold' stb. (ezek is mind megtalálhatóak a Helpben)
Példa: xlabel('X tengely', 'FontSize', 12)

- Használtuk még érintőlegesen a get() függvényt is:
  element = get(dataset,index) megadja a dataset "indexedik" elemét

4. Negyedik óra (lineáris egyenletrendszerek, spektrális elemzés, leképezések)

1. Mátrix osztások/szorzások:

   - *Emlékeztető:* $Ax = b$-t átrendezve (jobbról A^{-1}-el szorozva) kapjuk, hogy $x = A^{-1}b$. Ebből is látható, hogy a mátrixok körében a szorzás nem kommutatív, ezért kétféle osztás definiálható.
   - **bal osztás:** A\b azt az x mátrixot jelenti, amelyre A*x=b, azaz A\b = $A^{-1}b$ = x
   - **jobb osztás:** b/A azt az x mátrixot jelenti, amelyre x*A=b azaz b/A=bA^{-1}=x
   - Inverz számolást nem csak bal osztással, hanem beépített inverz számoló függvénnyel is végezhetünk. (Ez gyorsabb, pontosabb általában.)
     Példa: inv(A)

2. Sajátérték, sajátvektor:

   - *Emlékeztető:* $\lambda x = Ax$ (λ a sajátérték, A az áttérési mátrix, v a jobboldali sajátvektor)
   - *Emlékeztető:* $\lambda x = uA$ (λ a sajátérték, A az áttérési mátrix, v a baloldali sajátvektor)
   - **eig:** Paraméterként egy mátrixot kell átadni, a visszatérési értékek száma szerint pedig így osztályozhatjuk a függvényt:
     - **egy** visszatérési értékkel: a mátrix sajátértékeit tartalmazó oszlopvektorral tér vissza
       Példa: eig(A)
     - Ha egy visszatérési értékkel hívjuk a függvényt, de megadunk neki egy második paramétert, akkor mátrix formában adja vissza a sajátvektorokat (=diagonális mátrix)
       Példa: eig(A, 'matrix')
     - **két** visszatérési értékkel: A (normált) jobboldali sajátvektorokat tartalmazó mátrixxal és a sajátértékeket tartalmazó diagonális mátrixxal tér vissza.
       Példa: [V,D] = eig(A)
     - **három** visszatérési értékkel: A baloldali normált hosszú sajátvektorokat is visszaadja a fentebb említett jobboldali SV-ket, SÉ-ket tartalmazó mátrixok mellett.
       Példa: [V,D,U] = eig(A)

3. Komplex számok:

   - **real:** Egy komplex szám valós részét adja meg. (Használható természetesen mátrixokra is, ekkor egy ugyanolyan méretű mátrixxal tér vissza, melyben a valós részek vannak.)
     Példa: real(A)
   - **imag:** A képzetes részt adja meg. Használata a real függvénnyel analóg.
     Példa: imag(A)
   - **isreal:** Eldönti a megadott inputról, hogy abban csak valós számok vannak-e, ennek alapján egy igaz-hamis (1-0) értékkel tér vissza. (Tehát fontos megjegyezni, hogy ÖSSZES elemet tekinti, egyetlen komplex szám esetén is hamis visszatérési értéket ad.)
     Példa: isreal(A)

5. Ötödik óra (differenciálszámítás, integrálszámítás)

1. **Numerikus deriválás:**

   - Diszkrét értékeken történik, ezért fontos a jelek felbontása (alacsony felbontással mintavételezve egy folytonos függvényt nem kapunk "kellőképpen" folytonos értékeket);
   - a differenciálhatóság feltétele általában a függvény folytonossága (vannak persze kivételek);
   - MATLAB-ban teljes folytonosságról nem beszélhetünk (minimális lépésköz eps), de megfelelő felbontást választva jó közelítéssel numerikusan is kiszámítható a derivált;
   - MATLAB-ban a derivált mindig differencia-hányados.

2. **Differenciahányados számítása:**

   - **diff(vektor)** - előállítja az elemenként vett különbségeket, n elemű vektorból n-1 elemű vektort képez, melynek első eleme az eredeti vektor 1. és 2. elemének különbsége, 2. eleme az eredeti vektor 2. és 3. elemének a különbsége...stb.
   - **Differenciahányados:** előállítjuk az értékkészlet és az értelmezési tartomány adatsorainak különbségét (dy = diff(y); dt = diff(t);), majd ezeknek vesszük a hányadosát: differenciahanyados = dy ./ dt;

3. **Deriválás adott pontban:**

   - Ha a vizsgált függvény képlettel felírható (pl. polinom, szögfüggvény, stb.), akkor adott hosszúságú és felbontású mintavétel nélkül is megadható egy adott pontban a derivált;
   - Ehhez a vizsgált pont tetszőlegesen kicsi környezetét megvizsgáljuk

4. **Differenciálás görbeillesztés segítségével:**

- Ha az adatainkra jól illeszkedik egy differenciálható függvény görbéje, akkor az illesztett görbe adott pontbeli deriváltja alapján becsülhetjük az adataink deriváltját.
- **Polinom:** Ha csak egy vagy néhány pontban van szükség a deriváltra, akkor megtehetjük, hogy néhány (3-5) egymást követő pontra illesztünk polinomot. Ha a pontok számánál eggyel kisebb fokszámú polinomot választunk, az pontosan fog illeszkedni minden pontra. Ha előre nem tudjuk az illesztednő polinom fokszámát, akkor többféle fokszámú polinomot illesztve valamilyen kritérium alapján kiválasztjuk a legjobban illeszkedőt. Törekedjünk a lehető legalacsonyabb fokszámú polinom használatára.
  Így néz ki:
  pf = polyfit(x, y, 4); (ez korábban le volt írva 3.óránál)
  dpf = polyder(pf); (a polyder magadja a polinom deriváltját)
  dy = polyval(dpf, x);
  plot(x, y, 'x', x, polyval(pf, x), '-', x, dy, '–');
- **Gauss-keverék (gaussn):**
  - Ebben az esetben az illesztés után kapott modellt kell kézzel deriválni, amivel aztán meg tudjuk becsülni az adatsorra a differenciálhányadost.
  - **fit** függvény, az illesztendő függvény típusa **gaussn**, ahol n a Gauss-keverék komponenseinek száma (a csúcsok száma alapján becsülhető)
  - fitted = fit(x', y, 'gauss3', 'StartPoint', $\begin{bmatrix}0.6 & 5 & 0.6 & 0.4 & 8 & 0.15 & 0.2 & 9 & 0.07\end{bmatrix}$); gauss3 a típus, a StartPoint paraméterek segítenek a programnak még jobb illeszkedésű görbéket találni, de ez bonyolult, nem kell tudnunk elvileg

- **Trigonometrikus illesztés:**
  fit függvény, az illesztendő függvény típusa sin(n) (vagy esetleg fourier(n)), ahol n a trigonometrikus komponensek száma
  Gauss-keverék és trigonometrikus függvény illesztése esetén a deriválást nekünk magunknak kell kézzel elvégezni, és az így kapott függvény segítségével tudjuk becsülni a differenciálhányadosokat

5. **Numerikus integrálás:**

- A deriváláshoz hasonlóan lehet vektorértékek és megadott függvény alapján is integrálni (integrál := a függvényértékek és az x-tengely közötti területrészek előjeles összege);
- Lehetőségek: egyszerű összeadással, trapézszabály alkalmazásával, vagy megadott függvény alapján
- **Egyszerű összeadással:**
  x = linspace(2*pi, 2.5*pi, 10);
  y = x .* sin(x);
  szelesseg = x(2)-x(1);
  osszeg_1 = szelesseg*sum(y);
  figure;
  hold on;

bar(x, y, 1, 'w', 'EdgeColor', 'b', 'LineStyle', '-'); (oszlopdiagram, érdemes rákeresni a helpben, mert sokféle van. Egy változó megadásával ugyanúgy "index szerint" megy mint a plot, két változóval x-et y szerint plottolja. Mátrixot is meg lehet adni neki.)
plot(x, y, 'r*-');
title(strcat('egyszeru osszeadassal: ', num2str(osszeg_1, 5)), 'FontSize', 14);

- **Trapézszabállyal:**
  x = linspace(2*pi, 2.5*pi, 10);
  y = x .* sin(x);
  osszeg_2 = trapz(x, y); (y-t integrálja x értékeinek függvényében)
  figure;
  hold on;
  stem(x, y);
  plot(x, y, 'r*-');
  title(strcat('trapezszaballyal: ', num2str(osszeg_2, 5)), 'FontSize', 14);
- **Anonim függvények:**
  - A MATLAB lehetőséget ad arra, hogy függvényeket "tároljunk" változókban, (ha azok kellőképpen egyszerűek);
  - A konstrukció: **fv = @(x) x + 3;** ahol fv a függvény neve, @(x) jelenti a bemenő paraméter(eke)t, x + 3 pedig a bemenő paraméterektől függő kifejezés. Ezután meg tudjuk hívni a függvényt értékadással:
    Példa: fv(3) esetén visszakapjuk 3 + 3 értékét (ha a függvényünk a fenti példa)
- **Numerikus integrálás függvénnyel:**
  x = linspace(2*pi, 2.5*pi, 10);
  fv = @(t) t .* sin(t);
  osszeg_3 = integral(fv, x(1), x(end)); (integrálja a 'fv' függvényt az x(1) - x(end) intervallumon)
  fprintf(['Elojeles osszegek:egyszeru osszeadassal: %6.4ftrapezszaballyal: "%6.4ffuggvennyel: %6.4f'], osszeg_1, osszeg_2, osszeg_3);
- **Megjegyzések:**
  integrálásnál a sima összeadást lehetőleg ne használjuk;
  tetszőleges vektoros adatsorokhoz: trapézszabály;
  anonim függvényekkel felírható görbékhez: integral (quad) függvény

6. Hatodik óra (differenciálegyenletek)

1. **Ismétlés:**

- DE: Olyan egyenlet, amelyben az ismeretlen egy függvény, és szerepel benne ennek az ismeretlen függvénynek valamely deriváltja is.
- DE rendje: az ismeretlen függvény legmagasabb fokú deriváltjának fokszáma.

- A megoldás során az f(t) függvényt keressük.

$$f'(t) = g(f(t), t)$$

- Alapvetően numerikus integrálással kellene dolgoznunk, ez azonban nem mindig pontos, ezért beépített DE megoldóval szoktunk dolgozni.
- **ode45:** 3 paraméterrel hívjuk általában, sorrendben: F-el, a megadott függvénnyel, a t által bejárt intervallummal (pl [0,3], tulajdonképpen ez az ÉT), illetve a kezdeti feltételek értékével, y(0) = c-vel, ahol c tetszőleges konstans vagy intervallum. (Pl ha y(0) = 1, akkor 1 lesz ez a harmadik bemeneti paraméter.)
  Példa: [t45,y45] = ode45(F,[0 3],1)
  ahol t45, y45 a két visszatérési érték (ábrázolásnál t az x tengely, y marad az y).
- F megadása pl így történhet: **F = @(t,y) 2*y**
  ahol a függvény maga a 2*y, a többi az szintaktika

7. Hetedik óra (3D ábrázolás, LaTeX)

1. **3D ábrázolás:**
   - A MATLAB beépített függvényekkel lehetőséget biztosít pontfelhők, görbék és felületek hatékony térbeli ábrázolására.
   - A *plot*-hoz hasonlóan ezek is értelmezési tartomány (ÉT) értékkészlet (ÉK) alakban működnek, ahol ÉT és ÉK is lehet R, R^2,R^3 (az ábrázolt alakzat függvényében, részletek később)
   - Jellemzően 3D-ben ábrázolt alakzatok:
     pontfelhők(x, y, z koordináták)
     (parametrikus térgörbék (R -> R^3))
     felületek (R^2 -> R)
     vektormezők(R^2 -> R^2, R^3 -> R^3)
2. **Pontfelhő:**
   - parancs: **plot3()**
     Példa: pl = plot3(x, y, z, '.'); (így még a pl változóba is kimenti, valamint az x,y,z koordinátákat ábrázolja pontokkal a '.' vonalmegadás miatt)
3. **Felületek ábrázolása:**
   - R^2 -> R leképezés ((x,y) pontból f(x,y)-ba képez)
   - Az értelmezési tartományhoz kell egy térháló, amelyen kiszámoljuk az adott felület értékeit:
   - **meshgrid** – ez hozza létre a térhálót általunk megadott tartományon, tetszőleges felbontással
     - így néz ki: [X,Y] = meshgrid(2:4,1:5);

- A meshgrid lényegében koordinátamátrixokat hoz létre, amihez egy rajzoló függvénnyel hozzá tudunk rendelni értékeket, így tudjuk ábrázolni ezeket 3D-ben.
- bemenete: 2db vektor, a fenti példában $\begin{bmatrix}2 & 3 & 4\end{bmatrix}$, $\begin{bmatrix}1 & 2 & 3 & 4 & 5\end{bmatrix}$
  ha csak egy bemenetet adunk meg neki, akkor úgy veszi, hogy ugyanazt a vektort adjuk meg neki mindkét bementetben (tehát meshgrid(x) = meshgrid(x,x) valójában)
- kimenete: 2db bemenet1xbemenet2 méretű mátrix, az elsőben az első bemenet elemi, a másodikban a második bemenet elemei vannak úgy elrendezve, hogy ha a 2 mátrixból vesszük az összetartozó elemeket, akkor a két bemenet meghatározta térháló koordinátáit kapjuk (amikhez aztán valamilyen függvényértéket rendelhetünk)

- Rajzoló parancsok:
  - **surf:** A surf(X,Y,Z) parancs egy (rácsos) felületet rajzol ki (a rácspontok közti területet is kitöltve valamilyen szabály szerint)
    a Z-ben vannak az X-től és Y-tól függő értékek, ezeket ábrázoljuk magasságként és színként a 3D-s ábrán
  - **mesh:** a mesh(X,Y,Z) parancs csak egy rácsozatot rajzol ki, nem színezi ki a rácsok közötti területet, csak a rácsot magát
  - **contour:** (csak) szintvonalak ábrázolása,
    - így néz ki: C = contour(X,Y,Z);
    - szintvonalak feliratozása: **clabel(C);**
    - **colorbar;** - az ábra mellé odarakja a színskálát is (nem csak contournál működik, hanem mindegyiknél)
    - **contour3()** - 3D-s szintvonalak
    - **contourf()** - a szintvonalak közötti részt színezi ki
  - **surfc:** surf felület és szintvonal kirajzolása egyben, függvénymeghívás ugyanúgy, mint az előzőeknél (nem muszáj változóba is menteni a függvényrajzolást)
  - **meshc:** mesh felület és szintvonal kirajzolása, függvénymeghívás itt is ugyanúgy
  - **quiver:** vektormező kirajzolása (3D-sen: quiver3)
    - (x,y)-ból (f(x,y), g(x,y))-ba képez
    - quiver(u,v) - ahol u-t és v-t szintén meshgriddel kapjuk meg

4. **A view használata:**

- **view(az,el) :** forgatjuk az ábrát, így más szemszögekből, nézőpontokból láthatjuk
- A view parancs mindig az aktuális figure-re lesz érvényes, nem kell másként hivatkozni rá.
- *az = azimuth*: z - tengely körüli horizontális forgatás fokban, negatív y - tengelytől kezdődik, óramutató járásával ellenkező irányban forgat
- *el = elevation (magasság)*: +, ha az ábrát fölülről nézzük, − ha az ábrát alulról nézzük

- **Példa a szerkezetére:**
```
El = 30;
for Az = 0:180
   view(Az,El)
   pause(0.1)
end
```

5. További függvények:
   - **Peaks** - nem kell nagyon tudni, bonyolult, de a lényeg, hogy a peaks(n) visszatér egy n-szer n-es mátrix-szal, amivel könnyebb(?) a felületeket kirajzolni
   - **gradient:** a Z=f(x,y) függvény x és y irányú gradiensét számítja ki (dx,dy) , ezek adják a nyilak irányát és hosszát ([X,Y] = gradient(Z);)
   - **clf:** törli az előző ábrát a figure ablakból
   - **surfnorm:** a 3D-s (egységnyi hosszú) normálvektorok kiszámítására
   - **axis square:** négyzet, vagy kocka alakú ábra (a tengelyeket azonos hosszúra rajzolja), alapértelmezetten a megjelenő ábra arányai nem ilyenek
6. **Matematikai szövegek, TeX interpreter:**
   - Az ábrán megjelenő szövegekben szereplő matematikai kifejezések elegáns megjelenítéséhez a TeX (szedő és tördelő programnyelv) matematikai módjának elemei használhatóak. Az xlabel, ylabel, title, stb. függvények a TeX matematikai módjának megfelelően írt karakterláncokat alapértelmezésben képesek megfelelően értelmezni.
   - Ugyanaz az írásmód, mint itt LaTexben a matematikai képleteknél
   - Link, ahol megtalálhatók ezek a képletek: `https://en.wikibooks.org/wiki/LaTeX/Mathematics`

8. Nyolcadik óra (cellatömbök, struktúrák, fájlműveletek)

1. **Cellatömb:** Olyan adattípus, melyet különböző típusú és/vagy méretű változók tárolására használhatunk. (Az egyes cellákban használhatunk különböző típust, de egyetlen cellán belül nem lehet többféle típus. Kivéve pl, ha a nagy cellatömb egyik cellájába még egy cellatömböt rakunk.)
2. **Cellatömb létrehozása:**
   - Üres cellatömbként a **cell()** függvénnyel. Bemeneti paraméterként a sorok, oszlopok (esetlegesen dimenziók) számát várja.
     Példa: cell(2,3,4)
     ekkor egy $2 \times 3 \times 4$-es cellatömböt fogunk kapni.
   - elemeinek felsorolásával is lehetséges.
     Példa: C={2, ['a', 'b'; 'c', 'd']; eye(3), [-1;2]}

Ebben az esetben is figyeljünk rá, hogy ugyanúgy mint a vektorok létrehozásánál, space-l, illetve vesszővel tudjuk elválasztani/beírni azokat az adatokat, amiket egy sorba szeretnénk tenni, míg pontosvesszőkkel az oszlopokat koordinálhatjuk.
Illetve a cellatömb a karaktereket, stringeket egybe fogja olvasni, és egy változóban, cellában fogja eltárolni őket.

3. **Indexelés:**

   - A cellatömb elemeinek indexelése kapcsos { } zárójelekkel történik. (Az keresett/megadott indexnél tárolt értéket adja vissza a matlab.)
     Példa: C{2}
     ahol C egy tetszőleges, minimum 2 elemet tartalmazó cellatömb, és mi a második elem értékét keressük.
   - Adott indexen tárolt értéket is ezzel a zárójelezéssel tudunk megváltoztatni.
     Példa: C{2, 1}= 'abcdf'
   - Rész-cellatömb indexelése "sima" ( ) zárójelekkel történik.
     Példa: C(4)
     ekkor a C cellatömb 4. celláját kapjuk vissza, mint egy 1×1-es cellatömb, ugyanazokkal az adatokkal, mint ami C-ben is volt a 4. helyen.

4. **Cellatömb megjelenítése:**

   - **cellplot:** Bemenetként csupán a megjelenítendő cellatömböt várja, és a plot függvényhez hasonlóan, külön figure-ként jeleníti meg a cellatömböt.
     Példa: cellplot(C)

5. **Struktúrák**

   - A C++, C nyelvek struct típusához hasonló, van egy változónév, és azon belül hozhatunk létre újabb változókat/adattárolókat. Ezek az adattárolók névvel vannak ellátva, lehetnek különböző típusúak/méretűek a cellatömbhöz hasonlóan. (A különbség a kettő között, hogy itt mindennek van valamilyen címkéje, nem "csak úgy" kerülnek be a különböző adatok a tárolóba.)
   - Struktúra létrehozása egyszerűen a változónév (például hallgató), és azon belül az egyes mezőnevek megadásával történhet. (A structhoz hasonlóan .-al választhatjuk el őket. A változónevet pedig nem kell külön létrehozni, az automatikusan létrejön, ha elkezdjük feltölteni a struktúrát adatokkal!)
     Példa: hallgato.nev = 'Pitypiritty Palkó', hallgato.szulhely = 'Üveghegyi Barlangodú'

6. **Elérési útvonalak**

   - Fontos arra odafigyelnünk, hogy csapatmunka esetén az egyes személyeknél más lehet az elérési útvonalakban az elválasztó karakter (pl \ vagy /). Ebben az esetben használhatjuk a **filesep** parancsot, ami megadja az aktuálisan használt elválasztó karaktert.

Példa: filepath = mappa filesep uticelmappa filesep fajl
Tipp: érdemes a filesep-et egy rövidebb, pl f nevű változóba kimenteni, így kevesebbet kell majd írnunk, illetve a kód is rövidebb, átláthatóbb lesz

- **pwd:** Az aktuális könyvtár elérési útvonalát kérdezhetjük le vele.
  Példa: pwd
- **dir:** Az aktuális könyvtárban a fájlok kilistázására alkalmas. Használható szűrőkkel is.
  Példa: dir
  Példa: dir([pwd filesep '*.mat']) Itt a '*.mat' rész az azt jelenti, hogy azokat a fájlokat keressük, amiknek .mat végződésük van. A csillag tulajdonképpen helyettesít mindent, annak a helyén bármi lehet. (Linuxosoknak ismerős lehet a szintaktika.)

7. **Fájlkezelés - formázott szövegek**

- Előnye, hogy látszik a tárolt adatok formátuma is. Használható logok készítésére, vagy egyszerű adatok olvasható tárolására.
- **fprintf:** Formázott kiiratás. Meg kell adni neki a kiiratás "mikéntjét", azaz hogy milyen típusként, milyen hosszan, stb. írja ki az adott adatot, illetve természetesen magát a kiiratandó adatot is.
  Példa: fprintf('%d\n',round(1.442))
  ahol az 1.442-őt kerekítve, integerként (%d), új sorokba iratjuk ki
  Példa: fprintf(fajl,'%d')
  Példa: fprintf(fajl,' ez egy plusz szoveg: %d amit barhova bele lehet irni')
- **fscanf:** Beolvasás, 3 bemeneti paraméter: mit, hogyan (milyen formátumban) és hányat (dimenziók száma)
  Példa: fscanf(fajl,'%f', 4)
  ahol a fajl nevű megnyitott fájlból olvasunk be 4 db lebegőpontos (%f) értéket
- **textscan:** Ez is beolvasás, de cellatömbbe olvas be, ezért megmaradnak a különböző adattípusaink is beolvasás után, úgy ahogy az eredeti fájlban is voltak.
  Példa: textscan(fajl,'%f', 4)
- Hasznos formázások:
  – %f lepegőpontos szám
  – %s vagy %d integer
  – %.2 2 tizedesjeggyel kiiratva
  – inf: a végéig olvas be
  – ezt: [2 inf] dimenzióként megadva a végéig olvassa be a számokat (2 oszlopban)

8. **Fájlkezelés - bináris szövegek**

- A matlab is ilyen fájlokkal dolgozik, illetve beépített tárolási módja is bináris fájlokat generál/olvas be (.mat).
- Nem derül ki a fájlból a formátum, és a fájl csak programmal olvasható, ugyanakkor kompaktabb adattárolásra ad lehetőséget.

- **fwrite:** Adatok fájlba történő kiírására ad lehetőséget, binárisan. Első paramétere a hova (fájl), második a mit (lehet változó is, lehet - ahogy a példában is - megadott számintervallum pl. Hívható harmadik paraméterrel is, ekkor megadhatjuk a típust, ahogyan ki szeretnénk íratni.
  Példa: fwrite(fid,[1:9], 'double')
  ahol 1-9-ig kiiratjuk a számokat a fid nevű fájlba. (De megadhatunk itt más tömböt is.)
- **fread:** Egy már megnyitott, bináris fájl olvasására használhatjuk. Hívása a file nevével/ID-jával történik. (Hívható így is: **fread(fid, inf, 'double')**, ahol a inf a végéig olvasás, double a típus megadása.
  Példa: fread(fileID)
- **save:** A fájl elmentésére szolgál. Használható csak az elmentendő fájl nevével hívva is. (Nem kell ', vagy " hozzá!) Megadható neki paraméterként azt is, hogy mit mentsen.
  Példa: save(fid), save pelda.mat
  Példa: save(fid, 't', 'y'), ekkor a t és y változókat mentettük bele
- **load:** Egy .mat fájl betöltésekor a függvény megnyitja és betölti a fájlban található változókat.
  Példa: load('pelda.mat'), load pelda.mat
  Példa: load('pelda.mat', 't'), ha tudjuk, hogy van benne egy t nevű változó és csak azt szeretnénk betölteni

9. **Fájlok megnyitása, bezárása:**

   - **fopen:** Fájlok megnyitására alkalmas (a megnyitás olvasásra történik).
     Példa: fid = fopen('pelda.mat'), fid = fopen(filepath), ahol a filepath nevű változóban megadtuk az elérési útvonalat
   - **fclose:** Fájlok megnyitására alkalmas.
     Példa: fclose(fid)

9. Kilencedik óra (ábrák mentése, GUI, fájlkezelés)

1. **Ábrák mentése:**

   - Az általunk készített plotokat több formátumban is el lehet menteni
   - File -> Save As
   - **saveas(fig,filename,formattype);** //fig a matlabos változód neve, amit el akarsz menteni, filename az, hogy mi legyen a neve mentés után, formattype pedig a formátuma
   - Legfontosabb fájlformátumok:
     - többféle kép: .png, .jpg, .tif, .bmp
     - .pdf

- skálázható vektorgrafikus file: .svg
- Matlab Figure file: .fig: tartalmazza az adatot, és mindegyik plot tulajdonságot vissza tudjuk tölteni és meg tudjuk változtatni a tulajdonságait

2. **A readImg() függvény:**

- **img = imread(filename);** - a beolvasott képtől függően (RGB vagy greyscale) hoz létre egy uint8 mátrixot, mivel a pixelek 8 inten tárolódnak a [0, 255] intervallumban.
  - Beolvassa a megadott fájlnévből a képet:
  - RGB esetén: width x height x 3 méretű mátrix lesz az img
  - GreyScale esetén: width x height mátrix lesz az img
- A függvény:

```
function readImg()
global orig_img; %(erre a változóra azért van szükség, hogy bármikor a "Reset" gomb
megnyomásánál vissza tudjuk tölteni az eredeti fájlt)
global imgFilename;
global img;
  % ha létezik a filename:
if exist(imgFilename)
   % kép beolvasása
   orig_img = imread(imgFilename);
   % kép eltárolása egy másik változóban
   img = orig_img;
end
```

3. **A plotImg() függvény:**

- **imshow(img);** - A beolvasott képet lehet plottolni, ehhez be kell állítani, hogy az aktuális axes, amire rajzolja a képet, a GUI-n lévő legyen.
- A függvény:

```
function plotImg(handles) //a handles-el a GUI objektumait érjük el
   global orig_img;
   global img;
   % ha már be lett töltve a kép
   if ~isempty(orig_img)
     axes(handles.axes1);
     imshow(orig_img); % kép plottolása
     img = orig_img; % kép eltárolása
   end
```

4. **A greyScale() függvény:**

- function greyScale(handles)

```
   global img;
```

```
% rgb -> greyscale átalakítás
img_gray = rgb2gray(img);
% threshold érték kinyerése
threshold = get(handles.slider1, 'Value');
% ha már be lett olvasva a kép
if ~isempty(img)
  % thresholdot alkalmazva fekete fehérré alakítás
  img_bw = uint8(imbinarize(img_gray,threshold)).*255;
  axes(handles.axes1);
  % ábrázolás
  imshow(img_bw);
end
```

- Az **img_gray = rgb2gray(img);** szürkeárnyalatossá alakítja a beadott rgb képet.
- Az**img = imbinarize(img_gray,threshold);** egy bizonyos thresholdnál (küszöbnél) kisebb értékeket 0-ra, nagyobbakat 1-re állít. Mivel a képeket 0-255 között értelmezzük, ezért minden elemet meg kell szorozni 255-tel és uint8 típussá kell alakítani.

10. Tizedik óra (képmentés, táblázatok, fájlműveletek)

1. **Ábrák mentése:**

   - Figure ablakból a print paranccsal tudunk ábrát elmenteni.
   - Attól függően, hogy milyen formátumba szeretnénk elmenteni az ábrát a következő szintaktikát használhatjuk:
     – LaTeX - print(figure_handler,'-depsc2','-r300',picture_name)
     – png (Word??!) - print(figure_handler,'-dpng','-r300',picture_name)
     – Felületek - print(figure_handler,format,'-zbuffer','-r300',picture_name)
     – Figure handler - figure_handler = figure;
     – Aktuális tengelyek - axes_handler = gca;

2. **Táblázatok:**

   - Olyan adattípus, melyet különböző típusú és/vagy méretű változók, valamint meta adatok rendszerezett tárolására használhatunk.
   - Az oszlopokba sorolt adatok tárolására a legmegfelelőbb, amire példa a vesszővel tagolt fájlok (csv), valamint a táblák (xls,xlsx. . .)
   - Megadható oszlop-, sornév, leírás, oszlop leírás, valamint mértékegység is az oszlopokhoz.
   - **table:** Üres táblázat létrehozása.
     Példa: table;
   - Oszlopnevek megadása a **Varieblanames**-el lehetséges.
     Példa: table(v1,v2,v3,'VariableNames',{'o1','o2','o3'})

3. **Indexelés táblázatokban**
   - () - táblázatot ad vissza
   - {} - homogén adatok esetén tömböt ad vissza
   - oszlopnévvel - vektort ad vissza
4. **Táblázatból tömbbe, tömbből táblázatba:**
   - Több függvénnyel is lehetséges az átjárás a két adattároló között, itt csak felsorolni fogom őket. Bemeneti paraméterként az átalakítandó adattárolót várják.
   - **Táblázatból tömbbe:**
     – array2table
       Példa: array2table(A), ahol A egy tömb
     – cell2table
     – struct2table
   - **Tömbből táblázatba:**
     – table2array
     – table2cell
     – table2struct
5. **Táblázatok műveletei, összefűzése, hozzájuk kapcsolódó függvények:**
   - tulajdonságok: height, width, istable
   - adatokon végzett műveletek: summary, ismember, sortrows, ismissing, varfun, rowfun, standardizeMissing, unique
   - táblázatok műveletei: intersect, union, join, setdiff, setxor, innerjoin, outerjoin
6. **Fájl írás/olvasás táblázatként:**
   - A matlab képes a legtöbb táblázatkezelő fájltípussal dolgozni: .txt, .csv, .dat, .xls, .xlsx, .ods
   - **readtable:** Fájlból táblázat beolvasása.
     Példa: readtable(fileName)
   - A táblázat beolvasása során használhatunk még plusz funkciókat a readtable-vel:
     – **FileType** a fájl típusa lehet ekkor: text - tagolt szöveg, vagy spreadsheet - táblázat
       Példa: readtable(fileName, 'FileType', 'text')
     – **Delimiter** - elválasztó karakter
       Példa: readtable('pelda.csv','Delimiter',';')
     – **Format** - formázó string, ami alapján megy a beolvasás
     – **Range** - beolvasást megadó téglalap alakú cellatartomány
     – **ReadVariableNames** - első sor használata a változók neveinek
       Példa: readtable('pelda.xls','ReadVariableNames',true)

  - **ReadRowNames** - első oszlop használata a sorok neveinek

- **writetable** Táblázat írására használható.
  Példa: writetable(t,fileName)
- Itt a plusz funkciók:
  - **FileType** a fájl típusa lehet ekkor: text - tagolt szöveg, vagy spreadsheet - táblázat
  - **Delimiter** - elválasztó karakter
  - **Sheet** - lap megadása szövegesen, vagy az indexével
  - **Range** - beolvasást megadó téglalap alakú cellatartomány
  - **WriteVariableNames** - első sor használata a változók neveinek