Numerikus módszerek építőmérnököknek Matlab-bal

Dr Laky Piroska

Budapesti Műszaki és Gazdaságtudományi Egyetem
Építőmérnöki Kar

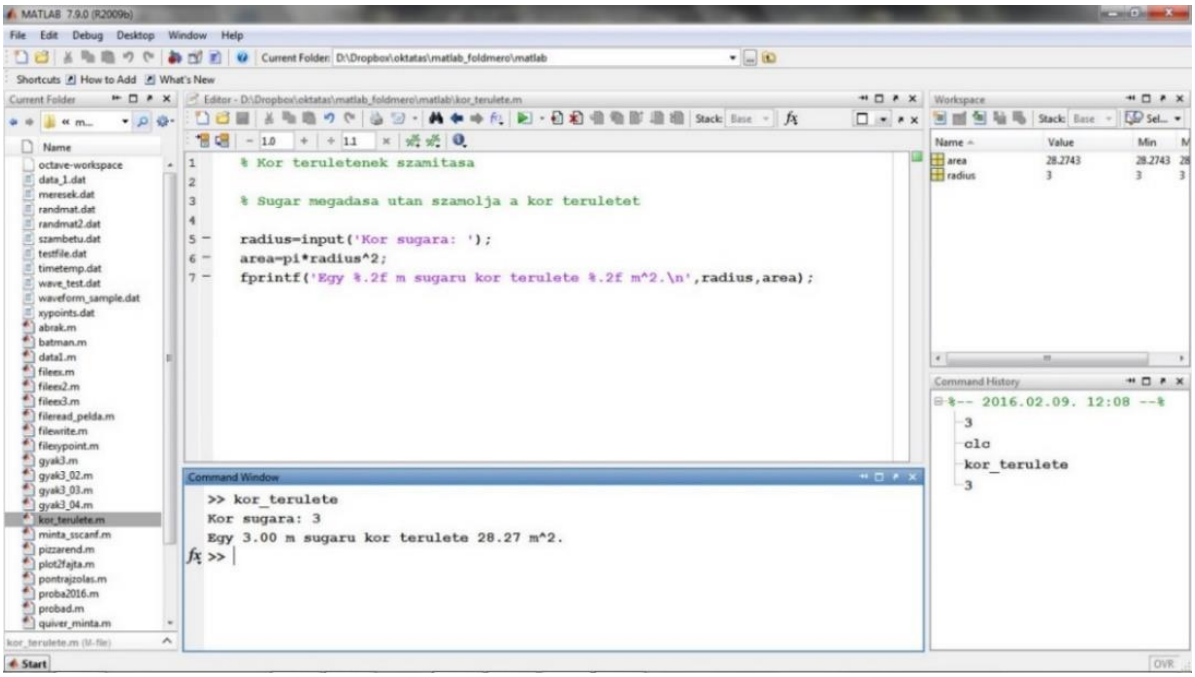
2018

TARTALOMJEGYZÉK

1. MATLAB/OCTAVE ALAPOZÓ[1]

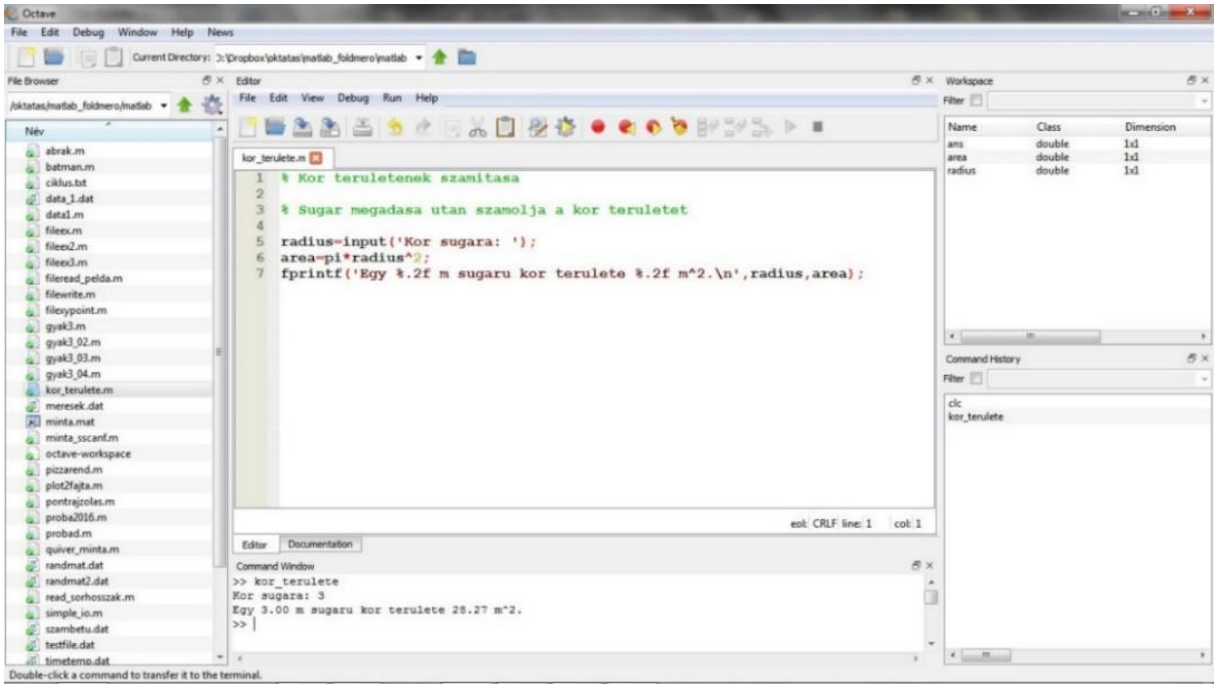
A numerikus módszerek gyakorlatok során Matlab matematikai környezetben mutatjuk meg a mérnöki problémák megoldásához használható különböző numerikus eljárásokat. Otthoni gyakorláshoz használható, 2017 márciusától ingyenesen, a Matlab szoftver a BME egyetemi licenccel (http://www.bme.hu/hirek/20170330/Matematikai_es_muszaki_megoldasok_egyszerubben_es_gyorsabban). Jó alternatív megoldás lehet az Octave matematikai környezet is, ami egy ingyenes, nyílt forráskódú program, ahol lényegében ugyanazokat a parancsokat használhatjuk, mivel az Octave készítői törekednek a Matlab kompatibilitásra. Még a grafikus felület is nagyon hasonló a két programban, tetszés szerint átrendezhető ablakokkal (lásd 1., 2. ábra). Az Octave a https://www.gnu.org/software/octave/ oldalról tölthető le, jelenleg a 4.4.1 változat, ami 2018. augusztus 9-én jött ki.

A Matlab oldalán létre lehet hozni egy MathWorks account-ot (célszerű BME-s email címet használni, ha az Online Matlab-hoz is szeretnénk hozzáférni). A MathWorks account-tal lehetőség van megoldani a **Matlab Onramp**-ben található alap gyakorló feladatokat, ezt mindenkinek ajánlott végigcsinálni (kb. 2-3 óra). További jó, online gyakorlási lehetőségek vannak a **Matlab Cody** oldalán.



1MATLAB GRAFIKUS KÖRNYEZETE

[1] A segédlet készítése során felhasználva: Todd Young and Martin J. Mohlenkamp: Introduction to Numerical Methods and Matlab Programming for Engineers, Department of Mathematics, Ohio University, July 24, 2018, http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/book.pdf



2OCTAVE GARFIKUS KÖRNYEZETE

MATLAB MUNKAKÖRNYEZET, ALAPOK

A grafikus felület fontosabb részei: az éppen aktuális könyvtár, ahová mindent ment a program (**current folder**), a parancssor (**command window**), a munkakörnyezet (**workspace**) az éppen használt változókkal, az eddig futtatott parancsok (**command history**) és a szerkesztő (**editor**). Az adott panel nevére kattintva, lenyomva tartott bal egér gombbal áthúzhatóak a panelek, és a mozgatás során a kékre színezett területre helyezhetőek. Célszerű lehet rögzíteni a szerkesztőt (dock editor), ezt az Editor ablak jobb felső részén lévő nyílra kattintva tehetjük meg.

A későbbiekben amire mindig figyeljünk, hogy a Matlabban használt fájl nevek nem kezdődhetnek számmal és ne legyenek benne ékezetes karakterek, szóközök se! A program mindig azokat a fájlokat, függvényeket tudja éppen használni, amik a beállított aktuális könyvtárban vannak.

SEGÍTSÉG (HELP, DOCUMENTATION)

Nagyon sokat tanulhatunk a dokumentáció helyes használatából is, persze kellő angoltudás függvényében.

Ha csak egyszerűen begépeljük a **help** parancsot, akkor a Matlab kilistázza a különböző beépített függvények kategóriáit. Itt vagy duplán rákattintunk a kategória nevére, vagy begépelhetjük pl. (> jel jelzi a Matlab-ba begépelendő parancsokat, ezt a jelet nem kell begépelni!)

```
> help elfun
```

Ami ki fogja írni az ’Elementary math functions’-t, vagyis az alap matematikai függvényeket, pl trigonometriai, exponenciális, komplex és kerekítő függvények listája.

Ha tudjuk egy adott függvény nevét, akkor használhatjuk következőt:

```
> help 'parancsnév'
```

Ez megadja az adott parancs leírását, használatának módját. Pl.:

```
> help rand
```

Megadja, hogy a **rand** parancs 0-1 közötti egyenletes eloszlású véletlen számot generál, megadja a használatának módját, hogy lehet egy vagy több bemenettel is hívni és megadja a kapcsolódó parancsokat is (pl. **randn**, ami standard normális eloszlású véletlen számokat generál 0 várható értékkel és 1 szórással).

A **doc** parancs a **help**-hez hasonlóan működik, csak egy jóval részletesebb leírás az adott parancsról, sok példával. Pl.

```
> doc randn
```

Próbáljuk ki a **pwd** parancsot! Kérdezzük le **help**-pel ez mit is csinál!

Másik hasznos parancs a **lookfor** utasítás. Ezzel parancs részletekre is rákereshetünk, vagy bármilyen szóra, ami a parancs leírásban szerepel. Próbáljuk ki a következőt:

```
> lookfor rand
```

Ez minden parancsot kilistáz, aminek a nevében, vagy a rövid leírásában szerepel a rand szó. Ha túl sokáig tartana a keresés, akkor megszakíthatjuk a parancsot a **CTRL + c** billentyű kombinációval.

NÉHÁNY HASZNOS PARANCS

`clc`	– kitörli a command window ablak tartalmát
`clear`	– kitörli a változókat (lásd workspace)
`close`	– bezárja az aktuális ábrát, vagy az összeset (close all)
`CTRL+C`	– félbeszakítja az adott parancsot (kilépés pl. végtelen ciklusból)
%	– megjegyzés (a program figyelmen kívül hagyja ami ez után van a sorban)
;	– parancs végén a ; hatására nem jelenik meg az eredmény

'TAB' GOMB ÉS A NYILAK HASZNÁLATA A PARANCSSORBAN

Nagyon hasznos a 'tab' használata. Ha nem tudjuk pontosan egy adott parancs nevét, csak az elejét, és elkezdjük begépelni a parancssorba pl, hogy pref, majd utána nyomunk egy **tab**-t ha csak egyféle pref kezdetű parancs van, akkor kiegészíti, ha több, akkor megadja a lehetséges parancsokat. Itt több parancs is van ezzel a kezdettel pl. **prefdir** (annak a könyvtárnak a neve, ahol a beállítások, history stb. található) vagy a **preferences**, ami megnyitja a beállítások ablakot.

Szintén nagyon hasznos a nyilak használata a parancssorban, amivel korábbi parancsokat lehet újra előhozni, lefuttatni, módosítani. A korábbi parancsokat újra le lehet futtatni a **command history**-t használva is, dupla kattintással az adott parancson.

```
> %% Értékadás, változótípusok
> % Egyszerű értékadás (0.01 háromféleképpen)
> a = 0.01
> b = 1e-2
> c = 1d-2;
> a+c
> clear a % kitörli az 'a' változót
> clear % vagy 'clear all' kitörli az összes változó értekét
> pi % beépített érték (3.14)
> e = exp(1) % e^1 = e = 2.71
> b = e^-10 % hatványozás
```

Néhány megjelenítési, formázási lehetőség:

```
> format long % több tizedes jegy megjelenítése
> e, b
> format short % rövidebb megjelenítés
> e, b
```

A Matlab alap objektumai a mátrixok. A vektorok speciális mátrixok, pl. 1xn-es sorvektorok, vagy mx1-es oszlopvektorok. Mátrix/vektor definiálásához szögletes zárójelet használunk. Az elemeket egy soron belül vesszővel, vagy szóközzel választjuk el, sorok között az elválasztó a pontosvessző.

```
> z = [1 3 45 33 78] % sorvektor
> z = [1,3,45,33,78] % sorvektor másképp megadva
> t = [2; 4; 22; 66; 21] % oszlopvektor
> M = [1,2,3; 4,5,6] % 2x3-as méretű mátrix/tömb
```

Le lehet kérdezni egy vektor tetszőleges elemét, kerek zárójelbe téve az elem sorszámát:

```
> t(2) % eredménye: 4
> M(2,3) % eredménye: 6
> z(end) % z utolsó eleme: 78
```

Vagy felül lehet írni bármelyik elem értékét:

```
> t(2)=47
> p = [] % üres vektor
> z(3)=[]; % kitörli a 3. elemet, utána z = 1 3 33 78
```

Le lehet kérdezni egy részét a vektornak, mátrixnak:

```
> t(2:4) % eredménye az előző parancs után: 47 22 66
> t(1:29 % elgépelés esetén hibaüzenet
  t(39
      ?
 Error: Expression or statement is incorrect--possibly unbalanced (, {, or
[.
> t(1:29)
Index exceeds matrix dimensions.

> M(1:2,2:3) % eredménye: [2,3; 5,6]
```

Vektor, mátrix transzponáltja:

```
> tt = t' % t transzponáltja, sorvektor
> Mt = M' % eredménye: [1,4; 2,5; 3,6]
```

Vannak hasznos parancsok, amelyekkel egyszerűen lehet vektorokat előállítani:

```
> x1 = 1:10 % sorvektor 1-10-ig egész számok
> x2 = 1:0.3:10 % sorvektor 1-tol 10-ig, 0.3 osztásközzel
> x3= (1:0.3:10)' % oszlopvektor 1-tol 10-ig, 0.3 osztásközzel
> x4 = linspace(1,10,4) % 1 és 10 között 4 pontot vesz fel
```

Könnyű összefűzni vízszintesen, függőlegesen azonos sor/oszlop számú vektort/mátrixot.

```
> X = rand(2,3) % 2x3-as [0,1] közti véletlen számokból álló mátrix
> Y = ones(2,4) % 2x4-es egyesekből álló mátrix
> Z = eye(3) % 3x3-as egységmátrix
> W = zeros(2,4) % 2x4-es nullákból álló mátrix
> XY = [X, Y] % 2x7 elemű mátrix, vízszintesen összefűzve X és Y
> XZ = [X; Z] % 5x3-as mátrix, függőlegesen összefűzve X, Z
> XY2 = [X; Y] % hibaüzenet
Error using vertcat
Dimensions of matrices being concatenated are not consistent.
> XZ2 = [X,Z] % hibaüzenet
Error using horzcat
Dimensions of matrices being concatenated are not consistent.
```

Sorok oszlopok leválogatása

```
> XY(1,:) % XY első sora (: az adott sorban az összes elem)
> XY(end,:) % XY utolsó sora (: az adott sorban az összes elem)
> XY(:,1) % XY első oszlopa (: az adott oszlopban az összes elem)
> XY(:,end-1) % XZ utolsó előtti oszlopa (: az adott oszlop összes
  eleme)
```

Szövegek, mint karakterekből álló vektorok

```
> s = 'p' % szöveges/karakter (string) típusú változó 1x1 méretű
> me = 'Műszaki Egyetem' % string típusu változo 1x15 méretű
> bme = ['Budapesti ',me] % Budapest Műszaki Egyetem
> bme(13:17)
```

SCRIPT ÍRÁSA

Eddig parancssorból dolgoztunk, de egy bonyolultabb számítást, programot már nehéz parancssorban megírni, szükség esetén javítani. Célszerűbb lehet egy fájlba összegyűjteni a használt parancsokat, ez a script fájl. Itt is használhatunk minden korábbi Matlab függvényt, de egyszerűbb a javítás, futtatás. A Matlab alapértelmezett fájl típusa a *.m fájl, ez egy egyszerű szövegfájl, amit bármilyen szövegszerkesztővel szerkeszthetünk. Ezen kívül az újabb Matlab verziókban már használhatjuk a livescript fájltípust is (*.mlx), ez utóbbiban vegyesen használhatunk Matlab utasításokat és formázott szövegeket, képeket, illetve láthatjuk rögtön az eredményeket is. Ez viszont a Matlab saját formátuma, amit csak Matlab programon belül tudunk megnyitni.

A script fájlokat futtathatjuk a nevük begépelésével, de egyszerűbb az **F5** megnyomásával, ez rögtön menti és futtatja a programot. Figyeljünk, hogy a fájlnév ne kezdődjön számmal és ne legyenek benne szóközök, ékezetek! A fájlnevek csak betűvel kezdődhetnek, és csak az angol abc betűi, illetve számok és alulvonás szerepelhet bennük.

Amennyiben nem akarjuk az egész programot lefuttatni betehetünk egy **return** parancsot valahová, és akkor csak addig fog lefutni a program. Illetve az **F9**

lenyomásával csak a kijelölt rész fog lefutni. Másik megoldás, ha szekciókra osztjuk a fájlt dupla % jelek használatával (%%) és utána egy szekció cím megadásával. Egy bizonyos szekcióban lévő parancsokat a **CTRL+Enter** billentyűk megnyomásával tudunk futtatni. Kezdjünk egy új script fájlt a bal felső sarokban a plusz jelre kattintva (new), és ezzel megnyílik az Editorban egy üres lap. Mentsük el az aktuális könyvtárunkba gyak1.m fájl néven! A továbbiakban ebbe fogunk dolgozni.

EGYSZERŰ PLOTTOLÁS

Nézzük az alábbi táblázat adatait egy betonacél feszültség-fajlagos alakváltozás (σ-ε) diagramjából:

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

1. TÁBLÁZAT BETONACÉL FESZÜLTSÉG-FAJLAGOS ALAKVÁLTOZÁS DIAGRAMJA

Írjuk be a gyak1.m script fájlba:

```
> %% Betonacel alakváltozása
> x = [0,0.2,2,20,25] % kiírja a képernyőre a tartalmát
> y = [0,300,285,450,350]; % nem írja ki a képernyőre a tartalmát
```

Futtassuk vagy az **F5** paranccsal az egész fájlt, vagy egy kijelölt részt az **F9**-cel, vagy **CTRL+Enter**-rel a szakaszt!

Ha beírjuk a script fájlba, parancssorba egy létező változó nevét, akkor kiírja az aktuális tartalmát, még akkor is, ha az előző parancsnál az y után ; szerepelt, ezért ott ugyan nem írta ki a képernyőre a változó értékét, de a workspace-be elmentette!

```
> x, y
```

Vektor formátumban lévő adatokat a plot paranccsal rajzolhatunk ki:

```
> plot(x,y)
```

Ez egy vonallal össze fogja kötni a pontokat. Ha a pontokat szimbólumokkal szeretnénk jelölni, próbáljuk ki a következőket:

```
> plot(x,y,'x')
> plot(x,y,'o-')
> plot(x,y,'r*-')
```

Hasznos paraméterek:

Line Style	Description
-	Solid line (default)
--	Dashed line
:	Dotted line
-.	Dash-dot line

Marker	Description
o	Circle
+	Plus sign
*	Asterisk
.	Point
x	Cross
s	Square
d	Diamond
^	Upward-pointing triangle
v	Downward-pointing triangle
>	Right-pointing triangle
<	Left-pointing triangle
p	Pentagram
h	Hexagram

Long Name	Short Name	RGB Triplet
'yellow'	'y'	[1 1 0]
'magenta'	'm'	[1 0 1]
'cyan'	'c'	[0 1 1]
'red'	'r'	[1 0 0]
'green'	'g'	[0 1 0]
'blue'	'b'	[0 0 1]
'white'	'w'	[1 1 1]
'black'	'k'	[0 0 0]

LineWidth	vonalvastagság
MarkerEdgeColor	pont szimbólumok határoló vonalának a színe
MarkerFaceColor	pont szimbólumok kitöltése
MarkerSize	pont szimbólumok mérete

Használata, pl.:

```
> plot(x,y,'--gs','LineWidth',2,'MarkerSize',10,...
>       'MarkerEdgeColor','b','MarkerFaceColor',[0.5,0.5,0.5])
```

Elnevezhetjük a tengelyeket, vagy jelmagyarázatot, címet is fűzhetünk hozzá:

```
> xlabel('Fajlagos alakváltozás')
> ylabel('Feszültség')
> title('Betonacél feszültség-fajlagos alakváltozás diagramja')
```

Vagy bezárhatjuk az ábrát:

```
> close % ábra bezárása
```

FÜGGVÉNYEK

Nagyon sok matematikai és egyéb beépített függvény van, ezek megismeréséhez célszerű a **help**-et, dokumentációt böngészni. Nézzük meg néhány függvényt az alap matematikai függvények közül (Elementary math functions) – ld. **help elfun**

A függvények változói mindig kerek zárójelben vannak. A trigonometrikus függvényeknél az alapértelmezett szögegység a radián!

```
> sin(pi) % értéke 0 a számábrázolás pontosságán belül
> cos(pi) % -1
> tan(pi) % végtelen helyett egy nagy szám
> log(100) % természetes alapú logaritmus
> log10(100) % 10-es alapú logaritmus
> 3^4 % értéke: 81
> sqrt(81) % 9
> abs(-6) % 6
> exp(0)
```

Az **exp(0)** az e^0, értéke természetesen 1.

A beépített függvények nem csak számokon, hanem vektorokon is működnek.

```
> x = linspace(0,2*pi,40)
> y = sin(x)
> plot(x,y)
```

További beállítások lásd **help plot**!

FELHASZNÁLÓ ÁLTAL DEFINIÁLT EGYSOROS FÜGGVÉNYEK

Többféleképp lehet Matlab-ban saját függvényeket megadni, az egyszerűbb esetekben leginkább az egysoros függvényeket használjuk (ezt az angol nyelven 'anonymous function'-nek nevezik, ami nincs elmentve külön programként, csak egy változóhoz van hozzárendelve), pl definiáljuk a cos(2x) függvényt!

```
> f = @(x) cos(2*x)
```

Itt először egy @ szimbólum után meg kell adni a függvény független változóit, majd utána jön a formula. Rajzoljuk fel az előbb felrajzolt szinusz függvény mellé ezt is, most nem előre megadott pontpárokat használva, hanem szimbolikusan az **fplot** vagy **ezplot** paranccsal! (Megj.: Octave-ban és régebbi Matlab verzióba az ezplot parancs használható, újabb Matlab-ban az fplot).

```
> hold on
> fplot(f,[0,2*pi])
```

A **hold on** parancs nélkül, ha egy új rajzolási parancsot adok ki, akkor letörli az előző ábrát és úgy rajzolja fel az újat, **hold on** esetén megtartja a régit, és mellé rajzolja az újat. A **hold off** paranccsal visszaállítható az alapértelmezett mód. Az **fplot** esetén meg lehet adni az intervallumot, ha nem az alapértelmezett tartományon szeretnénk a függvényt megjeleníteni.

Az ábra törlését, grafikus ablak bezárását a következő parancsokkal tehetjük meg:

```
> clf % Clear current figure
> close % Close figure
```

Állítsuk elő saját függvényt használva 1-10-ig a számok négyzetét!

```
> f1 = @(x) x^2
```

Ellenőrizzük le egy adott x értékére:

```
> f1(3) % értéke: 9
```

Állítsuk elő az f függvényt használva 1-10-ig a számok négyzetét!

```
> x = 1:10;
> y = f1(x)
```

Erre hibaüzenetet kapunk:

```
Error using  ^
One argument must be a square matrix and the other must be a scalar. Use
POWER (.^) for elementwise power.
Error in gyak1>@(x)x^2
Error in gyak1 (line 100)
y = f(x)
```

Miért? Azért mert az x változónk egy sorvektor, négyzetre emeléskor pedig két sorvektort próbálunk összeszorozni, ami matematikailag helytelen. Ha nem vektor/mátrix műveletet szeretnénk végrehajtani, hanem azt szeretnénk, hogy elemenként hajtódjon végre a művelet, akkor egy pontot kell tenni a műveleti jel elé.

```
> f1 = @(x) x.^2
> y = f1(x) % 1  4   9  16  25  36  49  64  81  100
```

Mivel vektorok/mátrixok esetén az összeadás, kivonás, skalárral való osztás/szorzás eleve elemenként történik, ezért ezekben az esetekben nem kell pontot tenni a műveleti jel elé, csak a szorzás, osztás és hatványozás esetén: .*,./,.^.

A Matlabnak az egyik erőssége a vektorokkal, mátrixokkal való műveletek végzése, így sok esetben elkerülhetjük a lassabb ciklus műveletek alkalmazását.

FÜGGVÉNYEK KÜLÖN FÁJLBAN

A Matlab alapértelmezett fájltípusa a *.m kiterjesztésű szöveges fájl. Ennek két fő típusa van, script fájl és függvény (function). Az előbbit már használtuk az eddigi munkánk során, nézzük meg miben különbözik ettől a függvények írása!

A függvények külön fájlba történő megírásának egyik előnye, hogy nem csak abból a script fájlból használhatjuk, ahol épp dolgozunk, hanem bármely fájlból meghívhatóak, így ha van olyan feladat, ami gyakran ismétlődik, akkor azt célszerű egy külön függvényben megírni. Az egysoros függvényekkel szemben sokkal bonyolultabb számítások, műveletek elvégzésére használhatjuk őket, könnyebben paraméterezhetjük, hogy hány ki és bemenete legyen, magyarázatokat fűzhetünk hozzá.

Írjuk meg külön függvény fájlba az előbb megadott négyzet függvényt! Kattintsunk a bal felső sarokban a plusz jelre (new), és megnyílik az Editorban egy üres lap. írjuk be a következőket, majd mentsük el negyzetfv.m néven az aktuális könyvtárba. Fontos: a függvény neve (ami vastaggal van írva a következő kódban) meg kell egyezzen a fájl nevével, különben nem lehet meghívni!

```
> function y = negyzetfv(x)
> % Kiszámolja a bemenet négyzetét
>     y = x.^2;
> end
```

A függvények néhány fontos tulajdonsága:

- function szóval kezdődik
- Van legalább egy-egy kimenet és bemenet
- A kimenet, a függvény neve és a bemenet az első sorban található, és a függvény neve meg kell egyezzen az *.m fájl nevével
- A függvény belsejében valahol értéket kell adjunk a kimenetnek
- A függvény belső változói lokális változók, nem fognak megjelenni a workspace-ben, és a függvény sem fér hozzá a workspace-ben lévő változókhoz, csak ahhoz, amit megadtunk a bemenetnél.
- A függvényt nem lehet futtatni, csak egy másik fájlból, vagy parancssorból meghívni! A meghíváshoz a függvénynek az aktuális könyvtárban kell lennie (vagy egy olyan könyvtárban, ami benne van az elérési útban (path).
- A függvény első sora után megadott kommentek tartalmát listázza ki a help parancs az adott függvényre!

Hívjuk meg a megírt függvényt az x vektorunkra! Ehhez váltsunk vissza a gyak1.m script fájlra!

```
> negyzetfv(11) % 121
> negyzetfv(x) % 1   4   9   16   25   36   49   64   81   100
> help negyzetfv %  Kiszámolja a bemenet négyzetét
```

Egy függvénynek lehet több bemenete is, pl módosítsuk az előző függvényünket az alábbi módon, és mentsük el hatvany.m néven!

```
> function y = hatvany(x,p)
>     y = x.^p
> end
```

Egy függvénynek lehet több kimenete is egy vektorban összegyűjtve (hatvanyok.m):

```
> function [x2 x3 x4] = hatvanyok(x)
>     x2 = x.^2;
>     x3 = x.^3;
>     x4 = x.^4;
> end
```

Hívjuk meg a fenti függvényeket is a script fájlunkból!

```
> hatvany(x,3) % 1   8   27   64   125   216   343   512   729   1000
> [a b c] = hatvanyok(x)
> % a = 1 4   9 16 25 36 49 64 81 100
> % b = 1 8 27 64 125 216 343 512 729 1000
> % c = 1 16 81 256 625 1296 2401 4096 6561 10000
```

Ábrázoljuk az eredményeket egy új 2-es számú ábrán a **figure** parancs használatával! A **plot** parancsnál egymás után több összetartozó értéket is felsorolhatunk!

```
> figure(2)
> plot(x,a,x,b,x,c)
```

Mi is adhatunk egyéni színeket hozzá, illetve jelmagyarázatot is fűzhetünk hozzá olyan sorrendben megadva a jelmagyarázat szövegét, ahogy felrajzoltuk az ábrákat:

```
> plot(x,a,'black',x,b,'blue',x,c,'green')
> plot(x,a,'k',x,b,'b',x,c,'g')
> legend('négyzet','köb','x^4','Location','North')
> legend('négyzet','köb','x^4','Location','Best')
```

PROGRAM KOMMENTEK, 'HELP' ÍRÁSA SAJÁT FÜGGVÉNYEKHEZ

A több sorból álló programok fontos részei a kommentek. Egyrészt mások is megértik a programkódunkat, másrészt mi is emlékezni fogunk rá, ha később újra használni akarjuk. Célszerű nem csak a program elején, hanem minden új szekcióhoz is kommenteket írni. A Matlab-ban % jel után írhatunk kommenteket. Egy függvény esetében a kommentben célszerű megadni, hogy mi a függvény célja, milyen bemeneti és kimeneti értékek szerepelnek benne. Függvény esetében az első sor után megadott kommentek fognak megjelenni a help utasítás hatására segítségként.

MATLAB HIBAÜZENETEK

A programok írása során számos hibaüzenettel találkozunk, eddig is láttunk már néhányat. Fontos, hogy tudjuk ezeket értelmezni, ez segíthet kijavítani a hibáinkat!

Nézzünk egy példát elírásra a clear all helyett!

```
> cler all
Undefined function or variable 'cler'.
Did you mean:
>> clear all
```

A Matlab érzékeny a kis nagy betűk változására is:

```
> X = 3/4; x
Undefined function or variable 'x'.
```

Nézzünk példát egy szintaktikailag hibás Matlab utasításra:

```
> 1 X
1 X
  ↑
Error: Unexpected MATLAB expression.
```

Túl sok bemenő paraméter:

```
> sin(pi,3)
Error using sin
Too many input arguments.
```

Nem egyezik a mátrixban lévő sor/oszlop elemeinek száma:

```
> M = [1 2;3]
Dimensions of matrices being concatenated are not consistent.
> [3, 4, 5] * [1; 2; 3; 4]
Error using  *
Inner matrix dimensions must agree.
> a = 1:5, b = 1:3
> [a;b]
Error using vertcat
Dimensions of matrices being concatenated are not consistent.
```

Könnyen elgépelhető hiba lehet, hogy zárójel helyett 8 v 9 kerül beírásra:

```
   >  sin(pi9
sin(pi9
        ↑
Error: Expression or statement is incorrect--possibly unbalanced (, {, or [.
```

Lemarad egy zárójel:

```
   >  abs(sin(rand(2))
abs(sin(rand(2))
                ↑
Error: Expression or statement is incorrect--possibly unbalanced (, {, or [.
```

Elemenkénti műveletet akarunk végezni vektoron, de lemarad a pont:

```
   >  v =1:4;
   >  1/v
Error using  /
Matrix dimensions must agree.
```

A legrosszabb, amikor nincs hibaüzenet, az eredmény mégis hibás. Példa: számítsuk ki $\frac{1}{2\pi}$ értékét a következő utasítással! Miért hibás az eredmény?

```
   >  1 / 2*pi %  1.5708
```

KIEGÉSZÍTÉS OCTAVE HASZNÁLATÁHOZ

Ha valaki az Octave használata mellett dönt otthoni gyakorláshoz, akkor az Octave a https://www.gnu.org/software/octave/ oldalról tölthető le, jelenleg a 4.2.1 változat (2017. február 24 óta). Az Octave előnye az egyetemi Matlab-bal szemben, hogy mivel egy nyílt forráskódú program, nemcsak tanuláshoz, oktatási célra használható, hanem munkához is. Sok kiegészítő csomag is van hozzá, lásd https://octave.sourceforge.io/ illetve https://octave.sourceforge.io/packages.php , ezek egy jó része telepítve is van, csak be kell tölteni használatkor. Le lehet kérdezni, hogy mi van telepítve a **pkg list** paranccsal). Célszerű ismerni egy parancsot, ami eltér a Matlab-tól, hogy ha az Octave-ban azt szeretnénk, hogy bármilyen művelet eredménye folyamatosam jelenjen meg a Command Window-ban, és ne oldalanként tördelve, akkor a következő paranccsal tehetjük meg:

```
   >  page_screen_output(0)
```

SYMBOLIC CSOMAG TELEPÍTÉSE

A numerikus módszerek gyakorlatok során szimbolikus számításokat is végezni fogunk, ehhez szükséges az Octave-ban egy extra symbolic csomagot telepíteni (ez a matlab-ban is egy külön toolbox). Ez a csomag python-sympy alapra épül, ezért külön kell telepíteni. Windows alatti telepítés (a következő link alapján: https://github.com/cbm755/octsympy/wiki/Notes-on-Windows-installation)

1. Töltsük le a symbolic-win-py-bundle-x.y.z.zip fájlt következő oldalról : https://github.com/cbm755/octsympy/releases (itt x.y.z a verziószám, pl. **symbolic-win-py-bundle-2.5.0.zip**) Ez tartalmazza a Python interpretert és a SymPy-t is.

2. Indítsuk el az Octave-t, állítsuk be aktuális könyvtárnak, ahová mentettük az előbbi fájlt.
3. Gépeljük be a következő parancsot (x.y.z helyett a megfelelő verziószám):

```
> pkg install symbolic-win-py-bundle-x.y.z.zip
```

4. Töltsük be a telepített csomagot:

```
> pkg load symbolic
```

Ellenőrizzük, hogy működik-e pl.

```
> syms x
> f = sin(cos(x));
> diff (f)
```

Eredmény ⇒ `-sin(x)·cos(cos(x))`

A symbolic csomag függvényei részletesen:
https://octave.sourceforge.io/symbolic/overview.html

SEGÍTSÉG (HELP, DOCUMENTATION)

Nagyon sokat tanulhatunk a dokumentáció használatából is (lásd a documentation fül, vagy az alábbi weblap), persze kellő angoltudás függvényében. Nyugodtan lehet nem csak az Octave saját dokumentációját nézni, hanem a Matlabét is az interneten, ott több, sokszor részletesebb példát is láthatunk az adott parancs használatára.

Octave help-je: https://www.gnu.org/software/octave/doc/interpreter/

Matlab help-je: http://www.mathworks.com/help/matlab/

Sok hasznos segítség, letölthető matlab fájl található a matlab centralon is: http://www.mathworks.com/matlabcentral/

HASZNÁLT MATLAB FÜGGVÉNYEK

help	-	matlab helpjének kategóriái, vagy segítség megadott témakörhöz, függvényhez a Command Window-ban
rand	-	Véletlen számok 0-1 között egyenletes eloszlásban
randn	-	Véletlen számok sztenderd normális eloszlásban, 0 várható értékkel és 1 szórással
doc	-	részletes dokumentáció adott függvényhez, parancshoz új ablakban
lookfor	-	keresés a help-ben adott szóra, szórészletre
clc	-	kitörli a command window ablak tartalmát
clear, clear all	-	kitörli a megadott változókat, vagy az összes változót
close, close all	-	bezárja az aktuális ábrát, vagy az összeset
CTRL+C	-	félbeszakítja az adott parancsot (kilépés pl. végtelen ciklusból)
%	-	megjegyzés (a program figyelmen kívül hagyja ami ez után van a sorban)

;	-	parancs végén a ; hatására nem jelenik meg az eredmény a Command Window-ban
Tab gomb	-	elkezdett parancsot kiegészíti
preferences	-	megnyitja a beállítások ablakot
prefdir	-	annak a könyvtárnak a neve, ahol a beállítások, history stb. található
↑↓ gombok	-	korábbi parancsokat vissza lehet hozni a Command Window-ba
pi	-	3.14…. (pi szám)
exp(1), exp(n)	-	e^1 =2.71…, e^n
^	-	hatványozás
format long	-	több tizedes jegy megjelenítése
format short	-	rövidebb megjelenítés
[1, 2, 3; 4, 5, 6]	-	vektor, mátrix megadása
'	-	vektor, mátrix transzponáltja
[A,B] vagy [A B]	-	mátrixok összefűzése egymás mellé (sorok száma egyenlő)
[A;B]	-	mátrixok összefűzése egymás alá (oszlopok száma egyenlő)
A(1,:)	-	mátrix első sora
A(:,1), A(:,end)	-	mátrix első/utolsó oszlopa
linspace(x1,x2,n)	-	[x1,x2] intervallumban n pont felvétele egyenletesen
ones	-	egyesekből álló mátrix
zeros	-	nullákból álló mátrix
eye	-	egységmátrix
figure	-	új ábra nyitása
plot	-	összetartozó pontpárok felrajzolása
xlabel, ylabel	-	x,y tengely feliratozása
title	-	ábra címe
sin, cos, tan	-	szögfüggvények (alapértelmezett mértékegység a radián!)
log, log10	-	természetes alapú logaritmus, 10-es alapú logaritmus
sqrt	-	négyzetgyök
abs	-	abszolút érték
hold on, hold off	-	felülírja, vagy ne írja felül a meglévő ábrát az új ábrával
fplot, ezplot	-	függvények felrajzolása
.* ./ .^	-	elemenkénti szorzás, osztás, hatványozás vektoroknál
clf	-	ábra törlése (nem zárja be az ablakot)
legend	-	jelmagyarázat

2. MATLAB ELÁGAZÁSOK, CIKLUSOK, FÁJLMŰVELETEK

ELÁGAZÁSOK, CIKLUSOK

KÉTIRÁNYÚ FELTÉTELES ELÁGAZÁS - IF, ELSEIF, ELSE

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

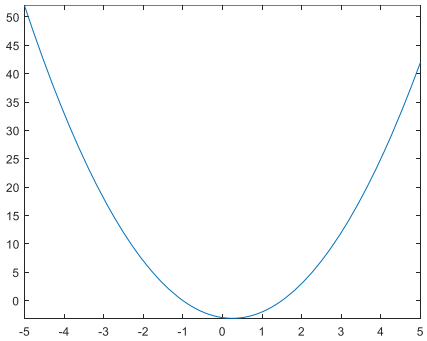
$$2\,x^2 - x - 3 = 0$$

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

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

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
```

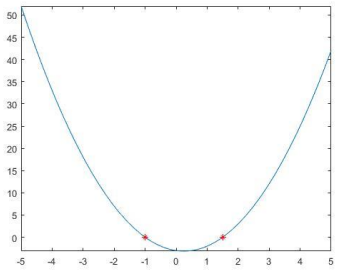
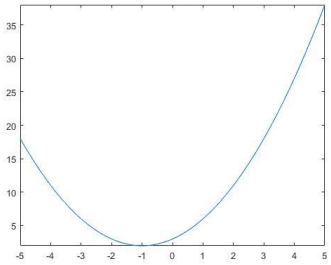
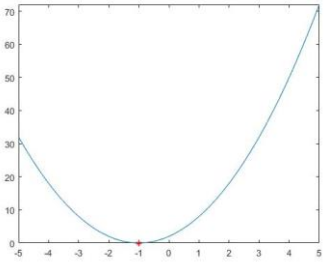
A fenti függvényben az **if**, **elseif**, **else, end** típusú kétirányú elágazást használtuk. Az alap **if** utasítás szerkezete: **if** feltétel; mit tegyen ha igaz a feltétel (lehet egy vagy több sorban is), **end**; Az end előtt lehetnek elágazások benne az **elseif** (egyébként, ha…), illetve az **else** (egyébként) használatával. 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!

Megj: Octave használata esetében adjuk meg, hogy az eredményeket ne laponként hanem folyamatosan írja ki a képernyőre - **page_screen_output(0)**.

```
> page_screen_output(0) % Csak Octave esetében!
> %% 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. 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

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

Nézzük a következő 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?')
> ev = 33; ho = 3; nap = 3;
> ev2 = ev + ho/12 + nap/365;
> fprintf('A kapitány 33 éves')    % nem rak be sortörést a végére
> fprintf('A kapitány 33 éves\r\n')% \r\n - sortörés[2]
> sprintf('A kapitány 33 éves')   % szöveges változóba teszi az
  eredményt (ans)
> sprintf('A kapitány %d éves, %d hónapos és %d napos',ev,ho,nap) %  'A
  kapitány 33 éves, 3 hónapos és 3 napos'
> sprintf('A kapitány %f éves', ev2) % 'A kapitány 33.258219 éves'
> sprintf('A kapitány %.2f éves', ev2) % 'A kapitány 33.26 éves'
> sprintf('A kapitány %8.2f éves', ev2) % 'A kapitány    33.26 éves'
> sprintf('A kapitány %08.2f éves', ev2) % 'A kapitány 00033.26 éves'
> sprintf('A kapitány %+6.2f éves', ev2) % 'A kapitány +33.26 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

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

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!

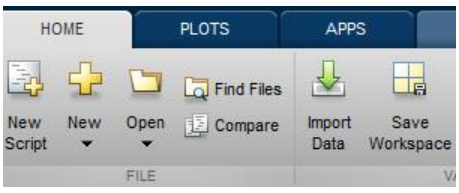
```
> a = 123.123, b = -123.123
> fpm(a)  % '123-07-23'
> fpm(b)  % '-123-07-23'
```

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

FÁJLMŰVELETEK

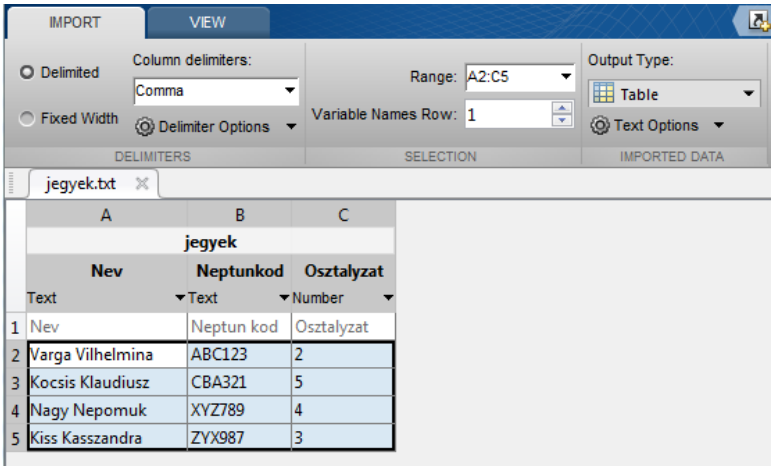
A mérnöki munkák során gyakran előfordul, hogy egy műszeres mérés eredményeit kell feldolgozni és ezeket például egy szöveges fájlban kapjuk meg valamilyen formátumban. Ha Matlabbal szeretnénk a méréseket feldolgozni, akkor be kell tudjuk olvasni ezeket az adatokat. A beolvasás a formátum függvényében 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

2. 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)!

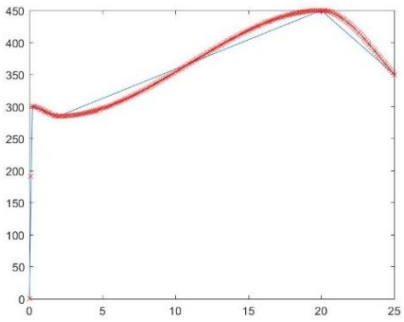
```
> x = adat(:,1); % első oszlop
> y = adat(:,2); % második oszlop
> plot(x,y);
> xlabel('\epsilon');ylabel('\sigma');
```

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.




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

3. ÁBRA BETONACÉL FAJLAGOS ALAKVÁLTOZÁS-FESZÜLTSÉG ÁBRÁJA

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)[3]

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!

[3] Otthoni átnézésre

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 (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;
```

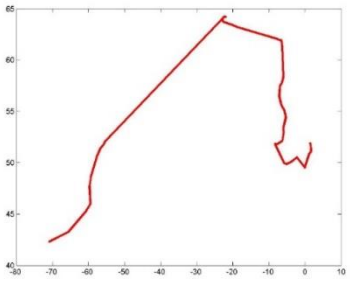
Így lehet például egy sorból kinyerni egy bonyolultabb szerkezet esetén a minket érdeklő információt. Persze rengeteg egyéb kiíró, beolvasó művelet létezik még matlabban (input/output functions), ezeket részletesebben megnézhetjük a **help iofun** parancs kiadásával.

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!

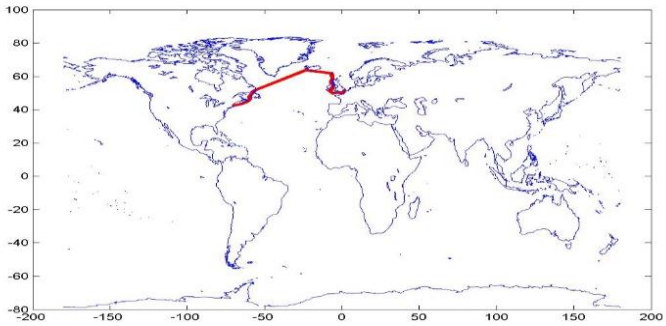
```
> figure(2)
> plot(long, lat,'r','LineWidth',3)
```



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?



A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

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

3. SZÁMÍTÁSOK HIBÁI

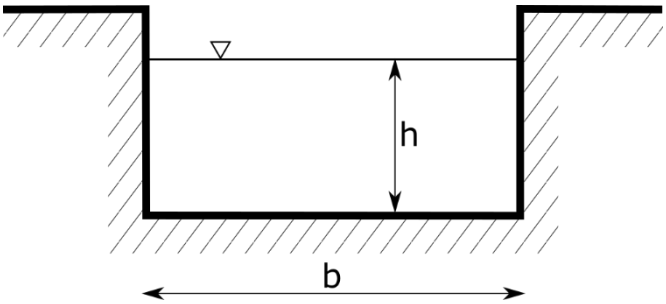
BEVEZETÉS A NUMERIKUS MÓDSZEREKBE

Bizonyos feladatokat, problémákat a hagyományos analitikus matematikai módszerekkel nem, vagy csak nagyon nehezen lehet megoldani. Ezekben az esetekben használhatóak a numerikus módszerek. Az analitikus megoldások egzakt megoldások, ahol a feladatban szereplő változók zárt képletekkel kifejezhetőek. Ilyen például egy másodfokú egyenlet megoldó képlete. A numerikus megoldás csak egy megközelítő numerikus értéke a tényleges megoldásnak. Bár a numerikus megoldás csak egy közelítés, az értéke nagyon pontos is lehet. A legtöbb numerikus módszer iteratív eljárás, ahol fokozatosan közelítjük a megoldást, amíg a kívánt pontosságot el nem érjük. Nézzünk egy építőmérnöki példát, olyan esetre, ami analitikusan nem megoldható!

Hidraulikában gyakori feladat a csatorna méretezés. A nyílt felszínű csatorna alakja, anyaga, esése, szélessége és a vízmagasság függvényében levezethető az elszállított vízhozam nagysága. Nézzük például egy szabadfelszínű, téglalap keresztmetszetű csatorna vízhozamának a képletét!

$$Q = \frac{\sqrt{S}}{n} \cdot \frac{(b \cdot h)^{\frac{5}{3}}}{(b + 2 \cdot h)^{\frac{2}{3}}}$$

ahol Q - vízhozam, n - Manning-féle érdességi együttható, S - esés, b - csatorna szélessége, h - vízmagasság.



Ez egy egzakt képlet Q-ra, ha azonban arra vagyunk kíváncsiak, hogy a mértékadó vízhozam mekkora vízmélységgel képes lefolyni, akkor azt már nem tudjuk explicit módon kifejezni. Korábban erre a feladatra különböző grafikus vagy táblázatos formájú méretezési segédleteket használtak. Ugyan analitikus módon most sem tudjuk előállítani a megoldást, viszont a számítógépek megjelenése óta, numerikus módszereket használva előre megadott pontosságú közelítő értéket tudunk adni a megoldásra. Ez azt jelenti, hogy ha visszahelyettesítjük a megoldást h-ra, akkor nem kapjuk ugyan vissza pontosan a vízhozam (Q) értékét, de nagyon közel leszünk hozzá.

Évszázadok óta vannak kidolgozott numerikus technikák az ilyen feladatok megoldására, de ezeknek az alkalmazása a mai számítógépek megjelenése előtt nagyon bonyolult volt. A kézzel, vagy mechanikus számológéppel végzett számítások nagyon időigényesek és könnyen elszámolhatóak voltak. Ezek a technikák csak a számítástechnika megjelenésével terjedhettek el, mivel ezekkel már nem okoz problémát a sok ismétlődő, bonyolult számítás elvégzése rövid idő alatt.

Egy mérnöki probléma megoldása során először definiálni kell a feladatot, változókat, feltételeket (határértékek, kezdőértékek). Utána fel kell írni a fizikai modellt a problémához, lehet ez a vízhozam képlete, tömegvonzás, Newton törvényei stb. El kell dönteni, hogy megoldható-e a feladat analitikusan vagy csak numerikusan. Lehet-e esetleg elfogadható egyszerűsítéseket (pl. linearizálás) tenni az analitikus

megoldáshoz. Időnként numerikus számítások esetében is egyszerűsítésekkel kell élni, ha túl bonyolult a modell, hogy belátható időn belül megoldható legyen a feladat (lásd például időjárás előrejelzések).

Numerikus számítások alkalmazása esetén is ki kell választani, hogy melyik módszert alkalmazzuk. Minden feladat típushoz sokféle kidolgozott módszer közül választhatunk. A módszerek különbözhetnek pontosságban, számításigényben és a programozás bonyolultságában is. A kiválasztott algoritmust az általunk használt programozási nyelven implementálni is kell. Matlab/Octave alkalmazása esetén a legtöbb esetben nem nekünk kell leprogramozni az algoritmusokat, mivel nagyon sok beépített numerikus módszer található meg bennük, azonban ezeknek a hátterével is célszerű megismerkedni, hogy helyesen tudjuk alkalmazni őket.

A feladat megoldása után le is kell valamilyen módon ellenőriznünk a megoldást. Egy nemlineáris egyenlet esetében ez történhet például visszahelyettesítéssel. Bonyolultabb esetekben, például differenciálegyenletek megoldásakor, a numerikus megoldást összehasonlíthatjuk egy hasonló probléma ismert megoldásával, vagy megoldhatjuk különböző módszerekkel és vizsgálhatjuk ezek eltéréseit.[4]

KEREKÍTÉSI HIBA, LEBEGŐPONTOS SZÁMÁBRÁZOLÁS

Mivel a feladatokat számítógép segítségével fogjuk megoldani, ismernünk kell a számítógépek korlátait is, tudnunk kell, hogy hogyan tárolják a számítógépek a számokat, milyen hibák fakadhatnak a tárolási módból, vagy az algoritmusok megválasztásából. Nézzük meg a következő egyszerű példákat!

Próbáljuk ki Matlab-ban a következőt:

```
> x1 = 0.3, x2 = 0.1+0.1+0.1
```

Egyenlő egymással ez a két szám? Látszatra igen.

```
> x1==x2
```

Ez utóbbi művelet (**==**) egy logikai művelet, eredménye igaz(1) vagy hamis(0). (A logikai nem egyenlő kifejezése Matlab-ban: **~=**). A kerekítési hiba miatt most hamis(0) értéket fogunk kapni. Ezek szerint 2x2 néha 5? Nézzük meg mekkora a $tg\left(\frac{\pi}{2}\right)$ értéke? A tangens függvény π/2-nél nincs értelmezve, a végtelenhez tart. Mi történik, ha Matlabban szeretnénk ezt kiszámolni? Hibaüzenetet lapunk?

```
> tan(pi/2)
```

Nincs hibaüzenet, Matlab-ban úgy tűnik, mégiscsak értelmezve van a függvény π/2-nél, hiszen kapunk rá választ: 1.6331e+16. Miért lehet ez? Ehhez nézzük meg, hogyan tárolja a számítógép a számokat! Erre több lehetőség is van, az egyik módszer a Matlab által is használt dupla pontosságú lebegőpontos számábrázolás (double). Ez a 64 bites számábrázolás a kettes számrendszeren alapszik:

$$(-1)^s \cdot m \cdot 2^{(e-1023)},$$

[4] Lásd: Amos Gilat, Vish Subramaniam (2011): Numerical Methods, An Introduction with Applications Using MATLAB (SI Version), John Wiley & Sons (Asia)

ahol *s* az előjel bit (1 bit), *m* a mantissza (az értékes jegyek 52 biten) és *e* az exponenciális kitevő (11 biten) (s+m+e=>1+11+52 = 64 bit).

A végtelen sok valós számból nem lehet mindent pontosan reprezentálni, lásd pl. 0.1 a kettes számrendszerben végtelen szakaszos tizedes (vagyis jelen esetben 'kettedes') tört lesz, amiből mi csak az első 52 bitet használjuk, a többit eldobjuk. Ez a kerekítési hiba.

$$\frac{1}{10} = \frac{1}{2^4} + \frac{1}{2^5} + \frac{1}{2^8} + \frac{1}{2^9} + \frac{1}{2^{12}} + \frac{1}{2^{13}} + \cdots$$

Kettes számrendszerben az egy tized egyenlő: 0.0001100110011001100110011...

Ezért van, hogy amikor összeadjuk a 0.1-et háromszor, az nem egyenlő a 0.3-mal, és a tangens függvény is értelmezve van Matlab-ban π/2-nél, mivel a π/2-t sem tudjuk pontosan megadni. Ezt a hibát nevezzük a véges számábrázolásból eredő kerekítési hibának. A kerekítési hiba (δ) nagyságrendjét a gépi pontossággal (ε_m - machine precision) lehet megadni. A numerikus számításokban $1 + \delta = 1$, ha $\delta < \varepsilon_m$. Matlab-ban a gépi epszilon értéke:

```
> eps
```

Értéke: 2.2204e-16. Ez a lebegőpontos számábrázolásban két egymást követően ábrázolható szám eltérése. Adjunk hozzá két számhoz 10^{-14} értéket, ami már a gépi epszilonnál nagyobb, az első szám 1 legyen, a második 12345.

```
> a = 1; b = 12345; c = 1e-14;
> a+c==a % a+c egyenlő-e a-val? - 0 (hamis)
> b+c==b % b+c egyenlő-e b-val? - 1 (igaz)
```

Miért lehet az, hogy az egyik esetben sikerült hozzáadni ugyanazt a gépi epszilonnál nagyobb értéket a számhoz, a másik esetben pedig nem?

A válasz erre az, hogy a gépi pontosság alapértéke az 1-től való távolság, de értéke a szám nagyságrendjével változik. Nézzük meg!

```
> eps(a) % 2.2204e-16
> eps(b) % 1.8190e-12
> eps(1e20) % 16384
```

Ezt azt jelenti, hogy 10^{20} után tárolható legkisebb szám 16384-gyel nagyobb. Ha 16384/2-t vagy annál kisebb számot adunk hozzá 10^{20}-hoz, akkor a szám értéke nem fog változni, ha ennél nagyobbat, akkor már igen (a kerekítés miatt, ha 16384/2+1-et adunk hozzá, akkor már felfelé fog kerekíteni).

```
> 1e20 == 1e20+16384/2 % a válasz igaz
```

Nézzünk egy másik tipikus hibát, a kioltó hibát, amikor közel azonos nagyságú számokat vonunk ki egymásból. Nézzük meg Matlab-ban a következő példát:

```
> x1 = 10.000000000000004 % 14 db 0 a tizedespont után a 4-es előtt
> y1 = 10.00000000000004 % 13 db 0 a tizedespont után a 4-es előtt
> (y1-10)/(x1-10)
```

A várt eredmény: 0.000000000000004 / 0.00000000000004 = 10, a kapott eredmény azonban: 11.5!

Miért fontos nekünk a numerikus hibák ismerete? Érdemes néhány tanulságos esetet megnézni a numerikus hibák okozta katasztrófák között! Az Öbölháború idejében

1991-ben egy Patriot légvédelmi rakéta nem találta el az iraki Scud rakétát, ami miatt 28-an meghaltak. Az oka numerikus hiba volt. A tizedmásodpercben mért időket a rendszer megszorozta 1/10-del, hogy másodpercet kapjon, mivel ezt nem lehetett pontosan reprezentálni binárisan, fellépett egy kis kerekítési hiba, sokszor egymás után elvégezve a műveletet a hiba halmozódott. 100 órás üzem esetén az eltérés: 0.34 másodperc volt, ami alatt egy Scud rakéta több, mint fél kilométert tesz meg (1.676 m/s sebességgel).

Ugyancsak numerikus hiba okozta 1996-ban az ESA Ariane 5 rakétájának 40 másodperccel a kilövése után történő felrobbanását! („Some disasters caused by numerical errors" - http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html). Érdekes lehet még a következő oldal is: https://www5.in.tum.de/persons/huckle/bugse.html, ahol egy nagyobb gyűjtemény található szoftverhibák okozta problémákról (sok köztük a numerikus számítási probléma).

ABSZOLÚT ÉS RELATÍV HIBA

A hibákat többféleképpen csoportosíthatjuk, a pontos értéktől való tényleges eltérést abszolút hibának nevezzük, azonban a valóságban sokszor nem ezzel lehet a legjobban reprezentálni a hiba nagyságát, célszerű bevezetni a relatív hiba fogalmát is. Legyen egy x valós szám közelítése $\tilde{x}$.

Abszolút hibának nevezzük a tényleges eltérést: $\Delta = |x - \tilde{x}|$

A relatív hiba az abszolút hiba osztva x értékével: $\varepsilon = \frac{|x - \tilde{x}|}{|x|}$

Amikor x egy körüli érték, akkor nincs lényeges eltérés a kettő között, azonban amikor x>>1, a relatív hiba jobban tükrözi a hiba jelentőségét. Nézzünk erre egy példát! Legyen két távolság (t_1, t_2), amit becslünk $(\tilde{t_1}, \tilde{t_1})$.

$t_1 = 1000\ m; \quad \tilde{t_1} = 900\ m;$ $\Delta = 100\ m; \quad \varepsilon = 10\ \%;$	$t_2 = 200\ m; \quad \tilde{t_1} = 100\ m;$ $\Delta = 100\ m; \quad \varepsilon = 50\ \%;$

Az abszolút hiba mindkét esetben 100 m, de az első becslés mégis sokkal pontosabb, mint a második és ez a relatív hibák nagyságában is tükröződik (10% ill. 50%).

STABILITÁS, KONDÍCIÓSZÁM

Miután láttuk, hogy a numerikus hibákkal együtt kell élnünk a számítások során, a következő fontos kérdés, hogyan kezeljük őket? Megbízhatunk-e a kapott eredményben? Ehhez még két fogalmat kell megismernünk, az adott probléma érzékenységét/kondícionáltságát és az algoritmus stabilitását.

- Egy matematikai probléma jól kondicionált ha a bemeneti paraméterek kis változására az eredmény is kis mértékben változik.
- Egy algoritmus numerikusan stabil ha bemeneti paraméterek kis változására az eredmény is kis mértékben változik.

A pontosság függ a probléma kondicionáltságától és az algoritmus stabilitásától.

Pontatlanságot okozhat, ha stabil algoritmust alkalmazunk rosszul kondicionált problémára, vagy instabil algoritmust alkalmazunk jól kondicionált problémára.

STABILITÁS

Nézzük meg a következő másodfokú egyenlet megoldását!

$$x^2 - 100.0001\,x + 0.01 = 0$$

Ennek az egzakt megoldása a következő: $x_1 = 100; x_2 = 0.0001;$ Megoldás felírható a másodfokú megoldóképlettel:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}; \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a};$$

Oldjuk meg Matlab-ban, az eredmények megjelenítését állítsuk át több tizedes jegyre!

```
> format long;
> a = 1; b = -100.0001; c = 0.01;
> D = sqrt(b^2 - 4*a*c) % 99.999899999999997
> x1 = (-b + D)/(2*a) % 100
> x2 = (-b - D)/(2*a) % 1.000000000033197e-04
```

x_2-re nem kaptunk pontos eredményt a kerekítési hiba miatt. Miután *b* negatív, így ebben az esetben a számlálóban két egymáshoz nagyon közeli értéket kell kivonni egymásból, itt is fellép a kioltó hiba.

Sok esetben, amikor a matematikai kifejezés két egymással közel megegyező kifejezés különbségét tartalmazza, a feladat átalakítható olyan formába is, ami kevésbé érzékeny a kerekítési hibákra. x_2 esetében ezt megtehetjük, ha megszorozzuk a formulát $(-b + \sqrt{b^2 - 4ac})/(-b + \sqrt{b^2 - 4ac})$-vel:

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \cdot \frac{-b + \sqrt{b^2 - 4ac}}{-b + \sqrt{b^2 - 4ac}} = \frac{2c}{-b + \sqrt{b^2 - 4ac}};$$

Használjuk most ez utóbbi kifejezést a megoldáshoz:

```
> x2m = (2*c)/(-b+D) % 1.000000000000000e-04
```

Most a várt eredményt kaptuk. Nézzünk egy másik példát az algoritmus megválasztásának fontosságára! Kétféle módon is közelíthetjük az e^{-x} értékét:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots$$

$$e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots}$$

Számoljuk ki a függvény értékét x = 8.3 helyen a fent megadott két módszerrel! Töltsük be az emx.m függvényt, ami két kimenettel megadja a kétféle közelítést n tag esetében, x helyen!

```
> function [f g] = emx(x,n)
>     f = 1; % első közelítés 1 - x + x^2/2 - x^3/6 + ...
>     p = 1; % első közelítés a nevezőre (1 + x + x^2/2 + x^3/6 + ...)
```

```
>       for i=1:n
>           s = x^i/factorial(i);
>           f = f +(-1)^i*s;
>           p = p + s;
>           g = 1 / p;
>       end
>   end
```

A megoldás pontos értéke: exp(-8.3) = 2.4852e-04

Próbáljuk ki az előbbi függvényt n = 10, 20, 30 esetben! Álljunk vissza a kevesebb számjegy kijelzésére, most elég lesz ez is.

```
>   format short
>   megoldas = exp(-8.3) % 2.4852e-04
>   [f g] = emx(8.3,10) % f=188.0344,   g=3.1657e-04
>   [f g] = emx(8.3,20) % f=0.2833,     g=2.4856e-04
>   [f g] = emx(8.3,30) % f=2.5151e-04, g=2.4852e-04
```

Az eredmények:

n=	10	20	30
f	188.0344	0.2833	2.5151e-04
g	3.1657e-04	2.4856e-04	2.4852e-04

Látjuk, hogy a két algoritmus közül a második sokkal hamarabb közelíti meg a pontos értéket, nem mindegy milyen módszerrel oldjuk meg a feladatot!

KONDÍCIÓSZÁM

Oldjuk meg a következő lineáris egyenletrendszert!

$$6x_1 - 2x_2 = 10$$

$$11.5x_1 - 3.85x_2 = 17$$

Az egyenletrendszer mátrixos alakban ($A \cdot x = b$):

$$A = \begin{pmatrix} 6 & -2 \\ 11.5 & -3.85 \end{pmatrix}; \quad b = \begin{pmatrix} 10 \\ 17 \end{pmatrix}$$

Oldjuk meg ezt a feladatot Matlab-ban. A megoldás matematikailag: $x = A^{-1} \cdot b$. Matlab-ban az **inv** parancs számítja egy mátrix inverzét. Oldjuk meg a feladatot!

```
>   A = [6,-2; 11.5,-3.85]; b = [10; 17];
>   x = inv(A)*b % 45.0000; 130.0000
```

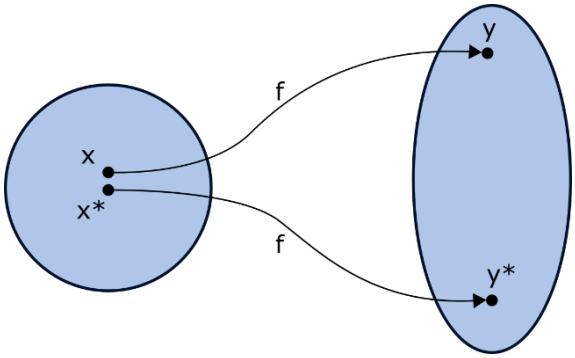
Megoldás a 45 és a 130 lett. Módosítsuk egy kicsit a második egyenletben az x_2 együtthatóját -3.85-ről -3.84-re, és oldjuk meg újra a feladatot!

```
>   A = [6,-2; 11.5,-3.84]
>   x = inv(A)*b % 110.0000; 325.0000
```

Most a megoldás 110 és 325 lett. Csak egy kicsit változtattunk az egyenletrendszeren, mégis óriási lett a változás a végeredményben! Azt vártuk volna, hogy mivel a két

bemenet nagyon közel van egymáshoz a megoldások is hasonlóak lesznek. Nem így történt. Mi okozhatja ezt?

Vannak olyan mátrixok, amelyek nagyon érzékenyek a bemenet kis megváltozására. Ezt az érzékenységet lehet mérni a mátrix kondíciószámával. A kondíciószám (κ) teremt kapcsolatot a kimenet relatív hibája és a bemenet relatív hibája között. Minél nagyobb ez a szám, a bemenet kis megváltozása annál nagyobb változást fog okozni a kimenetet illetően.



$$\kappa = \left| \frac{\frac{f(x) - f(\tilde{x})}{f(\tilde{x})}}{\frac{x - \tilde{x}}{\tilde{x}}} \right| = \left| \frac{\tilde{x}}{f(\tilde{x})} \cdot \frac{f(x) - f(\tilde{x})}{x - \tilde{x}} \right| = \left| \frac{\tilde{x} \cdot f'(\tilde{x})}{f(\tilde{x})} \right|$$

Nézzük meg ennek a mátrixnak a kondíció számát!

```
> cond(A) % 4.6749e+03
```

Eredménye: 4674.9 lett. Minél nagyobb ez a szám, annál bizonytalanabb a megoldás, ugyanis a bemenet kis hibája ugyanennyi szeresére nagyítódik föl a kimenetnél! Erre nagyon oda kell figyelni a mérnöki munkák során, ahol a bemeneti mérések, állandók többnyire csak közelítő értékek, hibákkal terheltek.

CSONKÍTÁSI HIBA

A véges számábrázolásból adódó kerekítési hiba hatását már láttuk, és azt is, hogy mennyire fontos ennek következtében az algoritmus helyes megválasztása, például kerülni kell a közel azonos nagyságú számok kivonását, ha van rá mód.

Egy másik fontos hiba akkor lép fel, amikor az egzakt matematikai kifejezés helyett annak közelítését alkalmazzuk a numerikus számítások során, például: Taylor-soros közelítés, numerikus deriválás, integrálás. Általában akkor lép fel a probléma, amikor egy bonyolult, szimbolikusan nem megoldható problémát, egyszerűbb, számítógéppel könnyebben kezelhető problémával helyettesítjük.

A Taylor-soros közelítésre már láttunk példát az előzőekben is, minél több tagot vettünk figyelembe, értelemszerűen annál kisebb volt a csonkítási hiba és megfelelően választott algoritmus esetén viszonylag gyorsan sikerült jó közelítést elérni. Most nézzük meg az e^x közelítését Taylor-sorral 4 taggal: $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!}$

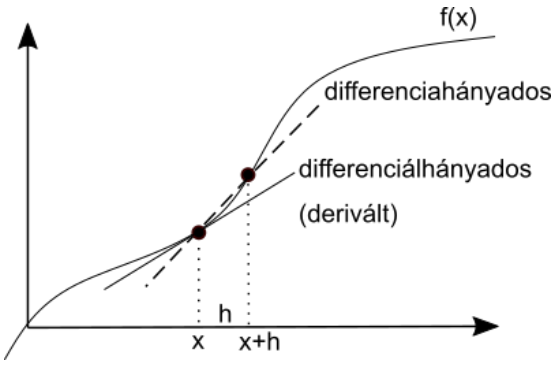
```
> x=1
> f = exp(x) % 2.7183 - tényleges érték
> g = 1 + x + x^2/2 + x^3/6 % 2.6667 - csonkítási hibával terhelt érték
```

Nem csak a Taylor soros közelítés tartalmaz csonkítási hibát, hanem a numerikus integrálás is, vagy a derivált differenciahányadossal való közelítése is. Nézzünk ez utóbbira egy példát:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

A csonkítási hiba felső korlátja becsülhető. Határozzuk meg a csonkítási hiba felső korlátját az alábbi függvény deriváltjának a differenciahányadossal történő közelítése esetén, x=2 helyen[5]!

$$y = x^3$$



Közelítsük numerikusan a deriváltat a differenciahányadossal!

$$f'(x) \approx \frac{f(x+h)-f(x)}{h} = \frac{(2+h)^3 - 2^3}{h}$$

A csonkítási hibát a Taylor-soros közelítés alapján becsülhetjük:

$$f(x+h) = f(x) + f'(x)\cdot h + \frac{f''(\xi)}{2}\cdot h^2$$

, ahol $x < \xi < x+h$. Ezért átrendezve igaz a következő:

$$\left|\frac{f(x+h)-f(x)}{h} - f'(x)\right| \le \frac{f''(\xi)}{2}\cdot h$$

A fenti egyenlet bal oldalán a csonkítási hibát találjuk (a közelítés és a tényleges érték eltérése), a jobb oldalon pedig ehhez egy felső korlátot, aminél biztosan kisebb lesz a hiba. Jelöljük H-val ezt a hibát, a felső korlátot pedig ε-nal. Esetünkben pontosan ismerjük az első ($y' = 3\,x^2 = 12$) és a második deriváltat is ($y'' = 6x$), így a csonkítási hibára igaz a következő:

$$H(h) = \left|\frac{(2+h)^3 - 2^3}{h} - 12\right| \le \frac{f''(\xi)}{2}\cdot h = \frac{6\,\xi\,h}{2} = 3\,\xi\,h$$

A csonkítási hiba becsült felső korlátja h függvényében maximális ξ esetén ($\xi = x + h$), $x = 2$ helyen:

$$\varepsilon(h) = 3\,(2+h)\,h$$

A fenti képletből látható, amint az várható is volt, hogy minél kisebb h intervallumokra osztjuk fel a függvényt a differenciahányadosok számításához, annál kisebb lesz a csonkítási hiba.

TELJES HIBA

Ábrázoljuk az előző derivált közelítésének becsült felső korlátját és számítsuk ki a tényleges hibát is a lépésköz változásának függvényében! Vegyük fel a következő lépésközöket:

$$h = 1, 10^{-1}, 10^{-2}, \cdots, 10^{-15}$$

```
> format long
> n = 0:-1:-15, h = 10.^n
```

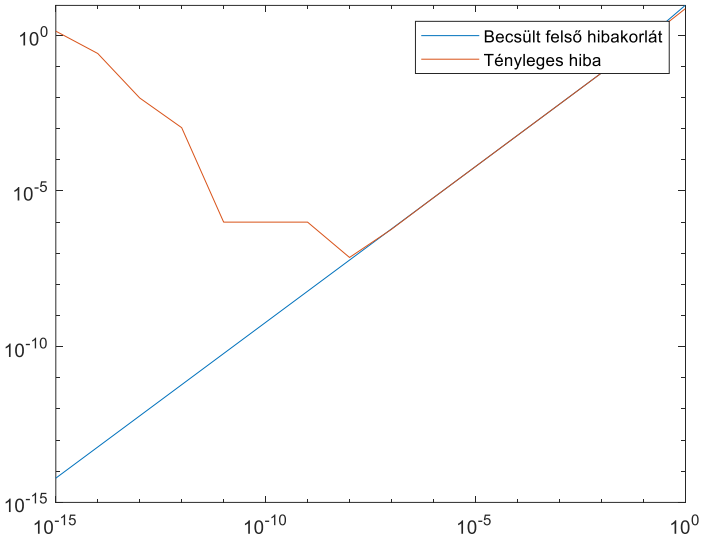
[5] Paláncz Béla numerikus módszerek példatára alapján

Adjuk meg egysoros függvényként a becsült felső korlátot és a tényleges hibát is! (Figyeljünk a . használatára, mivel most vektorok elemenkénti műveletéről van szó)

```
> e = @(h) 3*(2 + h).*h; % becsült felső korlát
> H = @(h) abs(((2 + h).^3 - 8)./h - 12) % tényleges hiba
```

Számoljuk ki a két függvény értékét a különböző lépésközökhöz és ábrázoljuk az eredményeket log-log koordináta rendszerben (ezt a **loglog** paranccsal tehetjük meg)!

```
> figure(1)
> loglog(h,e(h)); hold on
> loglog(h,H(h))
> legend('Becsült felső hibakorlát','Tényleges hiba')
```



Mit látunk a fenti ábrán? Ahogy csökken a lépésköz, úgy csökken a csonkítási hiba becsült felső korlátja is és egy darabig a tényleges hiba is ennek megfelelően, azonban egy pont után (valahol 10^{-8} környékén) hirtelen elkezd nőni újra a tényleges hiba. MI lehet ennek az oka? Ez abból adódik, hogy a teljes hiba a csonkítási és a kerekítési hiba összegeként áll elő. Nagyjából 10^{-8} értékig a csonkítási hiba dominál, utána viszont erőteljesen elkezd nőni a kerekítési hiba. Ennek a jelenségnek igen fontos szerepe van például a differenciálegyenletek numerikus megoldásánál!

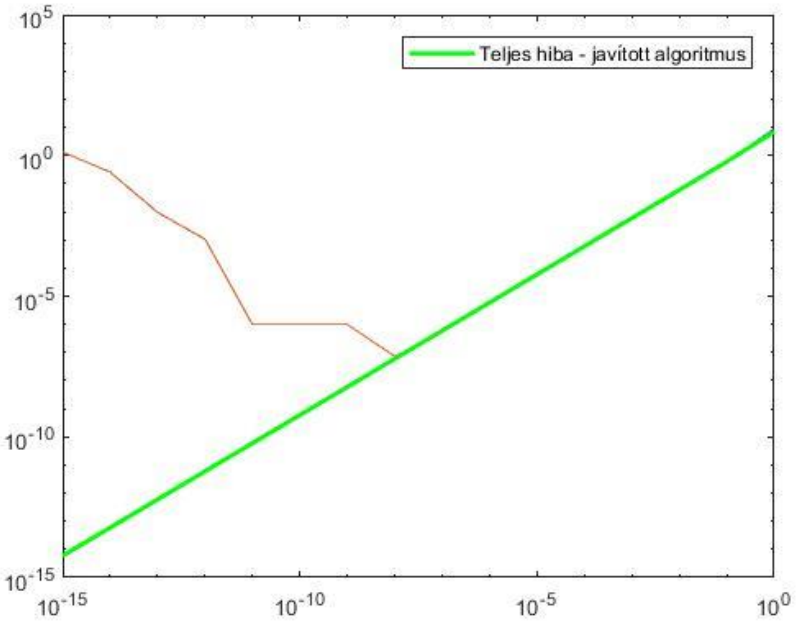
Természetesen hasonlóan a korábban látottaknál itt is lehet olyan algoritmust választani, ami kevésbé érzékeny a kerekítési hibára, hiszen a fő hiba itt is a kioltó hiba, mivel a differenciahányados számlálójában két közel azonos számot vonunk ki egymásból, minél kisebb h, annál kisebb a két szám közötti eltérés. Most a differenciahányadost könnyen egyszerűsíthetjük:

$$\frac{(2+h)^3 - 2^3}{h} = 12 + 6\,h + h^2$$

Ebből a csonkítási hiba:

$$H(h) = |12 + 6\,h + h^2 - 12| = |6\,h + h^2|$$

```
> H2 = @(h) abs(6*h + h.^2)
> abr1=loglog(h,H2(h),'g','LineWidth',2)
> legend(abr1,'Teljes hiba - javított algoritmus')
```



A differenciahányadost itt mi egyszerűsítettük manuálisan, természetesen lehetőség van ilyen egyszerűsítésekre Matlab-ban is, ezek azonban már a szimbolikus számítások témakörébe tartoznak, ami a Symbolic Math Toolbox része (Octave-ban ehhez telepíteni kell a symbolic package-t, lásd az 1. gyakorlat Kiegészítés Octave-hoz fejezetét).

Szimbolikus számítások során nem konkrét számokkal számolunk, hanem változókkal. Ehhez először meg kell adni a Matlab számára **syms** paranccsal, hogy melyek lesznek a szimbolikus változóink, ezekkel a szimbolikus változókkal meg kell hívjuk a függvényt, ekkor egyszerűsíthetjük a kifejezést a **simplify** paranccsal. A **simplify** parancs eredménye egy szimbolikus változó lesz, amibe nem lehet konkrét számokat behelyettesíteni, csak, ha visszaalakítjuk hagyományos függvénnyé a kifejezést a **matlabFunction** paranccsal.

```
> syms h
> Hsym = simplify(H(h)) % Hsym = abs(h*(h + 6))
> H2 = matlabFunction(Hsym) % H2 = @(h)abs(h.*(h+6.0))
```

A Workspace-ben is láthatjuk, hogy Hsym értéke szimbolikus. Nézzük meg mi történik, ha megpróbálunk egy konkrét értéket behelyettesíteni!

```
> H2(1e-1) % 0.610000000000000
> Hsym(1e-1)
Subscript indices must either be real positive integers or logicals.

Error in sym/subsref (line 841)
            R_tilde = builtin('subsref',L_tilde,Idx);

Error in gyak3 (line 78)
Hsym(1e-1)
```

A második esetben hibaüzenetet kapunk, mivel szimbolikus kifejezésbe nem lehet behelyettesíteni egy értéket.

A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

==	-	Logikai egyenlőség
~=	-	Logikai ’nem egyenlő’
eps	-	Gépi epszilon/gépi pontosság nagysága,
factorial	-	Faktoriális, n!
inv	-	Mátrix inverze
cond	-	Kondíció szám
loglog	-	Ábrázolás logaritmikus skálán (mindkét tengelyen)
syms	-	Szimbolikus változók, kifejezések definiálása
simplify	-	Szimbolikus kifejezések egyszerűsítése
matlabFunction	-	Szimbolikus kifejezések függvénnyé alakítása

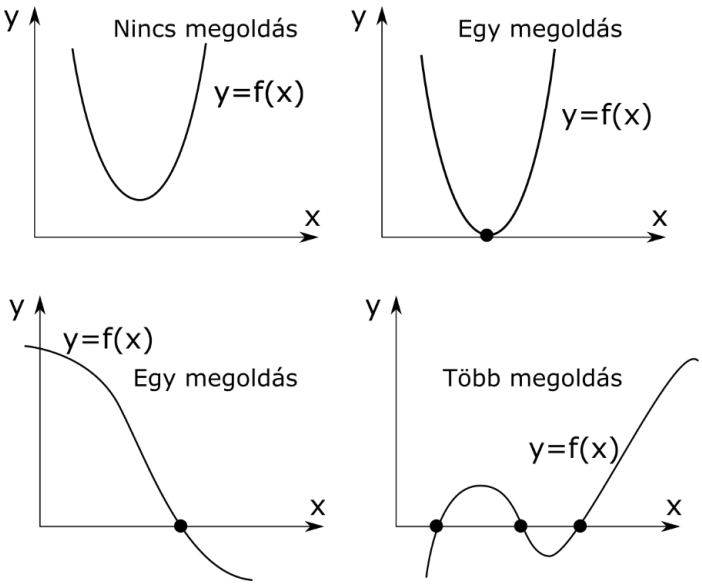
4. NEMLINEÁRIS EGYENLETEK GYÖKEI

Az $f(x) = 0$ egyenlet numerikus megoldása, zérushelyeinek/gyökeinek a megtalálása, sok esetben merül fel megoldandó feladatként. Ahhoz, hogy megoldjunk egy nemlineáris feladatot általánosan, az egyenletet először (ha nem ilyen formában volt megadva) át kell alakítanunk

$$f(x) = 0$$

alakba. Ennek az egyenletnek a megoldását az egyenlet gyökének, vagy zérushelyének is nevezik. Az x megoldás visszahelyettesítve kielégíti az egyenletet, vagyis az egyenlet értéke nulla vagy közelítőleg (adott hibahatáron belül) nulla lesz. Grafikusan a megoldás az a pont, ahol a függvény metszi az x tengelyt.

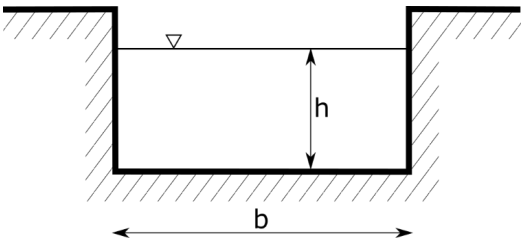
Az $f(x) = 0$ egyenlet megoldása sok esetben csak numerikusan, iterációkkal lehetséges, azaz akkor fogadjuk el a megoldást, ha egy megadott Δ hibakorlát alatt van, $f(x) \leq \Delta$. A megoldási módszerek többsége ún. lokális módszer, azaz az iterációhoz szükség van egy vagy több induló értékre. A megoldásra sokféle algoritmust dolgoztak ki. Az algoritmusok jellemzője, hogy egyszerre csak egy gyök meghatározását teszik lehetővé, több zérushely esetén, több kezdeti értékkel kell lefuttatni őket.



CSATORNA MÉRETEZÉSI PÉLDA

Nézzünk meg egy csatorna méretezési feladatot a hidraulika témaköréből!

A nyílt felszínű csatorna alakja, anyaga, esése, szélessége és a vízmagasság függvényében levezethető az elszállított vízhozam nagysága. Nézzük például egy szabadfelszínű, téglalap keresztmetszetű csatorna vízhozamának a képletét!



$$Q = \frac{\sqrt{S}}{n} \cdot \frac{(b \cdot h)^{\frac{5}{3}}}{(b + 2 \cdot h)^{\frac{2}{3}}}$$

ahol Q - vízhozam, n - Manning-féle érdességi együttható, S - esés, b - csatorna szélessége, h - vízmélység. Mekkora vízhozam tartozik 1 és 2 méteres vízmélységhez? Határozzuk meg a csatornában a vízszint magasságát, h-t, 3 m^3/s mértékadó vízhozam esetén, 0.8 ezrelék esés esetén, 0.02 Manning-féle

érdességi együttható és 2 m csatorna szélesség mellett! Vagyis: $S = 0.0008; n = 0.02; Q = 3\ {}^{m^3}/_{s}; b = 2\ m$[6]

A feladat első felére, hogy mekkora vízhozam tartozik 1 és 2 méteres vízmélységhez, könnyű válaszolni, ez csak egyszerű behelyettesítést jelent. Először adjuk meg a változók értékeit és definiáljuk a vízhozamot (Q) a vízmélység (h) függvényében!

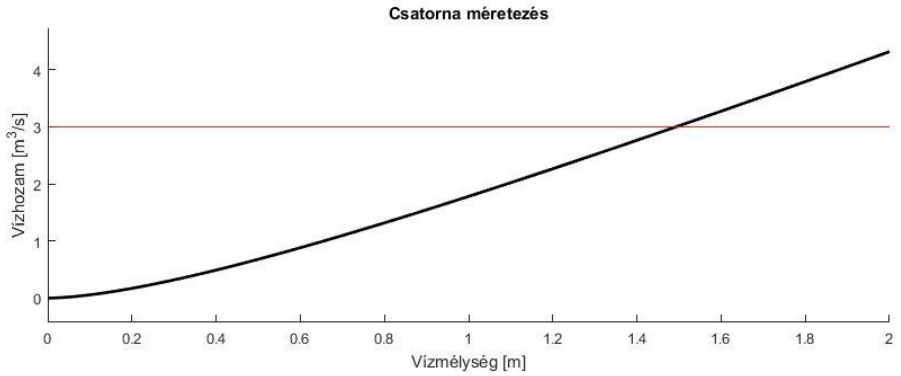
```
> clear all; clc;close all;
> % Változók értékadása
> S = 0.0008; n = 0.02; Q = 3; b = 2;
> % Vízhozam egysoros függvényként: h független változó
> Q=@(h) sqrt(S)/n*(b*h).^(5/3)./(b+2*h).^(2/3);
```

Mekkora vízhozam tartozik 1 és 2 m-es vízmélységhez?

```
> % Mekkora vízmélység tartozik 1 és 2 m-es vízmélységhez?
> Q(1), Q(2) %  1.7818 illetve 4.3170 m^3/s
```

A második kérdés az volt, hogy mekkora lesz a *Q*=3 m^3/s vízhozamhoz tartozó vízmélység. Mivel nem tudjuk kifejezni az egyenletből *h*-t *Q* függvényében, ezért itt most csak közelítő, numerikus megoldás jöhet szóba. A két behelyettesített érték alapján az 1 m-hez 1.78 m^3/s, a 2 m-hez pedig 4.31 m^3/s vízhozam tartozik, tehát valahol 1 és 2 méter között lesz a keresett vízmélység a mértékadó 3 m^3/s vízhozamhoz. Ábrázoljuk a függvényt a 0-2 m tartományon és rajzoljuk be a kívánt Q=3 m^3/s értéket is! Függvényeket az **fplot** paranccsal tudunk megjeleníteni, összetartozó pontpárokat pedig a **plot** paranccsal. Az **fplot** paranccsal megjelenített görbék tulajdonságait a **set** paranccsal állíthatjuk be (pl. '**Color**','**LineWidth**' opció), ha előtte valamilyen változóhoz hozzárendeljük az ábrát.

```
> % A függvény ábrázolása a [0;2] intervallumon
> figure(1); hold on;
> h1 = fplot(Q, [0 2]);
> set(h1,'Color','k','LineWidth',2)
> % y=3 vonal berajzolása a [0 2] intervallumon két pontot megadva
> plot([0,2],[3,3],'r')
> % Ábra feliratozás
> title('Csatorna méretezés');
> xlabel('Vízmélység [m]');
> ylabel('Vízhozam [m^3/s]');
```

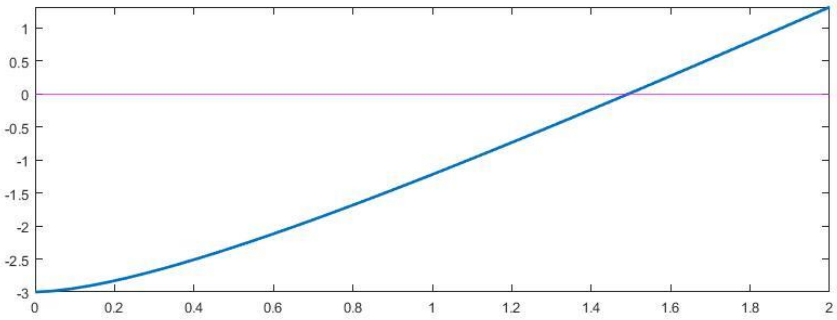


Az ábrából látszik, hogy a megoldás valahol 1.4 és 1.6 között lesz. A megoldás megtalálásához használható algoritmusok viszont mindig a zérushelyet/gyökhelyet

[6] A példa Paláncz Béla (2012): Numerikus módszerek példatárából való, kis átalakítással.

keresik, vagyis hol metszi a függvény az x tengelyt. Ezt az alakot kell nekünk is definiálni egy átrendezés után, vagyis át kell alakítanunk a Q(h)=3 egyenletet, f(h)=0 alakba:

```
> f = @(h) Q(h)-3
> figure(2); h2 = fplot(f, [0 2]);
> set(h2,'LineWidth',2)
> % y=0 vonal berajzolása a [0 2] intervallumon két pontot megadva
> hold on; plot([0,2],[0,0],'m')
```



A megoldás, a gyökhely vagy zérushely meghatározása többféle módszerrel történhet, a két fő típusa ezeknek a zárt és a nyílt intervallum módszerek.

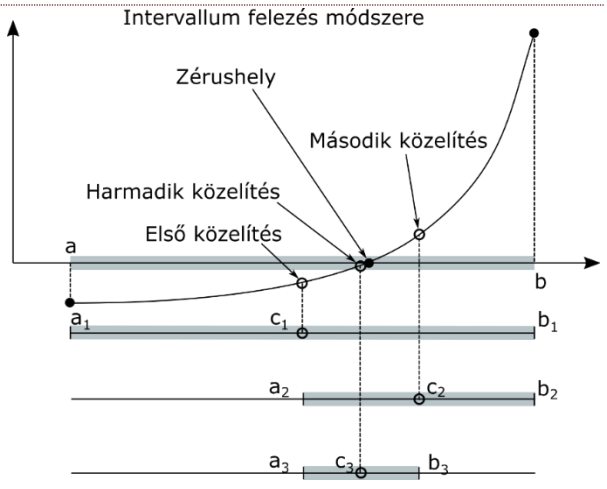
ZÁRT INTERVALLUM MÓDSZEREK

Zárt intervallum módszerek esetén egy [a,b] intervallumot adunk meg, amiben benne van a megoldás. Ebben az esetben az intervallum két végpontjában ellentétes előjelük lesz a függvényértékeknek, azaz $f(a) \cdot f(b) < 0$. hiszen a kettő között vált előjelet a függvény, áthalad a nullán. Ebben az esetben az intervallumon belül biztosan található zérushely (c), amennyiben f(x) folytonos. Ilyenkor fokozatosan szűkítjük az intervallum nagyságát, vizsgálva a végpontok függvény értékeit, addig, amíg már vagy nagyon közel lesz a függvény értéke 0-hoz (megadott pontosságon belül), vagy maga az intervallum lesz nagyon kicsi. Többféle zárt intervallum módszer létezik, különbség abban van közöttük, hogy milyen módszerrel szűkítik az intervallum nagyságát. A zárt intervallum módszerek mindig eredményre vezetnek, csak az egyik módszer lassabban, a másik gyorsabban.

INTERVALLUM FELEZÉS MÓDSZERE (BISECTION METHOD)

A kezdeti intervallumot megfelezve megvizsgáljuk a végpontokban a függvényértékeket, ahol eltérő előjelű, az lesz az új intervallum, ezt ismételjük a kívánt Δ pontosságig.

1. $c = (a + b)/2$
2. ha $|f(c)| < \Delta \rightarrow$ vége
3. ha $f(a) \cdot f(c) < 0$, akkor $b = c$, különben $a = c$



HÚRMÓDSZER (REGULA FALSI METHOD)

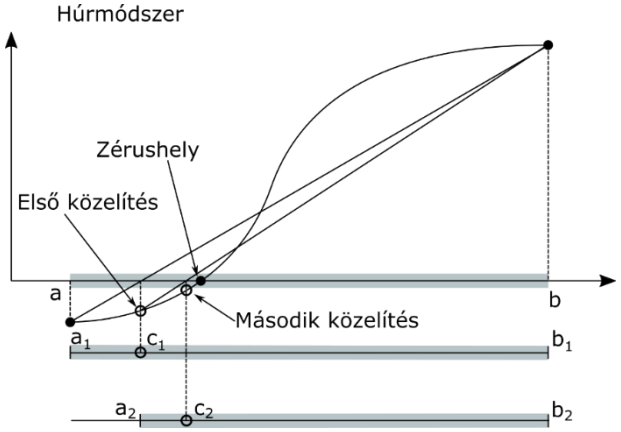
Az intervallumfelezés módszerénél hatékonyabb megoldás a húrmódszer (általában gyorsabban konvergál). Itt az intervallum végein kiszámolt f(a) és f(b) pontokat összekötő húrnak az x tengellyel való metszéspontját határozzuk meg mint új közelítő értéket. Az egyenes x tengellyel való metszéspontja (y=0), hasonló háromszögekből kiszámítható:

$$\frac{b-a}{f(b)-f(a)} = \frac{c-a}{0-f(a)}$$

1. Megoldása x=c:

$$c = \frac{a \cdot f(b) - b \cdot f(a)}{f(b)-f(a)}$$

2. ha $|f(c)| < \Delta \rightarrow$ vége
3. ha $f(a) \cdot f(c) < 0$, akkor $b = c$, különben $a = c$



INTERVALLUM FELEZÉS ÉS HÚRMÓDSZER MATLAB-BAN

Nézzük meg, hogyan tudunk saját Matlab/Octave függvényt írni az intervallum felezés módszerére! Adjunk meg egy Δ hibakorlátot és egy maximális iteráció számot (N)! Ehhez feltétel vezérelt ciklusra lesz szükségünk (**while** ciklus). A leállási feltételek, ha elérjük a közelítés hibaküszöbét, vagy a maximális iteráció számot! Itt szükséges lesz egy logikai ÉS használatára a két feltétel együttes vizsgálatára, ezt többféleképpen is megadhatjuk Matlab-ban: feltetel1 **&&** feltetel2 vagy **and**(feltetel1, feltetel2).

```
> function [c, i] = intfelezes(f, a, b, delta, N)
>     c = (a+b)/2;    % Intervallumfelezés (1. iteráció)
>     i = 1;          % Iterációk száma
>     % Leállási feltétel:
>     % elérjük a közelítés küszöbértékét vagy a maximális iteráció
  számot
>     while abs (f(c)) > delta && i <= N
>         if f(c)*f(a) < 0
>             b = c;
>         else
>             a = c;
>         end;
>         i = i + 1;
>         c = (a+b)/2;
>     end;
> end
```

A húr módszer implementálása ugyanígy történhet, csupán a c kiszámításában van különbség az első és a további iterációkban, `c=(a+b)/2` helyett:

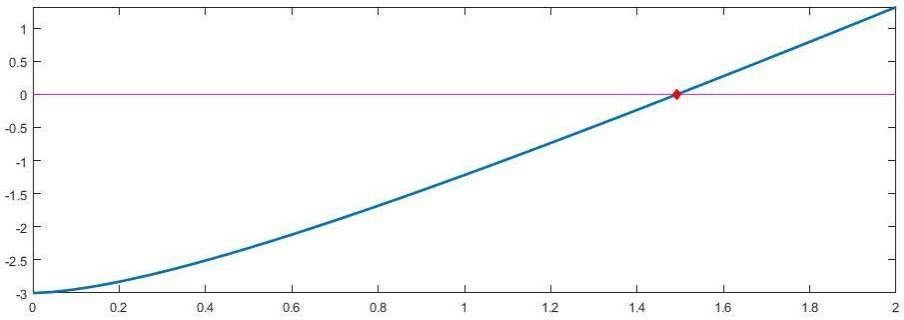
```
> c = (a*f(b) - b*f(a))/(f(b) - f(a));
```

CSATORNA MÉRETEZÉS ZÁRT INTERVALLUM MÓDSZEREKKEL

Használjuk az általunk megírt függvényeket (**intfelezes.m** illetve **hur.m**) a megoldáshoz. Nézzük meg a kapott függvény értékeket és rajzoljuk be az ábrába a

megoldást. Ahhoz, hogy a saját függvényeinket (intfelezes.m, hur.m) használni tudjuk, ugyanabban a könyvtárban kell legyenek, mint, ahová dolgozunk!

```
> [xint, iint]= intfelezes(f, 1.4, 1.6, 1e-9, 100) % interv. felezés
> % xint = 1.4929, iint = 28
> [xhur, ihur]= hur(f, 1.4, 1.6, 1e-9, 100) % húr módszer
> % xhur = 1.4929, ihur = 4
> % Ellenőrzés
> f(xint), f(xhur) %   -4.5629e-10,  -1.3299e-10
> % Ábrázolás az eredeti függvénybe visszahelyettesítve
> plot(xhur, f(xhur), 'rd','MarkerFaceColor','r');
```



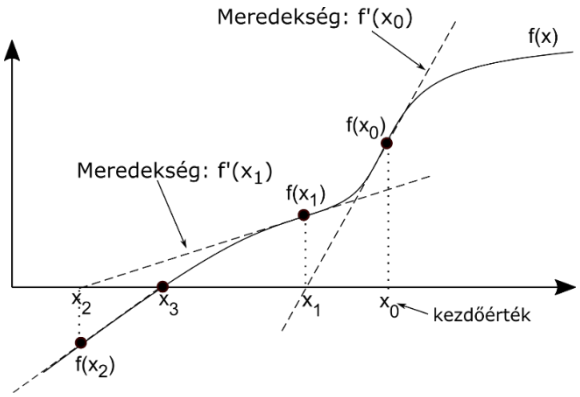
Értékeljük az eredményeket! A megoldások ugyanazok? Melyik volt a gyorsabb algoritmus? Melyik adta a pontosabb eredményt?

NYÍLT INTERVALLUM MÓDSZEREK

A nyílt intervallum módszereknél csak a zérushely egy közelítő x_0 értékét ismerjük. A módszereknek a konvergenciája általában sokkal gyorsabb, mint a zárt intervallum módszereké, amennyiben konvergálnak! Szigorúbb konvergencia feltételeket igényelnek, és nem mindig adnak eredményt. Többféle módszer van ezekből is például a gradiens típusú Newton és szelő módszer és a gradiens nélküli fixpont, és ennek tovább fejlesztése a Wegstein módszer (f(x)=0 egyenletet g(x)=x alakra hozva).

NEWTON MÓDSZER (NEWTON'S METHOD)

A Newton módszer (vagy Newton-Raphson módszer) abban az esetben használható, ha a függvény folytonos és differenciálható és tudjuk, hogy egy adott kezdőérték közelében van megoldása a feladatnak. A módszer elején az x_0 kezdőpontban kiszámoljuk a függvény értékét ($f(x_0)$) és az (x_0,$f(x_0)$) ponthoz húzott érintőnek megkeressük az x tengellyel való metszéspontját. Ez adja a következő x_1 közelítő értéket. Addig folytatjuk, amíg f(x) értéke kisebb nem lesz egy megadott Δ értéknél vagy el nem érjük a megadott maximális iteráció számot.



f'(x) az érintő meredeksége az ábra alapján:

$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}}$$

Ebből levezethető az alábbi iterációs képlet, a Newton-módszer általános alakja:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

A Newton-módszer levezethető a függvény Taylor-soros közelítéséből is, az első két tagot figyelembe véve (a függvény linearizálásából):

$f(x) \approx f(x_i) + f'(x_i) \cdot (x_{i+1} - x_i) = 0$, ahol $f(x) = 0$.

A Newton-módszer, ha működik, akkor általában gyorsan konvergál. Azonban látjuk, hogy a módszer alkalmazásához ismerni kell a függvény deriváltját is. Ez bizonyos esetekben nehézségekbe ütközik, nem ismerjük, vagy túl bonyolult lenne kiszámolni, ekkor alkalmazhatjuk a szelő-módszert, ami a Newton-módszer közelítése.

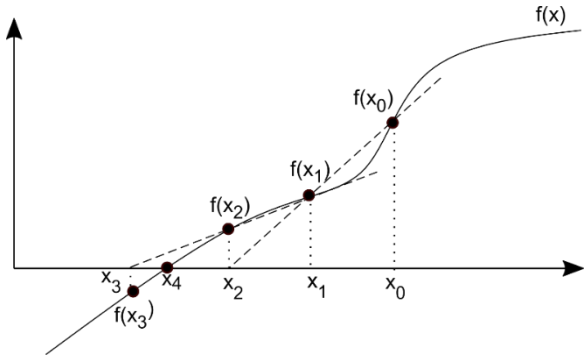
SZELŐ MÓDSZER (SECANT METHOD)[7]

A szelő módszer a Newton módszer véges differencia közelítése. Akkor alkalmazható, ha nem ismerjük a függvény deriváltját (vagy nehéz lenne előállítani). Általában lassabban konvergál (lásd a két ábrán ugyanazt a példát), és az elején két pont felvétele szükséges, x_0 és x_1 az ábrán (de ellentétben a húr módszerrel, ezeknek nem kell feltétlenül közrefogniuk a megoldást). Helyettesítsük be a Newton módszer képletébe a derivált véges differencia közelítését, az első iterációban a két felvett pont adatait használva!

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

ebből levezethető a szelő módszer általános képlete:

$$x_{i+1} = x_i - f(x_i) \cdot \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$



NEWTON MÓDSZER MATLAB-BAN

A Newton módszerhez a függvény deriváltját is szükséges ismerni (df), ha ez megvan, akkor a következő függvénnyel oldhatjuk meg a feladatot (newton.m):

```
> function [x2, i] = newton(f, df, x0, delta, N)
>     x1 = x0;
>     x2 = x1 - f(x1)/df(x1); % első közelítés
>     i = 1; % iterációk száma
>     while abs(f(x2))>delta && i<=N
>         x1 = x2;
>         x2 = x1 - f(x1)/df(x1);
>         i = i + 1;
>     end
> end
```

[7] Otthoni átnézésre

CSATORNA MÉRETEZÉS NEWTON MÓDSZERREL

Határozzuk meg az *f* függvény *h* szerinti deriváltját szimbolikusan! Ehhez a **diff** parancsot használhatjuk, miután szimbolikussá alakítottuk az *f* függvényt a **sym** paranccsal. (Octave esetében be kell tölteni a symbolic csomagot, ha már korábban feltelepítettük: **pkg load symbolic**, lásd első gyakorlat kiegészítése).

```
> % Szimbolikus deriválás
> s=diff(sym(f),'h')
> % s = (10*2^(1/2)*(2*h)^(2/3))/(3*(2*h + 2)^(2/3)) -
  (4*2^(1/2)*(2*h)^(5/3))/(3*(2*h + 2)^(5/3))
```

Az eredmény azonban nem egy függvény hanem egy szimbolikus változó, ebbe nem lehet konkrét értékeket behelyettesíteni. Definiáljuk függvényként, hogy később dolgozni tudjunk vele! Ehhez az egyik megoldás, hogy az eredményt egyszerűen másoljuk be a függvény definíció után, vagy használhatjuk a **matlabFunction** parancsot is, ami szimbolikus kifejezésből függvényt állít elő.

```
> % szimbolikus kifejezésből függvény: definíció majd CTRL+C,CTRL+V
> df = @(h) (10*2^(1/2)*(2*h)^(2/3))/(3*(2*h + 2)^(2/3)) -
  (4*2^(1/2)*(2*h)^(5/3))/(3*(2*h + 2)^(5/3))
> % más megoldás: matlabFunction használata
> df = matlabFunction(s)
```

Oldjuk meg Newton módszerrel is!

```
> % megoldás Newton módszerrel
> [xnew, inew]= newton(f, df, 1.6, 1e-9, 100)
> % xnew = 1.4929, inew = 3
```

A zérushely természetesen ugyanaz, mint korábban. Látjuk, hogy még a megoldástól távolabbi 1.6 méteres kezdőértéket választva is gyorsabban konvergált az eljárás, mint bármelyik zárt intervallum módszer, mindössze 3 iteráció elegendő volt. Hátránya a módszernek, hogy meg kellett határozni a függvény deriváltját is.

BEÉPÍTETT MATLAB FÜGGVÉNY - FZERO

Az előző algoritmusok nagyon jól bemutatták a numerikus számítások alapjait. Nem mindegy milyen algoritmust adunk meg, milyen kezdőértéket, gyorsabban vagy lassabban kapunk eredményt egy iterációs eljárás végén (ha egyáltalán kapunk és konvergál a módszer). A nemlineáris egyenlet gyökeinek megkeresése egy alapfeladat, ami nagyon sokszor előjön a számításaink során, természetesen a Matlabnak is van saját beépített függvénye (**fzero**), ami több módszer kombinálásból adódik össze, és Brent-Dekker algoritmunak hívják.

BRENT MÓDSZER (INVERSE QUADRATIC INTERPOLATION)

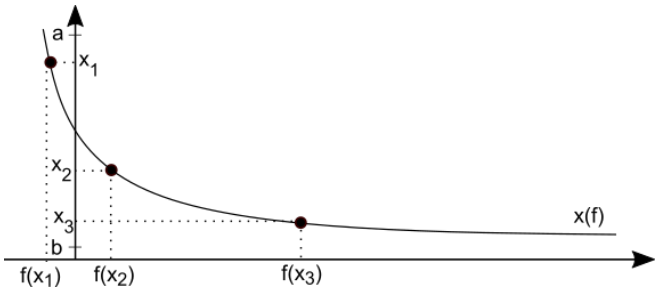
Brent-Dekker módszernek is nevezik, mivel Dekker korábbi módszerét fejlesztette tovább Brent. Hatékony és robosztus módszer, amely kombinálja az intervallum felezést, a húr módszert és az inverz kvadratikus interpolációt. Ezt használja a beépített **fzero** függvény is.

Az inverz kvadratikus interpolációhoz 3 pontot kell megadnunk az [a, b] intervallumban. A pontok koordinátáit felcserélt $(f(x_i), x_i)$ sorrendben adjuk meg, tehát most a független változó lesz a függvényérték. Erre a 3 pontra illesztünk egy másodfokú polinomot.

$$x(f) = \alpha_2 \cdot f^2 + \alpha_1 \cdot f + \alpha_0$$

A polinom illesztésekor meghatározzuk $\alpha_2, \alpha_1, \alpha_0$ értékét, majd, ha behelyettesítjük az f=0 értéket, rögtön megkapjuk az x=c helyet, ahol a függvény metszi a tengelyt. f=0 esetén:

$$c = x(0) = \alpha_0$$



CSATONA MÉRETEZÉS FZERO ALKALMAZÁSÁVAL

Oldjuk meg a csatorna méretezési feladatot az **fzero** parancsot használva! Az **fzero**-t meg lehet hívni egy illetve két kezdőérték megadásával is. Két kezdőérték esetén olyan intervallumot kell megadni, ahol van megoldás, tehát a függvény előjelet vált (zárt intervallum módszer).

```
> % meghívás két kezdőértékkel
> x = fzero(f,[1.4, 1.6]) % x = 1.4929
> % meghívás egy kezdőértékkel
> x = fzero(f,1.6) % x = 1.4929
```

Amennyiben egy kezdőértékkel hívjuk meg, a függvény azzal kezdi, hogy az egy kezdőérték körül keres egy olyan intervallumot, ahol a végeken eltérőek az előjelek és utána oldja meg a feladatot zárt intervallum módszerrel a Brent-Dekker algoritmust használva. Az egyes iterációs lépéseket ki is írathatjuk az **optimset** változó használatával. Megadhatjuk, hogy kijelezze-e az iterációkat (**'Display','iter'**), illetve mi legyen a számítás pontossága (**'TolFun'**, vagy **'TolX'** használatával a függvényérték vagy a független változó toleranciája).

```
> % meghívás egy kezdőértékkel, kiegészítő opciókkal
> x = fzero(f,1.6, optimset('Display','iter','TolFun',1e-9))
> % Search for an interval around 1.6 containing a sign change:
> %  Func-count    a          f(a)             b          f(b)
   Procedure
> %     1            1.6      0.274131           1.6      0.274131
   initial interval
> %     3         1.55475     0.157999        1.64525     0.390694
   search
> %     5           1.536     0.110027          1.664     0.439097
   search
> %     7         1.50949    0.0423222        1.69051     0.507665
   search
> %     8           1.472   -0.0531432        1.69051     0.507665
   search
> %
> % Search for a zero in the interval [1.472, 1.69051]:
> %  Func-count    x          f(x)             Procedure
> %     8           1.472   -0.0531432        initial
> %     9         1.49271  -0.000458365       interpolation
> %    10         1.49289   4.30326e-08       interpolation
> %    11         1.49289   -3.6593e-13       interpolation
```

```
>  %    12          1.49289    4.44089e-16        interpolation
>  %    13          1.49289    4.44089e-16        interpolation
>  %
>  % Zero found in the interval [1.472, 1.69051]
>  % x = 1.4929
```

EGYVÁLTOZÓS ALGEBRAI POLINOM GYÖKEI

Gyakran előforduló feladat, hogy a nemlineáris egyenlet, aminek a gyökeit keressük algebrai polinom, azaz x-nek csak egész kitevőjű hatványai szerepelnek benne.

Egy polinom általános alakja:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

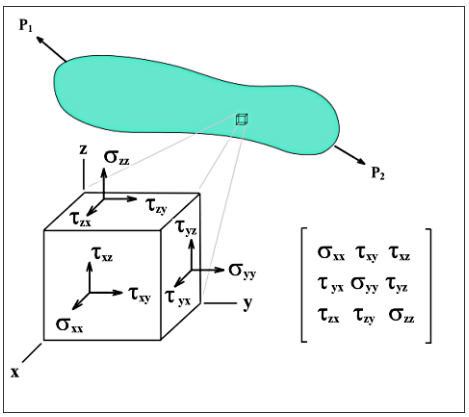
Az $a_n, a_{n-1}, \cdots, a_1, a_0$ együtthatók valós számok, n pedig egy nem negatív egész szám, a polinom fokszáma.

Ezeknek a gyökeire több parancs is van a Matlab-ban, amivel kezdőérték megadása nélkül is meghatározhatjuk az összes gyököt egyszerre. Az egyik a **roots** parancs numerikusan határozza meg egy egyváltozós polinom gyökeit, csak a polinom együtthatóit kell neki megadni egy vektorban, a legmagasabb fokú tagtól visszafelé. Pl. $3x^3 - 4x^2 - 23 = 0$ polinom együtthatói: [3, -4, 0, -23]. Egy másik parancs a **solve** szimbolikusan oldja meg a feladatot és szolgáltat egzakt értékeket a feladatra.

Nézzünk meg egy olyan példát szilárdságtanból, ami algebrai polinom gyökeinek megkeresésére vezethető vissza! Ilyen példa például a sajátérték feladatoknál a karakterisztikus polinom gyökeinek meghatározása.

FŐFESZÜLTSÉGEK MEGHATÁROZÁSA, SAJÁTÉRTÉK FELADAT MEGOLDÁSA

Gyakori feladat a mechanikában a feszültségtenzorból a főfeszültségek, feszültségi főtengelyek meghatározása! Egy P pont feszültség állapotát szemléltethetjük az alábbi ábrával[8]. A feszültségtenzor: *F*. A főtengelyek azok a tengelyek, ahol csak normálfeszültség ébred, nyírófeszültségek nem.



$$F_{(1,2,3)} = \begin{pmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{pmatrix}$$, ahol $\sigma_1 \geq \sigma_2 \geq \sigma_3$.

A főtengely probléma matematikai szempontból sajátérték feladatnak tekinthető. Az *e* főirányban lévő ρ_e feszültségvektor felírható a σ_e főfeszültség és az *e* főirány egységvektor szorzatával: $\rho_e = \sigma_e \cdot e$, illetve az *F* feszültségtenzor *e* irányú vetületével: $\rho_e = F \cdot e$. A kettőt egyenlővé téve (az elsőt egységmátrixszal szorozva): $F \cdot e = \sigma_e \cdot I \cdot e$, levezethető az alábbi képlet: $(F - \sigma_e \cdot I) \cdot e = 0$

[8] CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=591829

ahol I az egységmátrix. A fenti egyenlet egy homogén lineáris egyenletrendszer, ahol a triviálistól $(e = 0)$ különböző megoldást keressük, vagyis ahol a $det(F - \sigma_e \cdot I) = 0$. A meghatározott e vektorok lesznek a sajátvektorok, a főtengelyek, a σ_e értékek pedig a sajátértékek, a főfeszültségek. Írjuk fel a determinánst, ennek a kifejtése lesz a karakterisztikus egyenlet.

$$det(F - \sigma_e \cdot I) = \begin{vmatrix} (\sigma_x - \sigma_e) & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & (\sigma_y - \sigma_e) & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & (\sigma_z - \sigma_e) \end{vmatrix} = 0$$

Legyen most az F feszültségtenzor: $F = \begin{bmatrix} 50 & 20 & -40 \\ 20 & 80 & -30 \\ -40 & -30 & -20 \end{bmatrix} MPa$

Keressük meg az ehhez tartozó főfeszültségeket, oldjuk meg a $det(F - \sigma_e \cdot I) = 0$ egyenletet! A determinánst (**det** parancs) kifejtve a karakterisztikus egyenlet a következő lesz (*fo*-vel jelölve a főfeszültségeket, amik tulajdonképpen a sajátértékek), szimbolikus számítások segítségével:

```
> F = [50, 20, -40; 20, 80, -30; -40, -30, -20];
> syms fo
> eq = det(F-eye(3)*fo) % eq = - fo^3 + 110*fo^2 + 1500*fo - 197000
```

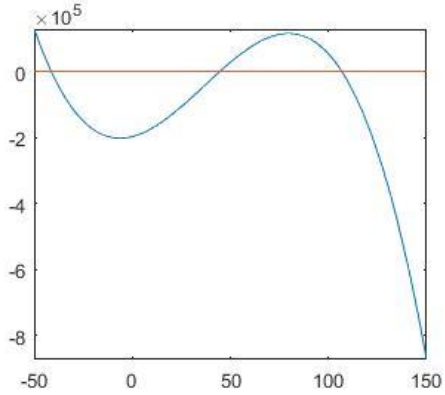
Ez egy harmadfokú algebrai polinom, aminek a gyökei adják a sajátértékeket:

$$eq = -{\sigma_e}^3 + 110\,{\sigma_e}^2 + 1500\,\sigma_e - 197000 = 0$$

Alakítsuk vissza függvénnyé a szimbolikus kifejezést és ábrázoljuk a függvényt a [-50,150] intervallumon!

```
> eq1 = matlabFunction(eq)
> %  @(fo)fo.*1.5e3+fo.^2.*1.1e2-fo.^3-1.97e5
> figure(3); fplot(eq1,[-50,150])
> % y=0 vonal berajzolása a [50,150] intervallumon
> hold on; plot([-50,150],[0,0])
```

Megkereshetjük a harmadfokú polinom gyökeit pl. az **fzero** függvénnyel. Az ábrán látszik, hogy most 3 sajátérték van, tehát az **fzero**-t 3 kezdőértékkel kell meghívjuk, hogy minden megoldást megtaláljunk. Kezdőértékeket az ábrából vehetünk, ahol közelítőleg metszi az x tengelyt a függvény! Legyenek ezek x=120, 50, -40!




```
> % Megoldás fzero-val
> fo1 = fzero(eq1,120) % 106.7674
> fo2 = fzero(eq1,50) %  44.6017
> fo3 = fzero(eq1,-40) % -41.3691
```

Az eredmény a három sajátérték, vagyis a három főfeszültség: $\sigma_1 = 106.7674, \sigma_2 = 44.6017,\ \sigma_3 = -41.36$. A három gyök megtalálásához 3 függvényhívás kellett. Polinomokra azonban vannak olyan kidolgozott eljárások, amelyek nem egy lépésben megadják az összes megoldást és még kezdőértéket sem kell megadni hozzájuk. Az egyik ilyen a szimbolikus kifejezésekre működő **solve** parancs. Oldjuk meg a **solve** használatával az *'eq = - fo^3 + 110*fo^2 + 1500*fo – 197000'* egyenletet!

```
> sol1 = solve(eq)
> %  root(z^3 - 110*z^2 - 1500*z + 197000, z, 1)
> %  root(z^3 - 110*z^2 - 1500*z + 197000, z, 2)
> %  root(z^3 - 110*z^2 - 1500*z + 197000, z, 3)
> sol2 = double(sol1)
> %     1.0e+02 *
> %    1.0677 + 0.0000i
> %   -0.4137 + 0.0000i
> %    0.4460 - 0.0000i
> sol3 = real(double(sol1))
> %   106.7674
> %   -41.3691
> %    44.6017
```

A **solve** parancs eredményeként nem konkrét számokat kapunk, hanem szimbolikus kifejezéseket. Ezeket számmá kell utána alakítanunk a **double** paranccsal. Az eredményül kapott számnak most azonban vannak numerikusan elhanyagolható kicsi komplex részei is, ezeket is elhagyhatjuk, ha csak a valós részt használjuk a **real** paranccsal. Egy parancsban is megadhatjuk az egészet, és megkapjuk mindhárom megoldást:

```
> sol = real(double(solve(eq)))
```

A másik algebrai polinomok esetében használható parancs a **roots**, ami numerikusan oldja meg az egyenletet. Ez is egy lépésben megadja az összes megoldást és itt sem kell kezdőértéket megadni. Itt viszont ki kell gyűjteni a polinom együtthatóit egy vektorba a legmagasabb fokú tagtól kezdve visszafelé a konstans tagig. Az $eq = -{\sigma_e}^3 + 110\,{\sigma_e}^2 + 1500\,\sigma_e - 197000$ egyenletnél létrehozhatunk kézzel is egy vektort, ami az együtthatókat tartalmazza:

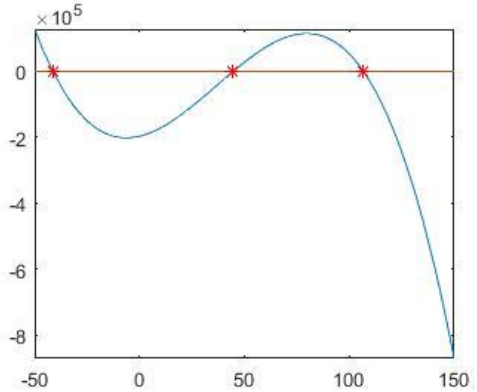
```
> % Együtthatók megadása vektorba írással
> c = [-1, 110, 1500, -197000]
```

Vagy használhatjuk a **sym2poly** parancsot is, ami egy szimbolikus polinomból kigyűjti az együtthatókat:

```
> % más megoldás
> c = sym2poly(eq)
> % c = [-1  110 1500 -197000]
```

Oldjuk meg **roots** paranccsal, és az eredményeket rajzoljuk be az ábrába!

```
> FO = roots(c)
> %   106.7674
> %   -41.3691
> %    44.6017
> plot(FO,eq1(FO),'r*')
```



Mint láttuk itt sem kellett megadni kezdőértékeket és megkaptuk egyszerre az összes gyököt. A megkapott főfeszültségekhez természetesen meg lehetne határozni a főtengelyek irányait is, ha visszahelyettesítenénk a főfeszültségeket az eredeti egyenletrendszerbe. Ki is használhatjuk azonban a Matlab rengeteg beépített függvényét, természetesen a sajátérték, sajátvektor problémára is van megoldás, mivel ez is egy nagyon gyakori feladat, méghozzá az **eig** parancs.

```
>  [V D]=eig(F)
>  % V =
>  %     0.3647    0.7722    0.5203
>  %     0.1664   -0.6038    0.7795
>  %     0.9162   -0.1977   -0.3487
>  %
>  % D =
>  %   -41.3691         0         0
>  %          0   44.6017         0
>  %          0         0  106.7674
```

Itt két kimenettel hívtuk az **eig** parancsot, és egyszerre megkaptuk az összes sajátértéket (D mátrix átlójában) és a hozzájuk tartozó sajátvektorokat is (V mátrix oszlopai)!

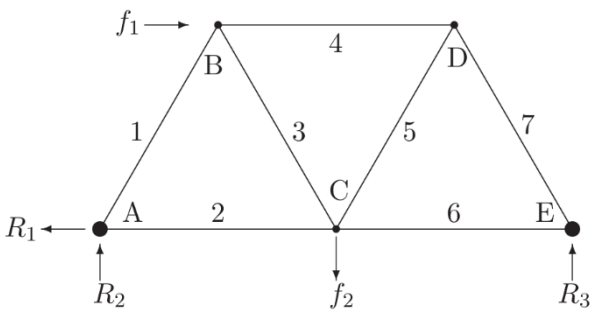
A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

set	-	Grafikus elem tulajdonságainak beállítása (pl. Color, LineWidth)
and(felt1, felt2), felt1 && felt2,	-	Logikai ÉS
diff	-	Szimbolikus deriválás
sym	-	Kifejezések, változók szimbolikussá alakítása
fzero	-	Egyváltozós egyenlet gyökeinek megkeresése numerikusan
det	-	Mátrix determinánsa
solve	-	Algebrai polinom gyökei szimbolikusan
roots	-	Algebrai polinom gyökei numerikusan
double	-	Szimbolikus kifejezésként megadott szám lebegőpontos számmá alakítása
real	-	Képzetes szám valós része
sym2poly	-	Szimbolikusan megadott algebrai polinom együtthatóinak kigyűjtése egy vekorba
eig	-	Mátrix sajátértékeinek, sajátvektorainak meghatározása

5. LINEÁRIS EGYENLETRENDSZEREK 1.

A lineáris egyenletrendszerek kiemelt jelentőséggel bírnak az alkalmazott matematika és általában a numerikus matematika területén. Lineáris egyenletrendszer megoldására vezet a műszaki alkalmazásokban az a gyakorlat, hogy a mérnök többnyire lineáris fizikai modelleket alkalmaz. Nem csupán azért mert tudatában van annak, hogy a matematikai modellt így könnyebb lesz megoldani, hanem azért is mert sok fizikai jelenséget egy adott állapot környezetében valóban lineárisnak tekinthetünk, például a Hook-törvény. A numerikus matematikában, mint majd később látni fogjuk, nagyon sok numerikus módszer alkalmazása visszavezethető lineáris egyenletrendszer megoldására, például az interpoláció problémája. A lineáris egyenletrendszerek kezelésében alapvető fontosságú lesz a mátrix algebra alkalmazása, amely kimondottan kedvez a numerikus számítógépi megoldások elvégzésének.

Nézzünk most egy példát statika témaköréből. Vizsgáljuk meg az ábrán látható egyenlő oldalú háromszögekből álló rácsos tartóra ható erőket! A csomóponti módszert használva számítsuk ki a rúderőket! Használjuk a $\cos(60°) = 0.5$ értéket és a $\sin(60°)$ helyett a $\frac{\sqrt{3}}{2} \approx 0.8660$ közelítést ($f_1 = 1000N$ és $f_2 = 5000N$).



Az erők vetületi összegének és tetszőleges pontra számított nyomatékösszegének zérust kell eredményezni. Vetületi egyensúlyi egyenletek alapján: $R_1 - f_1 = 0$. Az *E* pontra felírt nyomatéki egyensúlyi egyenlet alapján $2\,R_2 + 0.866\,f_1 - f_2 = 0$. E kettőből kifejezhető R_1, R_2 reakció: $R_1 = f_1,\ \ R_2 = -0.433\,f_1 + 0.5\,f_2$. A csomópontokra ható erőket vizsgálva a következő lineáris egyenletrendszert írhatjuk fel, ahol T_i-vel jelöltük a rúderőket:

A jelű csomópont, x irányú erők: $0.5\,T_1 + T_2 = R_1 = f_1$

A jelű csomópont, y irányú erők: $0.866\,T_1 = -R_2 = 0.433\,f_1 - 0.5\,f_2$

B jelű csomópont, x irányú erők: $-0.5\,T_1 + 0.5\,T_3 + T_4 = -f_1$

B jelű csomópont, y irányú erők: $0.866\,T_1 + 0.866\,T_3 = 0$

C jelű csomópont, x irányú erők: $-T_2 - 0.5\,T_3 + 0.5\,T_5 + T_6 = 0$

C jelű csomópont, y irányú erők: $0.866\,T_3 + 0.866\,T_5 = f_2$

D jelű csomópont, x irányú erők: $-T_4 - 0.5\,T_5 + 0.5\,T_7 = 0$

A fenti egyenletrendszer kellő idővel és türelemmel megoldható számítógép nélkül is, sorban eliminálva az egyes változókat, majd visszahelyettesítve őket. Nyilvánvaló azonban, hogy számítógéppel megoldva lényegesen egyszerűbb a feladat. A most következőkben megnézzük, hogy hogyan oldjunk meg számítógéppel lineáris egyenletrendszereket. A megoldáshoz a legfontosabb, hogy realizáljuk, hogy a lineáris egyenletek a mátrixokkal egyenértékűek, amelyek számokat és nem változókat

tartalmaznak. A fenti lineáris egyenletrendszer mátrixos alakban $A\,x = b$ felírva a következő lesz:

$$A = \begin{pmatrix} 0.5 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.866 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.5 & 0 & 0.5 & 1 & 0 & 0 & 0 \\ 0.866 & 0 & 0.866 & 0 & 0 & 0 & 0 \\ 0 & -1 & -0.5 & 0 & 0.5 & 1 & 0 \\ 0 & 0 & 0.866 & 0 & 0.866 & 0 & 0 \\ 0 & 0 & 0 & -1 & -0.5 & 0 & 0.5 \end{pmatrix}, b = \begin{pmatrix} f_1 \\ 0.433\,f_1 - 0.5\,f_2 \\ -f_1 \\ 0 \\ 0 \\ f_2 \\ 0 \end{pmatrix}$$

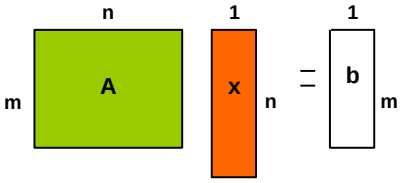
Azt se felejtsük el, hogy a gyakorlatban a mérnököknek igen nagyméretű mátrixokkal is dolgozni kell. Sokszor egy algoritmus, ami egy 2x2 vagy 3x3-as mátrixhoz megfelelő, alkalmatlan lehet pl. egy 2000x2000-es mátrixhoz, ezért az algoritmusok hatékonyságára is oda kell figyelnünk!

MEGOLDÁSOK LÉTEZÉSE ÉS EGYÉRTELMŰSÉGE

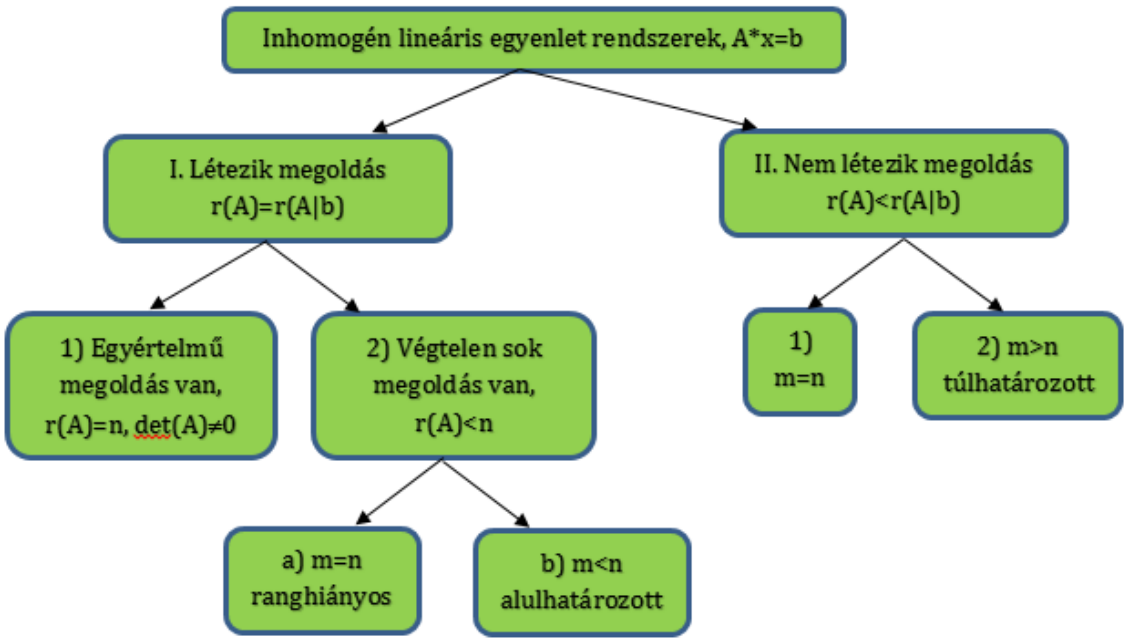
Nézzük meg először a lineáris egyenletrendszerek típusait megoldhatóság szerint! Az egyenletrendszer általános alakja mátrix formában felírva:

$$A \cdot x = b$$

ahol *A m*n*-es alakmátrix, *x* a keresett változók *nx1*-es oszlopvektora és *b* egy *mx1*-es oszlopvektor.



Beszélhetünk inhomogén (*b≠0*) és homogén (*b=0)* egyenletrendszerekről. A homogén esetben akkor van a triviális *x=0*-tól eltérő megoldás, ha az A mátrix determinánsa egyenlő nullával: **det**(A)=0. Erre láttunk példát az előző gyakorlaton a sajátérték problémánál. Inhomogén esetben a megoldások száma szerint a következő módon csoportosíthatjuk a megoldásokat:



4. ÁBRA MEGOLDÁS LÉTEZÉSE ÉS EGYÉRTELMŰSÉGE

LÉTEZIK ÉS EGYÉRTELMŰ A MEGOLDÁS

Létezik a megoldás, ha az A mátrix rangja (lineárisan független oszlopvektorainak száma, *r(A)*) megegyezik az A mátrixnak a b oszlopvektorral kibővített mátrix rangjával (*r(A|b)*), azaz r(A)=r(A|b). Matlab-ban a mátrix rangja a **rank** paranccsal számítható:

```
> rank(A), rank([A b])
```

Egyértelmű megoldás van, ha r(A)=r(A|b)=n illetve det(A)≠0, azaz az *A* mátrix rangja megegyezik a kibővített mátrix rangjával és ez a rang teljes (megegyezik az oszlopok számával is). A mátrix oszlopvektorai függetlenek. Ha a rang kisebb lenne, mint az oszlopok száma, akkor végtelen sok megoldás lenne. A megoldás:

$$x = A^{-1} \cdot b.$$

Vizsgáljuk meg, Matlabot használva, hogy a fent megadott rácsos tartónál van-e megoldása a feladatnak és egyértelmű-e? Az A és a b mátrix kézzel is bevihető lenne az alábbi módon:

```
> A = [0.5 1 0 0 0 0 0; 0.866 0 0 0 0 0 0; -0.5 0 0.5 1 0 0 0;
>    0.866 0 0.866 0 0 0 0; 0 -1 -0.5 0 0.5 1 0; 0 0 0.866 0 0.866 0 0;
>    0 0 0 -1 -0.5 0 0.5]
>  f1 = 1000; f2 = 5000;
>  b = [f1; 0.433*f1-0.5*f2; -f1; 0; 0; f2; 0]
```

Azonban az A és b értékei el is vannak mentve a **racsos.txt** fájlban. Az elírások elkerülése végett most töltsük be ezt a fájlt, majd válasszuk szét az A mátrixot és a b vektort, ami az utolsó oszlopban van!

```
> Ab = load('racsos.txt')
> A = Ab(:,1:end-1)
> b = Ab(:,end)
```

Nézzük meg, hogy van-e egyértelmű megoldása a feladatnak!

```
> size(A,2) % n=7 oszlop van
> rank(A), rank([A,b]) % 7=7,  van megoldás, egyértelmű: n=7
```

Mivel az *A* mátrix rangja és a *b* vektorral kibővített mátrix rangja is 7, így van megoldása a feladatnak. A megoldás egyértelmű, mivel ez egy 7x7-es négyzetes mátrix és a rang megegyezik az oszlopok számával is. Ellenőrizzük, hogy a determináns nem nulla-e?

```
> det(A) % =-0.3247 – egyértelmű megoldás van
```

Írjunk egy programot, ami eldönti, hogy van-e egyértelmű megoldása a feladatnak, vagy nincs, és kiírja a választ a parancssorba!

```
> if rank(A)==rank([A,b]) && det(A)~=0
>     disp('Egyertelmű megoldás van')
> else disp('Nincs egyértelmű megoldás')
> end
```

GAUSS-ELIMINÁCIÓ, HÁROMSZÖG MÁTRIXOK, VISSZAHELYETTESÍTÉS, PERMUTÁCIÓ

Azt már tudjuk, hogy a feladatnak van egyértelmű megoldása, már csak az a kérdés, ezt hogyan kapjuk meg? Mielőtt nekiállnánk a rácsos tartó rúderőinek számításához, nézzünk először egy egyszerűbb példát!

$$x_1 - 2x_2 + 3x_3 = 4$$
$$2x_1 - 5x_2 + 12x_3 = 15$$
$$2x_2 - 10x_3 = -10$$

Ezt felírhatjuk mátrixos alakban, vagy még egyszerűbben a kibővített mátrix alakjában:

$\begin{pmatrix} 1 & -2 & 3 \\ 2 & -5 & 12 \\ 0 & 2 & -10 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 15 \\ -10 \end{pmatrix}$, kibővített mátrixként: $\left(\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 2 & -5 & 12 & 15 \\ 0 & 2 & -10 & -10 \end{array}\right)$

A kibővített mátrixos alak előnye, hogy csak számokat tartalmaz, változókat nem, ezekkel könnyebben boldogulnak a számítógépek. Ismételjük át a matematikában korábban már bizonyára megismert Gauss-eliminációt!

A Gauss-elimináció célja, hogy felső háromszög mátrixba alakítsuk az alakmátrixot, ami után már egyszerű visszahelyettesítéssel megoldható az egyenletrendszer. Nézzük a fenti példát! Ahhoz, hogy felső háromszög mátrix legyen, a főátló alatt csupa nulla elem lehet csak. Nullázzuk ki a második sor első elemét (2,1), úgy, hogy kivonjuk az első sor kétszeresét a második sorból:

$$\left(\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 2 & -5 & 12 & 15 \\ 0 & 2 & -10 & -10 \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 0 & -1 & 6 & 7 \\ 0 & 2 & -10 & -10 \end{array}\right)$$

A harmadik sor első eleme (3,1) már nulla, itt nem kell semmit eliminálni, a 3. sor 2. eleme (3,2) azonban nem nulla. Ezt úgy tűntethetjük el, ha a harmadik sorból kivonjuk a második sor -2 szeresét.

$$\rightarrow \left(\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 0 & -1 & 6 & 7 \\ 0 & 0 & 2 & 4 \end{array}\right) \rightarrow \begin{array}{ll} x_1 - 10 + 6 = 4 & \rightarrow x_1 = 8 \\ -x_2 + 12 = 7 & \rightarrow x_2 = 5 \\ 2x_3 = 4 & \rightarrow x_3 = 2 \end{array} \uparrow$$

A fenti mátrix már egy felső háromszögmátrix, ami könnyen megoldható. Ne felejtsük el, hogy a mátrix minden sora megfelel egy-egy egyenletnek. Az utolsó sor: $2x_3 = 4$, aminek nagyon egyszerű a megoldása: $x_3 = 2$. A második sor megfelel a $-x_2 + 6x_3 = 7$ egyenletnek, visszahelyettesítve x_3 már ismert értékét a $-x_2 + 12 = 7$ egyenletet kell megoldani, amiből $x_2 = 5$ adódik. Miután ismerjük x_2 és x_3 értékét, az első sor $x_1 - 10 + 6 = 4$ egyenletre egyszerűsödik, amiből $x_1 = 8$ adódik. Az tette lehetővé, hogy ilyen egyszerűen visszahelyettesítéssel megoldjuk az egyenletrendszert, hogy sikerült felső háromszög mátrixszá alakítani az alakmátrixot. Ellenőrizzük Matlab-ban:

```
> A = [1 -2 3; 2 -5 12; 0 2 -10], b = [4; 15; -10]
> x = inv(A)*b % x = [8; 5; 2]
```

A Gauss-elimináció hatékonyságának növeléséhez szükségünk lehet a sorok felcserélésére, permutációjára (pivoting), hogy csökkentsük a numerikus (kerekítési) hibák hatását. Ilyenkor úgy cseréljük fel a sorokat, hogy az adott oszlopban a legnagyobb abszolút értékű elemet használhassuk a többi elem eliminálására. Egy nagyobb mátrixban minden oszlop elkészülte után célszerű megnézni, hogy a következő oszlopban melyik a legnagyobb abszolút értékű elem, és a szerint felcserélni a sorokat. Az előző példában, a numerikus pontosság növelése érdekében célszerű lett volna felcserélni először az első és a második sort, hiszen $|2| > |1|$, és a (2,1) elem kinullázása után a 2. és 3. sort is célszerű felcserélni.

LU FELBONTÁS

Sok esetben ugyanazt az **Ax = b** lineáris egyenletrendszert több **b** vektorra is meg kell oldanunk. Gondoljunk itt például egy híd próbaterhelésére, amit több különböző terheléssel is ellátnak, nem csak eggyel. Itt a különböző terhek jelentik a különböző **b** vektorokat. A fenti példából látható, hogy a munka nagyobb részét az **A** mátrixon végeztük. Ha több **b** vektorra is meg kell oldanunk ugyanazt az egyenletrendszert és **A** mátrix nagy, akkor szeretnénk elkerülni, hogy meg kelljen ismételni a Gauss-elimináció minden lépését az **A** mátrixra minden **b** esetében. Ezt megtehetjük az LU felbontást használva, ami tulajdonképpen eltárolja a Gauss-elimináció műveleteit is.

A fenti példát használva próbáljuk meg eltárolni, hogy az egyes lépésekben az adott sor hányszorosát kellett kivonni az aktuális sorból, hogy eliminájuk a megfelelő elemet. Tegyük ezeket zárójelbe, hogy megkülönböztessük a mátrix többi elemétől! Először a második sor első elemét (2,1) eliminátuk, úgy, hogy kivontuk az első sor kétszeresét a második sorból (most csak az **A** mátrixot nézzük **b** nélkül):

$$\begin{pmatrix} 1 & -2 & 3 \\ (2) & -1 & 6 \\ 0 & 2 & -10 \end{pmatrix}$$

A harmadik sor első eleme (3,1) már eleve nulla volt, itt nem kellett semmit eliminálni, írjunk ide (0)-t, a 3. sor 2. elemét (3,2) úgy tűntettük el, hogy a harmadik sorból kivontuk a második sor -2 szeresét:

$$\begin{pmatrix} 1 & -2 & 3 \\ (2) & -1 & 6 \\ (0) & (-2) & 2 \end{pmatrix}$$

Legyen U az előbb is megkapott felső háromszög mátrix (**U**pper trianglar matrix) és L egy alsó háromszög mátrix (**L**ower trianglar matrix), aminek az elemei legyenek az eltárolt műveletek és a főátlójában 1-esek:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix} \text{ és } U = \begin{pmatrix} 1 & -2 & 3 \\ 0 & -1 & 6 \\ 0 & 0 & 2 \end{pmatrix}$$

Ez az úgynevezett LU felbontás. Ennek a felbontásnak megvan az a tulajdonsága, hogy $L \cdot U = A$:

$$L \cdot U = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & -2 & 3 \\ 0 & -1 & 6 \\ 0 & 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & -2 & 3 \\ 2 & -5 & 12 \\ 0 & 2 & -10 \end{pmatrix} = A$$

Ellenőrizzük le Matlab-ban:

```
> L = [1 0 0; 2 1 0; 0 -2 1]
> U = [1 -2 3; 0 -1 6; 0 0 2]
> L*U-A
> norm(L*U-A) % 0 (ellenőrzés)
```

Láthatjuk, hogy az **A** mátrix előáll az **L** és az **U** mátrixok szorzataként. Itt **L** egy alsó, **U** egy felső háromszög mátrix. Amikor egy mátrix előállítható egyszerűbb mátrixok szorzataként, azt mátrix felbontásnak (decomposition) nevezzük. Ez éppen az LU felbontás. Ellenőrzésként általában az eltérésvektor/mátrix normáját ('hosszát') szokás megadni, ezt a **norm** paranccsal tehetjük.

MEGOLDÁS LU FELBONTÁSSAL

Ha a sorok felcserélését, permutációját is belevesszük, akkor **A** mátrix LU felbontása 3 mátrixból áll (**L**, **U**, **P** – permutáció mátrix):

$$P \cdot A = L \cdot U$$

A **P** permutáció mátrix egy egységmátrix felcserélt sorokkal, amelyekben ugyanazok a sorok vannak felcserélve, mint az **A** mátrixban. Pl. a következő **P** permutációs mátrix a második és a harmadik sor felcserélését jelenti: $P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$.

Matlab-ban LU felbontást permutációval együtt a következő paranccsal állíthatunk elő:

```
> [L U P] = lu(A)
```

Ha ezt szeretnénk az $A \cdot x = b$ lineáris egyenletrendszer megoldására használni, akkor először szorozzuk meg mind a két oldalt a permutációs mátrixszal:

$$P \cdot A \cdot x = P \cdot b \equiv d$$

Majd helyettesítsük be $P \cdot A$ helyére $L \cdot U$-t:

$$L \cdot U \cdot x = d$$

Ezek után csupán két egyszerű visszahelyettesítéses problémát kell megoldanunk, két háromszög mátrixszal, legyen $y = U \cdot x$.

- $L \cdot y = d$ (ahol L alsó háromszög mátrix és $d = P \cdot b$)
- $U \cdot x = y$ (ahol U felső háromszög mátrix)

RÁCSOS TARTÓ RÚDERŐI LU FELBONTÁSSAL

Oldjuk meg az előbbi rácsos tartó egyenletrendszerét is LU felbontással! A megoldás során használjunk olyan algoritmust, ahol figyelembe tudjuk venni az L és U mátrixok speciális voltát (alsó ill. felső háromszög mátrix)!

Először töltsük be újra a mátrixokat, majd állítsuk elő az LU felbontást!

```
> Ab = load('racsos.txt'); A = Ab(:,1:end-1); b = Ab(:,end);
> [L U P] = lu(A)
```

Most jön az $L \cdot y = d$ és az $U \cdot x = y$ megoldása. Ez tulajdonképpen két egyszerű visszahelyettesítést jelent, mivel tudjuk, hogy ezek a mátrixok alsó/felső háromszögmátrixok. Ehhez használhatjuk a lineáris egyenletrendszerek megoldásához használható **linsolve** parancsot. A **linsolve** parancsot használva az opcióknál megadható az *A* mátrix néhány tulajdonsága, ha előzetesen ismert (pl. alsó/felső háromszögmátrix, szimmetrikus, pozitív definit), ez lényegesen gyorsíthatja a megoldást. Itt az **LT** a lower triangle – alsó háromszögmátrix, az **UT** az upper triangle – felső háromszögmátrix rövidítése. Ezeket a tulajdonságokat egy struktúra típusban adhatjuk meg, ahol a változónév után ponttal hivatkozhatunk az adott tulajdonságra, hogy igaz-e vagy hamis.

```
> d = P*b;
> opt1.LT=true
> y = linsolve(L,d,opt1);
```

```
> opt2.UT=true
> x = linsolve(U,y,opt2)
> elteres = norm(A*x - b) % ellenőrzés
```

Az LU felbontás egy példa a mátrix felbontásokra, ahol az **A** mátrixot két egyszerűbb mátrixra bontottuk, egy alsó és egy felső háromszög mátrix szorzatára Gauss-eliminációval. Sok másféle mátrix felbontással is találkozhatunk, amelyek különböző esetekben lehetnek hasznosak. A legismertebbek az LU felbontáson kívül a Cholesky felbontás, QR felbontás és az SVD felbontás.

CHOLESKY FELBONTÁS

A Cholesky felbontás és használata lineáris egyenletrendszer megoldására nagyon hasonló az LU felbontásra. A különbség annyi, hogy itt nem egy alsó (**L**) és egy felső (**U**) háromszög mátrixra bontjuk **A** mátrixot, hanem egy darab **L** felső(!) háromszög mátrixunk lesz, és az **A** mátrix ennek és a transzponáltjának (ami már alsó háromszög mátrix) lesz a szorzata: $A = L^T \cdot L$

A megoldás menete $A \cdot x = L^T \cdot L \cdot x = L^T \cdot y = b$ alapján:

- $L^T \cdot y = b$ (ahol L^T alsó háromszög mátrix)
- $L \cdot x = y$ (ahol L felső háromszög mátrix)

Itt csak egy háromszög mátrixunk van, ami lényegesen egyszerűbb, mint a két háromszög mátrix az LU felbontás esetében. Viszont ezt a felbontást nem tudjuk mindig elvégezni, csak, ha néhány fontos feltétel teljesül:

- Az **A** mátrix szimmetrikus, azaz $\mathbf{A^T=A}$ és
- pozitív definit, azaz minden sajátértéke pozitív: $\lambda_i > 0, i = 1, \cdots, n$

Matlab-ban a sajátvektorokat az **`eig`** paranccsal állíthatjuk elő, amit kétféleképp hívhatunk:

```
> E = eig(A) % négyzetes X mátrix sajátértékei
> [V,D] = eig(A) % sajátvektorok V mátrixban (oszlopok), sajátértékek a
  D diagonálmátrixban
```

Vizsgáljuk meg, hogy a rácsos tartó A mátrixa szimmetrikus és pozitív definit-e?

```
> norm(A-A') %  1.5946
> eig(A) % [0.5000; -0.7136; -0.7136; -0.7136; 1.2136; 1.2136; 1.2136]
```

A fenti mátrix se nem szimmetrikus norm(A'-A)≠0, se nem pozitív definit (a sajátértékek nem mind pozitívak), tehát Cholesky felbontással nem oldható meg a feladat (**chol**(A) parancs hibaüzenetet adna). Nézzünk példát egy szimmetrikus, pozitív definit mátrixra! A **pascal** paranccsal előállíthatjuk a binomiális együtthatókat tartalmazó szimmetrikus Pascal mátrixot, a **diag** paranccsal pedig kivehetjük egy mátrix főátlójából az elemeket (vagy egy vektorból csinálhatunk diagonális mátrixot), a **min** paranccsal pedig lekérdezhetjük egy vektor legkisebb elemét (**max** paranccsal pedig a legnagyobbat).

```
> A = pascal(4) % a binomiális együtthatókat tartalmazó Pascal mátrix
> norm(A-A') % szimmetrikus, mivel A = A'
> [V D] = eig(A) % sajátvektorok és sajátértékek előállítása
> min(diag(D)) % pozitív definit, mivel a legkisebb sajátérték is>0
> L = chol(A) % elvégezhető a Cholesky felbontás
> norm(A-L'*L) % ellenőrzés
> b = rand(4,1) % tetszőleges b vektor
```

Az egyenletrendszer megoldása hasonló az LU felbontáshoz (különbség, hogy itt nincs permutáció, tehát d helyett b vektor lesz, L helyére L' és U helyére L kerül):

```
> opt1.LT=true
> y = linsolve(L',b,opt1);
> opt2.UT=true
> x = linsolve(L,y,opt2)
> elteres = norm(A*x - b) % ellenőrzés
```

MATLAB BEÉPÍTETT FÜGGVÉNYEK (INV, LINSOLVE, \ - MLDIVIDE)

Természetesen a Matlab-nak vannak beépített parancsai is az $A \cdot x = b$ egyenletrendszer közvetlen megoldására, anélkül, hogy nekünk kellene lépésenként elvégezni az LU vagy éppen a Cholesky felbontást. Egyértelmű megoldás esetén a beépített parancsok többsége ezeket a felbontásokat használja. Nézzünk rá példákat!

Háromféle lehetőséget fogunk megnézni:

- `x = inv(A)*b`
- `x = linsolve(A,b)`
- `x = A\b % vagy x=mldivide(A,b)`

Nézzük meg mi a különbség az egyes megoldások között a Matlab-ban! Határozzuk meg a beépített parancsokkal is a rácsos tartó rúderőit! Minden megoldásnál nézzük meg a megoldás időigényét is. Tegyük ezt úgy, hogy 10000-szer elvégezzük mindegyik számítást, hogy reálisan összevethetőek legyenek az idők. A Matlab-ban időmérést a **tic-toc** parancsokkal tudunk végrehajtani.

Nézzük először a matematikában hagyományosan felírható megoldást, az inverz számítás segítségével: $x = A^{-1} \cdot b$! Matlabban az **inv** parancs használható inverz számításra.

```
> %% Beépített függvények
> Ab = load('racsos.txt'); A = Ab(:,1:end-1); b = Ab(:,end);
> % Megoldás: x=inv(A)*b
> x1 = inv(A)*b % hagyományos inverz számítás
> tic
> for i=1:10000
>     x1 = inv(A)*b;
> end
> toc % Elapsed time is 0.118122 seconds.
> norm(A*x1-b) % 1.0904e-12
```

Természetesen a futás idő gépenként (sőt futtatásonként) is különbözni fog, de a nagyságrendjük összehasonlítható. A tesztgépen a futásidő 0.118 másodperc volt az inverz számításnál.

Használtuk korábban a **linsolve** parancsot alsó és felső háromszögmátrixokkal megadott egyenletrendszerek megoldására. Ez a parancs nem csak ezekben a speciális esetekben használható, hanem általános esetben is, ilyenkor nem adunk meg opciókat. Nézzük meg a megoldást ebben az esetben!

```
> % Megoldás: x = linsolve(A,b)
> x2 = linsolve(A,b) % nxn: Cholesky vagy LU felbontás
> tic
> for i=1:10000
>     x2 = linsolve(A,b);
```

```
> end
> toc % Elapsed time is 0.082469 seconds.
> norm(A*x2-b) % 6.0157e-13
```

A **linsolve** parancs LU felbontást használ a megoldáshoz, ha az A mátrix négyzetes. Itt a futásidő 0.082 másodperc lett, jóval gyorsabb, mint az előző esetben.

Nézzük meg a harmadik parancsot is, ez az x = A\b megoldás. A **** parancs megegyezik az **mldivide** paranccsal (x=mldivide(A,b)). Ez a függvény több megoldási módszer közül választja ki a legmegfelelőbbet a mátrix vizsgálata után. Négyzetes mátrix (nxn) esetében Cholesky felbontás és LU felbontás közül választ.

```
> % Megoldás: x = A\b
> x3 = A\b % nxn: Cholesky vagy LU felbontás
> x3 = mldivide(A,b) % ua., mint az előző
> tic
> for i=1:10000
>     x3 = A\b;
> end
> toc % Elapsed time is 0.024091 seconds.
> norm(A*x3-b) % 5.5695e-13
```

A futásidő ebben az esetben lett a legjobb, 0.024 másodperc, ami egy nagyságrenddel jobb, mint a hagyományos inverz számítás volt.

Nézzük meg az eredményeket is:

```
1.0e+03 *

   -2.3868   -2.3868   -2.3868
    2.1934    2.1934    2.1934
    2.3868    2.3868    2.3868
   -3.3868   -3.3868   -3.3868
    3.3868    3.3868    3.3868
    1.6934    1.6934    1.6934
   -3.3868   -3.3868   -3.3868
```

Lényegében mindenhol ugyanazt az eredményt kaptuk x-re, kis eltérések vannak, ha megnézzük az eltérésvektor normáját, a legpontosabb eredményt (legközelebb nullához) itt is az A\b parancs adta:

x=inv(A)*b esetén: $\|A \cdot x - b\| = 1.0904\text{e} - 12$

x=linsolve(A,b) esetén: $\|A \cdot x - b\| = 16.0157\text{e} - 13$

x=A\b esetén: $\|A \cdot x - b\| = 15.5695\text{e} - 13$

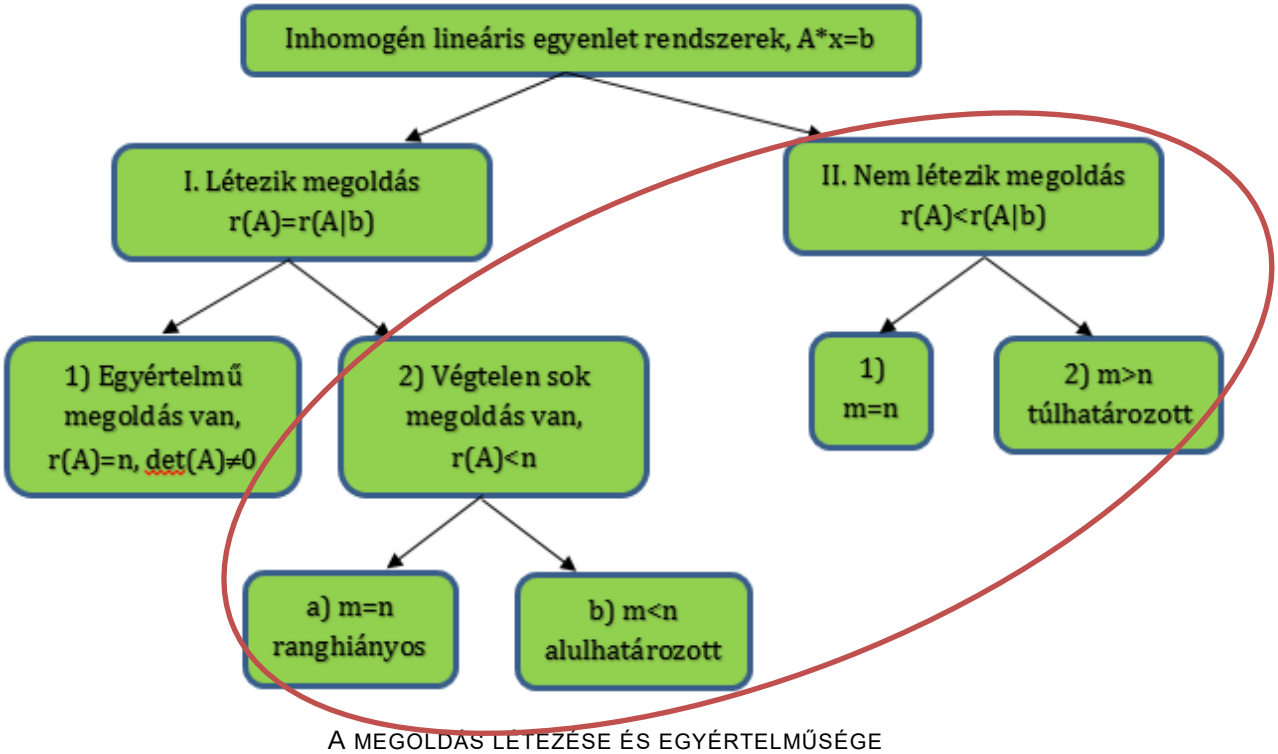
A fentiek alapján általános esetben, négyzetes mátrix és egyértelmű megoldás mellett célszerű az **x=A\b** parancsot használni. Amennyiben előre tudjuk, hogy a mátrix alsó/felső háromszögmátrix, szimmetrikus vagy pozitív definit mátrix, akkor célszerű az **x=linsolve(A,b)** parancsot alkalmazni és megadni az opcióknál a mátrix típusát.

A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

rank	-	Mátrix rangja
lu	-	LU felbontás
linsolve	-	Lineáris egyenletrendszer megoldása kiegészítő opciókkal (pl. alsó/felső háromszögmátrix, szimmetrikus, pozitív definit). Általános négyzetes mátrix esetén LU felbontást használ.
pascal	-	Előállíthatjuk a binomiális együtthatókat tartalmazó szimmetrikus Pascal mátrixot
diag	-	Kivehetjük egy mátrix főátlójából az elemeket vagy egy vektorból csinálhatunk vele diagonális mátrixot
min	-	Egy vektor legkisebb eleme
max	-	Egy vektor legnagyobb eleme
chol	-	Cholesky felbontás
norm	-	Vektor/mátrix normája (’hossza’)
tic, toc	-	Időmérés kezdete, vége
\ vagy mldivide	-	Általános lineáris egyenletrendszer megoldása (négyzetes mátrix esetén LU vagy Cholesky felbontással)

6. LINEÁRIS EGYENLETRENDSZEREK 2.

Eddig csak olyan eseteket vizsgáltunk, ahol létezett és egyértelmű volt az inhomogén lineáris egyenletrendszer megoldása. Többféle algoritmust is láttunk ezeknek a megoldására. A mérnöki gyakorlatban azonban előkerülnek a homogén lineáris egyenletrendszerek is, amire egy példát láttunk a főfeszültségek meghatározásakor, illetve azok az esetek is, amikor az inhomogén rendszernek nincs vagy végtelen sok megoldása van.



A MEGOLDÁS LÉTEZÉSE ÉS EGYÉRTELMŰSÉGE

Megjegyzés: a Matlabban elágazásokkal könnyen megvizsgálhatjuk, hogy van, vagy nincs megoldás és ha van, akkor egyértelmű-e vagy sem.

```
> if rank(A)==rank([A,b]) disp('Van megoldás')
>     if rank(A)==size(A,2) disp('Egyértelmű megoldás van')
>     else disp('Végtelen sok megoldás van')
>     end
> else disp('Nincs megoldás')
> end
```

VÉGTELEN SOK MEGOLDÁS VAN

Mi a helyzet akkor, amikor van ugyan megoldás, de nem egyértelmű? Például 2 egyenletünk van 3 ismeretlenre? Ilyenkor általában az egyik változó értéke tetszőlegesen felvehető és a másik kettő ennek függvényében számítható. Van megoldásunk, de nem egyértelmű, mivel végtelen sokféleképpen vehetjük fel az általunk megkötött változó értékét. Matematikai megközelítés szerint ez akkor lehetséges, ha a mátrix rangja megegyezik a kibővített mátrix rangjával viszont ez a rang kevesebb mint a mátrix oszlopainak a száma, azaz r(A)=r(A|b) és r(A)<n. Ez lehet ranghiányos eset négyzetes együtthatómátrix esetén (m=n) (pl. x+y=3, 2x+2y=6), vagy

alulhatározott egyenletrendszer (m<n), amikor kevesebb egyenletünk van, mint ismeretlenünk. Ilyenkor végtelen sok megoldás van, mert a mátrix nullterének bármelyik n vektorát hozzáadhatjuk a megoldáshoz ($A \cdot n = 0 \rightarrow A(x+n) = b$). Ilyenkor megoldásnak a végtelen sok közül egyet szoktak kiválasztani, méghozzá általában (de nem mindig) a legkisebb normájú megoldást: $min|x|$. A legkisebb normájú megoldást matematikailag a következő formulával kaphatjuk meg:

$$x = A^T \cdot (A \cdot A^T)^{-1} \cdot b$$

Nézzünk egy példát alulhatározott egyenlet rendszerre, ahol 2 egyenletünk van 4 ismeretlennel:

$$7 \cdot x_1 + 2 \cdot x_2 + 2 \cdot x_4 = 1$$
$$x_1 + 8 \cdot x_2 + x_3 + 8 \cdot x_4 = 2$$

Mátrixos alakban felírva:

$$A = \begin{pmatrix} 7 & 2 & 0 & 2 \\ 1 & 8 & 1 & 8 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Nézzük meg a megoldások számát!

```
> A = [7 2 0 2; 1 8 1 8], b = [1; 2]
> rank(A)     % 2
> rank([A b]) % 2
> size(A,2)   % 4
```

Az A mátrix rangja megegyezik a kibővített mátrix rangjával, tehát van megoldás, méghozzá végtelen sok, mivel r(A)= r(A|b) = 2 < n=4.

Oldjuk meg a feladatot, úgy, hogy a végtelen sok megoldás közül a legkisebb normájú megoldást válasszuk! Használjuk a megoldáshoz a fenti képletet!

```
> x = A'*inv(A*(A'))*b
> % xa =0.0745
> %     0.1195
> %     0.0127
> %     0.1195
> norm(A*x-b) % 0
> norm(x) % 0.1852
```

A megoldás hibátlan, visszahelyettesítéskor az eltérésvektor normája 0, a megoldásvektor hossza pedig 0.1852. Ez az összes lehetséges megoldásvektor közül a 'legrövidebb', legkisebb normájú.

Az előző órán láttuk, hogy az inverz számítás lassú és sokszor pontatlan is, ha lehet akkor el szokták ezt kerülni. Célszerű lenne itt is valamilyen mátrix felbontást alkalmazni, azonban a korábban megismert LU és Cholesky felbontás csak négyzetes mátrixok esetében működik. Vannak olyan felbontások is, amelyek működnek nem négyzetes esetekben is, ilyen például a QR felbontás és az SVD felbontás.

QR FELBONTÁS

A QR módszer azt a tényt használja ki, hogy bármely A mátrix felbontható egy Q és R mátrix szorzatára ($A = Q \cdot R$), ahol Q egy (négyzetes) ortonormált mátrix, ahol $Q^{-1} = Q^T$, tehát $Q^T\ Q = I$ és R egy m×n-es felső háromszög mátrix. Legyen most A mátrix nem négyzetes (mxn-es, m≠n):

$$A = Q\,R$$

Az $A\,x = b$ egyenletrendszer megoldásához írjuk fel az $A \cdot x = b$ egyenletet a felbontott mátrixokkal:

$$Q\,R\,x = b$$

szorozzuk meg mindkét oldalt balról $Q^{-1} = Q^T$-tal, és az eredmény legyen B:

$$R\,x = Q^T\,b = B$$

Megoldás lépései:

- $B = Q^T\,b$ kiszámítása
- $R\,x = B$ megoldása, ahol már egyszerű visszahelyettesítést végezhetünk, hiszen R egy felső háromszögmátrix. Az eredmény a legkisebb hibájú megoldás lesz.

Oldjuk meg az előző feladatot QR felbontással (Matlab-ban a **qr** parancs)! Ellenőrizzük a felbontás helyességét és, hogy Q tényleg ortonormált-e! A megoldás során használjuk ki a Q és R mátrixok speciális voltát!

```
>  %% QR felbontás
>  [Q R] = qr(A)
>  % Q = -0.9899   -0.1414
>  %     -0.1414    0.9899
>  %
>  % R = -7.0711   -3.1113   -0.1414   -3.1113
>  %           0    7.6368    0.9899    7.6368
>  norm(A-Q*R)    % 8.8818e-16 (felbontás ellenőrzése)
>  norm(Q'*Q)     %  1 (Q tényleg ortonormált, Q'*Q egységmátrix)
>  B=Q'*b % Az új jobb oldal
>  %    -1.2728
>  %     1.8385
>  % A megoldás
>  opt.UT=true;
>  x=linsolve(R,B,opt)
>  % x = 0.0741
>  %     0.2407
>  %          0
>  %          0
>  norm(A*x-b) % 0
>  norm(x) % 0.2519
```

QR felbontással nem ugyanazt a megoldást kaptuk, mint korábban, a megoldás itt is hibátlan, hiszen az eltérésvektor normája 0, a megoldásvektor hossza azonban nem 0.1852, hanem nagyobb, 0.2519. Feltűnő viszont, hogy van benne kettő darab nulla elem is. A QR felbontás nem a legkisebb normájú megoldást adja, hanem azt, amiben a legtöbb nulla elem található. Hogy melyik megoldást szeretnénk megkapni, az a konkrét feladattól függ. Előfordulhat olyan eset, amikor ez az előnyösebb, és olyan is, amikor a másik. Találhatunk vajon olyan felbontást, amivel a legkisebb normájú megoldást kapjuk meg? Nézzünk meg egy másik nem négyzetes esetben is használható felbontást, az SVD felbontást!

SVD FELBONTÁS

Nem négyzetes mátrixú lineáris egyenlet rendszereknél egy másik gyakran használt módszer az SVD felbontás. Az SVD felbontás angolul a szinguláris érték szerinti felbontás rövidítése (Singular Value Decomposition). Nézzük meg röviden mit jelent a sajátérték és a szinguláris érték egy A mátrixnál!

Sajátértékek (λ): Azt mondjuk, hogy a λ szám az (n×n-es) **A** mátrix sajátértéke, ha létezik olyan nem nulla **x** vektor, melyre $A\,x = \lambda\,x$. Az ilyen **x** vektorokat az **A** mátrix λ sajátértékhez tartozó sajátvektorainak nevezzük. A sajátértékeket megkapjuk az $|A - I\,\lambda| = 0$ karakterisztikus egyenlet megoldásával. Matlab-ban: sajátértékek: **`E = eig(A),`** sajátértékek és sajátvektorok: **`[V,D] = eig(A)`**

Szinguláris érték: Az m×n-es A mátrix szinguláris értékei az A^TA (nem nulla) sajátértékeinek négyzetgyökei. Matlab-ban: **`S = svd(A)`**. A szinguláris értékek száma egyenlő a mátrix rangjával.

Bármely (m×n-es) A mátrix felírható a következő formában:

$$A = U \cdot S \cdot V^T$$

ahol az U és V mátrixok ortonormáltak, azaz $U^{-1} = U^T, V^{-1} = V^T$ és S diagonálmátrix. Az U mátrix m×m-es, és oszlopvektorai az $A \cdot A^T$ sajátvektorai, a V mátrix n×n-es és oszlopvektorai az $A^T \cdot A$ mátrix sajátvektoraival egyeznek meg. Az S mátrix (m×n-es), főátlójában a $A^T \cdot A$ sajátértékeinek pozitív négyzetgyökei – az ún. szinguláris értékek – állnak, a többi elem nulla. Figyelembe véve, hogy U és V ortonormáltak és S diagonálmátrix, segítségükkel az A mátrix inverze/pszeudoinverze könnyen kiszámítható:

$$A^{-1} = V \cdot S^{-1} \cdot U^T$$

Oldjuk meg az előző feladatot most SVD felbontással!

```
>  %% SVD felbontás
>  [U,S,V] = svd(A)
>  % U = -0.3979   -0.9174
>  %     -0.9174    0.3979
>  %
>  % S = 12.1209         0         0         0
>  %           0    6.3312         0         0
>  %
>  % V = -0.3055   -0.9515    0.0368   -0.0015
>  %     -0.6712    0.2130   -0.0933   -0.7039
>  %     -0.0757    0.0629    0.9943   -0.0406
>  %     -0.6712    0.2130   -0.0356    0.7092
>  %
>  %  Ellenőrzések
>  norm(A-U*S*V') % felbontás ellenőrzése: 3.0531e-15
>  norm(U'*U-eye(size(A,1))) % U mxm-es ortonormált, U'=inv(U): 3.5606e-16
>  norm(V'*V-eye(size(A,2))) % V nxn-es ortonormált, V'=inv(V): 2.6547e-16
>  diag(S), sqrt(eig(A'*A)) % szinguláris érték A'A sajátértékeinek gyöke
>  %    12.1209; 6.3312
>  %    0.0000 + 0.0000i; 0.0000 + 0.0000i, 6.3312 + 0.0000i, 12.1209 + 0.0000i
>  %
>  %  A pszeudoinverz előállítása
>  invS=(1./S)' % S inverze, a diagonál mátrix reciproka, transzponáltja
>  %     0.0825       Inf
```

```
>  %        Inf    0.1579
>  %        Inf       Inf
>  %        Inf       Inf
>  invS(invS==Inf)=0 % ahol Inf (végtelen) elem volt, ott legyen 0
>  %     0.0825         0
>  %          0    0.1579
>  %          0         0
>  %          0         0
>  invA=V*invS*U' % A mátrix pszeudoinverze
>  %     0.1479   -0.0367
>  %    -0.0088    0.0642
>  %    -0.0066    0.0097
>  %    -0.0088    0.0642
>  x=invA*b %  A megoldás
>  % x = 0.0745
>  %     0.1195
>  %     0.0127
>  %     0.1195
>  norm(A*x-b) % 4.9651e-16
>  norm(x) % 0.1852
```

Az SVD felbontás ugyanazt a legkisebb normájú megoldást adta vissza, mint az első esetben láttuk, amennyiben a feladat ezt igényli, akkor célszerű az SVD felbontást használni!

MATLAB BEÉPÍTETT FÜGGVÉNYEK (LINSOLVE, \, PINV)

Természetesen itt is használhatjuk a Matlab beépített függvényeit, azonban, mint láttuk a különböző módszerek eltérő eredményt adnak, így nem mindegy melyik megoldást használjuk.

```
>  %% Beépített Matlab függvények
>  % QR felbontás - legtöbb nullát tartalmazó megoldás
>  xa = A\b % QR felbontás - legtöbb nullát
>  xb = linsolve(A,b)
>  % xa = xb = 0.0741
>  %           0.2407
>  %           0
>  %           0
>  % SVD felbontás - legkisebb normájú megoldás
>  xc = pinv(A)*b
>  % xc = 0.0745
>  %      0.1195
>  %      0.0127
>  %      0.1195
>  type pinv
```

A Matlab beépített **** (**mldivide**) és a **linsolve** parancsa QR felbontást használ nem négyzetes mátrixok esetében, a **pinv** parancs pedig a pszeudoinverzet számítja SVD felbontással (ennek a tartalmát a **type** paranccsal kiírathatjuk a képernyőre). Alulhatározott esetben az első kettő a legtöbb nullát tartalmazó megoldást fogja adni, a harmadik pedig a legkisebb normájú megoldást.

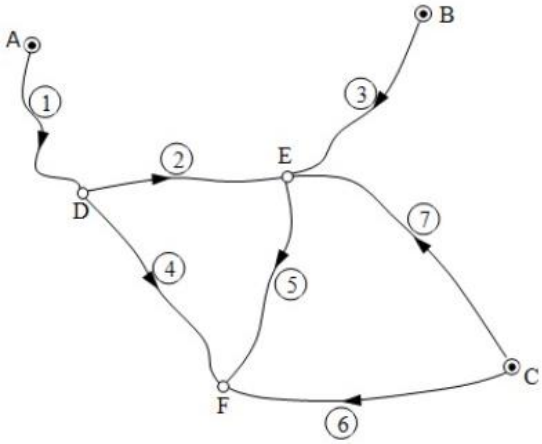
Emlékeztetőül négyzetes esetben a **** (**mldivide**) és a **linsolve** parancs LU vagy Cholesky felbontást használ.

NINCS MEGOLDÁS (LEGKISEBB HIBÁJÚ MEGOLDÁS)

Gyakran előforduló eset a mérnöki gyakorlatban, amikor matematikailag nincs megoldása az egyenletrendszernek, mert túlhatározott rendszerünk van, azaz több mérésünk van, mint ismeretlenünk. A geodéziában szinte mindig ilyen feladatokkal találkozhatunk, hiszen a mérési hibák miatt mindig több mérés történik, mint, ami matematikailag tényleg szükséges lenne a feladat megoldásához. A gyakorlatban sokszor előfordulnak a kis mérési hibák mellett durva hibák is, ami miatt sokszor megoldhatatlan lesz a feladat, ha nincs fölös mérésünk.

Nézzünk most egy geodéziai példát, ahol szintezési hálózatban kell meghatározni az alappontok magasságát, úgy, hogy van több fölös mérésünk is! Nézzük meg az alábbi ábrán látható szintezési hálózat kiegyenlítését. Megmértük az 1-7 szintezési szakaszok magasság különbségeit, ismerjük 3 alappont (A,B,C) tengerszint feletti magasságát és keressük a további 3 alappont (D,E,F) magasságát! A szintezési vonalak hosszai nagyjából megegyeznek így most azonos súlyúnak tekintjük az egyes vonalak méréseit. Az ismert magasságok: H_A=183.506m, H_B=192.353m, H_C=191.880m, és a mért magasság különbségek:

$$H_D - H_A = H_D - 183.506 = +6.135 \rightarrow 1.szakasz$$
$$H_E - H_D = +8.343 \rightarrow 2.szakasz$$
$$H_E - H_B = H_E - 192.353 = +5.614 \rightarrow 3.szakasz$$
$$H_F - H_D = +1.394 \rightarrow 4.szakasz$$
$$H_F - H_E = -6.969 \rightarrow 5.szakasz$$
$$H_F - H_C = H_F - 191.880 = -0.930 \rightarrow 6.szakasz$$
$$H_E - H_C = H_E - 191.880 = +6.078 \rightarrow 7.szakasz$$



Írjuk fel az egyenletrendszert mátrixos alakban, a meghatározandó ismeretlenek: H_D, H_E, H_F!

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} H_D \\ H_E \\ H_F \end{pmatrix} = \begin{pmatrix} 6.135 + 183.506 \\ 8.343 \\ 5.614 + 192.353 \\ 1.394 \\ -6.969 \\ -0.930 + 191.880 \\ 6.078 + 191.880 \end{pmatrix} = \begin{pmatrix} 189.641 \\ 8.343 \\ 197.967 \\ 1.394 \\ -6.969 \\ 190.950 \\ 197.958 \end{pmatrix}$$

A fenti egyenletrendszer esetében 7 egyenletünk van 3 ismeretlenre (m>n), ez egy túlhatározott egyenletrendszer. Írjuk be Matlab-ba ezt az egyenletrendszert! Bevihetjük a mátrixokat kézzel is, de el is vannak mentve az értékek a **szintezes.txt** állományban.

```
> A = [1 0 0;-1 1 0;0 1 0;-1 0 1;0 -1 1;0 0 1;0 1 0]
> b = [189.641; 8.343; 197.967; 1.394; -6.969; 190.950; 197.958]
```

Vagy

```
> Ab = load('szintezes.txt')
```

```
> A = Ab(:,1:3)
> b = Ab(:,4)
```

Ha megnézzük az első ábrát a megoldás létezéséről, akkor láthatjuk, hogy, akkor nincs megoldásunk, ha a mátrix rangja kisebb, mint a kibővített mátrix rangja, azaz r(A)<r(A|b). Ez akkor fordul elő, ha a *b* vektor "kilóg" az *A* oszlopvektorainak teréből. Ritkábban négyzetes alakmátrix esetén is (m=n) előállhat ez az eset, leggyakrabban azonban túlhatározott egyenletrendszernél (m>n), amikor több egyenletünk van, mint ismeretlenünk. Megoldhatatlan négyzetes alakmátrixra példa lehet a következő: x+y=2, és x+y=3, ahol két egyenlet van két ismeretlennel, még sincs megoldás. Nézzük meg, hogy a mi példánkban mi a helyzet!

```
> rank(A)      % 3
> rank([A b])  % 4
> size(A)      % 7 sor 3 oszlop
```

Most a mátrix rangja 3, a kibővített mátrixé pedig 4, tehát r(A)<r(A|b), azaz nincs megoldás. A mátrix oszlopainak száma n=3, sorainak száma pedig m=7, m>n, tehát ez egy túlhatározott egyenletrendszer.

A kérdés, hogyan tudjuk megoldani a megoldhatatlan egyenletrendszert? A válasz, hogy nem egy hibátlan megoldást keresünk, hanem a legkisebb hibájú megoldást! Ezt a megoldást úgy találhatjuk meg, hogy a maradék ellentmondások négyzetösszegét minimalizáljuk: $\|A \cdot x - b\| \rightarrow min$. A minimális hibájú közelítő megoldás matematikailag a következő lesz:

$$x = (A^T \cdot A)^{-1} \cdot A^T \cdot b$$

Matlab-ban:

```
> x = inv(A'*A)*A'*b  % 189.6153, 197.9588, 190.9830
```

A fenti megoldást nem szokták használni az inverz számítás időigényessége, pontatlansága miatt. Nézzük meg az előbb megismert két mátrix felbontást használva is az eredményeket!

QR FELBONTÁS

Oldjuk meg a szintezési feladatot is QR felbontással (Matlab-ban a **qr** parancs)! Most nem ellenőrizzük le a felbontás helyességét, csak az eltérésvektor normáját nézzük!

```
> [Q R] = qr(A)
> B=Q'*b % A jobb oldal
> % A megoldás
> opt.UT=true;
> x=linsolve(R,B,opt)
> %   189.6153
> %   197.9588
> %   190.9830
> norm(A*x-b) % 0.0506
```

Az eredmény: H_D=189.6153; H_E=197.9588; H_F=190.9830, az eltérésvektor normája pedig körülbelül 5 cm.

SVD FELBONTÁS

Oldjuk meg a szintezési hálózat kiegyenlítését most SVD felbontással! Most sem ellenőrizzük le a felbontás helyességét, csak az eltérésvektor normáját nézzük!

```
>  [U,S,V] = svd(A)
>  %  A pszeudoinverz előállítása
>  invS=(1./S)' % S inverze
>  invS(invS==Inf)=0 % ahol Inf (végtelen) elem volt, ott legyen 0
>  invS*S % ellenőrzés
>  invA=V*invS*U' % A mátrix pszeudoinverze
>  invA*A % ellenőrzés
>  x=invA*b %  A megoldás: x = 189.6153, 197.9588, 190.9830
>  norm(A*x-b) % 0.0506
```

Túlhatározott esetben mind a QR, mind az SVD felbontással ugyanazt az eredményt kaptuk, mint az eredeti minimális hibájú megoldáshoz tartozó formulával.

MATLAB BEÉPÍTETT FÜGGVÉNYEK (LINSOLVE, \, PINV)

Természetesen itt is használhatunk Matlab beépített függvényeket . Mint már láttuk, a Matlab beépített **** (**mldivide**) és a **linsolve** parancsa QR felbontást használ nem négyzetes mátrixok esetében, a **pinv** parancs pedig a pszeudoinverzet számítja SVD felbontással. Nézzük meg a megoldásokat ezekkel is!

```
>  x1 = A\b
>  x2 = linsolve(A,b)
>  x3 = pinv(A)*b
>  norm(A*x1-b) % 0.0506
```

A beépített függvényekkel történő megoldások nem figyelmeztetnek arra, hogy egyértelmű megoldás helyett most a legkisebb hibájú megoldást kaptuk meg, és hogy mekkora ez a maradék eltérés, ezért mindenképpen szükség van ellenőrzésre is. Nézzük meg a kétféle algoritmus futásidejét is 10000 futtatás esetén!

```
>  tic
>  for i=1:1000
>      x1 = A\b;
>  end
>  toc % Elapsed time is 0.005924 seconds.
>  tic
>  for i=1:1000
>      x3 = pinv(A)*b;
>  end
>  toc % Elapsed time is 0.063876 seconds.
```

A QR felbontás egy nagyságrenddel gyorsabb volt, mint az SVD felbontás. Túlhatározott esetben célszerű a QR felbontást használó **x=A\b** alkalmazása, a hatékonyság/gyorsaság miatt. Alulhatározott esetben azonban a feladat jellegétől függően, ha a végtelen sok megoldás közül a legkisebb normájúra van szükség, akkor az SVD felbontást használó **x=pinv(A)*b** megoldást alkalmazzuk, ha a legtöbb nullát tartalmazó lenne ideális, akkor pedig a QR felbontást használó **x=A\b** parancsot alkalmazzuk. A **linsolve** parancs alkalmazása akkor célszerű, ha előzetes ismereteink vannak a mátrix speciális voltáról (pl. háromszögmátrix, szimmetrikus, pozitív definit), mivel itt van lehetőségünk ezeket opcióként megadni.

ITERATÍV MÓDSZEREK (JACOBI, GAUSS-SEIDEL)

Lineáris egyenletrendszereket megoldhatunk direkt és iteratív módszerekkel is. Amiket eddig néztünk eddig, a mátrix felbontások, azok a direkt megoldások voltak. Nézzük meg mikor lehet célszerű nem direkt, hanem iteratív megoldásokat használni!

Nézzünk most egy másik lineáris egyenletrendszerre vezető példát! Határozzuk meg az egyes vízkezelő reaktorokban a koncentráció értékét az alábbi kapcsolás esetén, tökéletes keveredést (a koncentráció azonos a reaktortér minden pontjában) feltételezve.

A mérlegegyenletek:

$6\,c_1 - c_3 = 50$

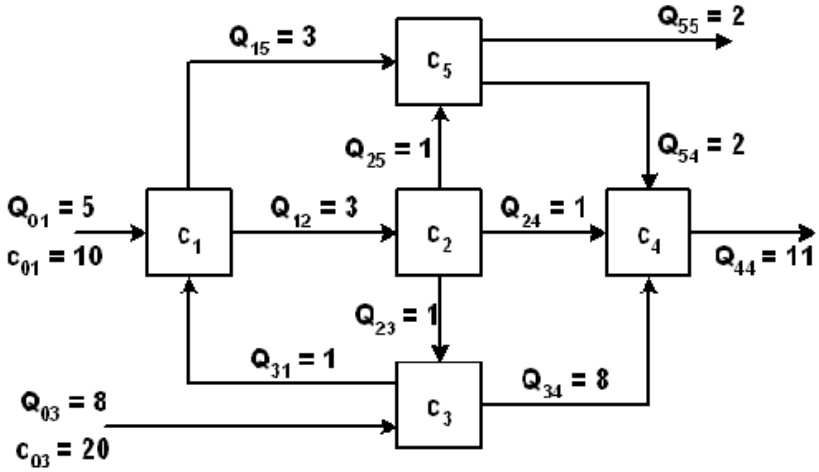
$-3\,c_1 + 3\,c_2 = 0$

$-c_2 + 9\,c_3 = 160$

$-c_2 - 8\,c_3 + 11\,c_4 - 2\,c_5 = 0$

$-3\,c_1 - c_2 + 4\,c_5 = 0$

Az egyenletrendszer mátrixos alakja:



$$A = \begin{pmatrix} 6 & 0 & -1 & 0 & 0 \\ -3 & 3 & 0 & 0 & 0 \\ 0 & -1 & 9 & 0 & 0 \\ 0 & -1 & -8 & 11 & -2 \\ -3 & -1 & 0 & 0 & 4 \end{pmatrix}; \qquad b = \begin{pmatrix} 50 \\ 0 \\ 160 \\ 0 \\ 0 \end{pmatrix}$$

Először vizsgáljuk meg hogy van-e megoldása a feladatnak és egyértelmű-e? Ehhez töltsük be az A mátrixot és b vektort tartalmazó vizkezeles.txt fájlt (vagy beírhatjuk kézzel is a mátrixokat)!

```
> Ab = load('vizkezeles.txt'), A = Ab(:,1:end-1), b = Ab(:,end)
> rank(A), rank(Ab) % 5, 5
> size(A,2) % 5
```

A mátrix rangja (5) megegyezik a kibővített mátrix rangjával is és az A mátrix oszlopainak a számával is, tehát létezik és egyértelmű a megoldás.

Az alakmátrixot vizsgálva megfigyelhetjük, hogy nagyon sok a nulla elem benne, ezeket ritka mátrixoknak nevezzük. Azokban az esetekben, ahol sok a nulla az alakmátrixban és a főátlóban lévő elemek dominálnak sokkal hatékonyabb a direkt megoldás helyett iteratív módszereket használni (és itt most nem az ilyen kis méretű mátrixoknál lényeges a hatékonyság, hanem a több ezerszer több ezres méreteknél!). Eddig a lineáris egyenlet rendszerek direkt megoldási módszereit vizsgáltuk, ahol a megoldást az egyenletekkel végzett elemi átalakításokkal kaptuk. Iteratív módszerek esetében meg kell becsülni minden változónak egy kezdőértéket és ezt használhatjuk utána egy iteratív eljárásában, hogy pontosítsuk az eredményt. A módszer hasonlít a nemlineáris egyenleteknél alkalmazható fixpont módszerhez. Az egyenleteket át kell alakítani explicit formába, minden változót kifejezve a többi változó függvényében.

$$\begin{array}{l} a_{11}x_1 + a_{12}\,x_2 + a_{13}\,x_3 = b_1 \\ a_{21}x_1 + a_{22}\,x_2 + a_{23}\,x_3 = b_2 \\ a_{31}x_1 + a_{32}\,x_2 + a_{33}\,x_3 = b_3 \end{array} \quad \rightarrow \quad \begin{array}{l} x_1 = \big(b_1 - (a_{12}\,x_2 + a_{13}\,x_3)\big)/a_{11} \\ x_2 = \big(b_2 - (a_{21}x_1 + a_{23}\,x_3)\big)/a_{22} \\ x_3 = \big(b_3 - (a_{31}x_1 + a_{32}\,x_2)\big)/a_{33} \end{array}$$

Az iterációs folyamatban először megbecsüljük a kezdőértékeket, majd ezeket a jobb oldalba behelyettesítve újabb értékeket kapunk x-re. A második iterációban ezeket az új értékeket helyettesítjük be a jobb oldalba, és így megyünk tovább, amíg a kívánt pontosságot el nem érjük, amikor a két egymást követő iterációban kapott megoldás közötti különbség kisebb az előre megadott tolerancia értéknél.

Az iterációs képlet a fenti explicit egyenletrendszer általános alakja lesz (ez tulajdonképpen a Jacobi módszer iterációs képlete):

$$x_i = \frac{1}{a_{ii}} \left(b_i - \Big(\sum_{\substack{j=1 \\ j \neq i}}^{j=n} a_{ij}\,x_j\Big) \right)$$

Kétféle speciális iterációs módszert nézünk most meg, a Jacobi iterációt és a Gauss-Seidel iterációt. A kettő között a különbség abban van, hogy mikor használjuk fel az újonnan kiszámolt értékeit az ismeretleneknek. Jacobi módszernél az ismeretlenek becsült értékei, amiket az explicit egyenletrendszer jobb oldalán használunk egyszerre kerülnek frissítésre minden iterációs lépés végén, vagyis egy kezdőérték készlettel kiszámoljuk az összes új értéket és utána megyünk tovább. A Gauss-Seidel módszernél a már kiszámolt ismeretlenek még az iterációs lépésen belül frissítésre kerülnek a további ismeretlenek kiszámolása során. Ez azt jelenti, ha egy iteráción belül pl. már kiszámoltunk egy új x_1-et, akkor ezt már x_2, x_3,... számításánál felhasználjuk. Ez utóbbi éppen ezért általában gyorsabban konvergál.

Az iterációs képletet előállíthatjuk mátrixos alakban is, amivel a későbbiekben könnyebben tudunk dolgozni. Alakítsuk át az $A \cdot x = b$ egyenletrendszert iterációs formulává! Adjunk hozzá, majd vonjunk is ki a baloldalból $B \cdot x$-et, ahol *B* mátrix megválasztásától függ majd az iteráció típusa (Jacobi vagy Gauss-Seidel).

$$B \cdot x + A \cdot x - B \cdot x = B \cdot x + (A - B) \cdot x = b$$

Rendezzük át:

$$B \cdot x = -(A - B) \cdot x + b$$

Szorozzuk meg balról B^{-1} mátrixxal:

$$x = -B^{-1}(A - B)\,x + B^{-1}b = -B^{-1}A\,x + B^{-1}B\,x + B^{-1}b$$

Emeljük ki x-et a jobb oldalon:

$$x = (I - B^{-1}A)x + B^{-1}b$$

A fentiek alapján írhatjuk fel az iterációs képletet a *(k+1).* iterációra:

$$x^{(k+1)} = (I - B^{-1}A)x + B^{-1}b$$

legyen $AI = (I - B^{-1}A)$ és $bi = B^{-1}b$, ekkor az iteráció képlete felírható a következő alakban:

$$x^{(k+1)} = AI \cdot x + bi$$

Ebben az általános alakban egységesen megadható a Jacobi módszer és a Gauss-Seidel módszer is. Eltérés csak a *B* mátrixban lesz, amivel *AI*-t és *bi*-t számoljuk. Jacobi módszer esetében *B* mátrix az *A* mátrix főátlója - **diag(diag(A))** (kifejtve az iteráció képletét így ugyanazt kapjuk, mint az explicit alakból meghatározott képlet, ahol mindig a főátló elemeivel (a_{ii}) kellett osztani), Gauss-Seidel módszer esetében pedig *B* mátrix az *A* mátrix alsó háromszögmátrixa - **tril(A)**.

Az iterációs megoldást elsősorban a diagonálisan domináns esetekben lehet hatékonyan alkalmazni, ez azt jelenti, hogy a főátlóban lévő elem abszolút értéke nagyobb, mint az abban a sorban lévő összes többi elem abszolút értékének az összege (a főátlóban lévő elemek dominálnak). A diagonális dominancia elégséges, de nem szükséges feltétele a konvergenciának. Nézzük meg, hogyan valósíthatjuk meg a Jacobi és a Gauss-Seidel iterációt Matlabban!

ITERÁCIÓS MÓDSZEREK MATLABBAN

Nézzük meg, hogyan lehet a kétféle iterációs módszert egységesen leprogramozni Matlabban, az alapértelmezett módszer legyen a Jacobi módszer! Nézzük meg az iterativ.m fájlt!

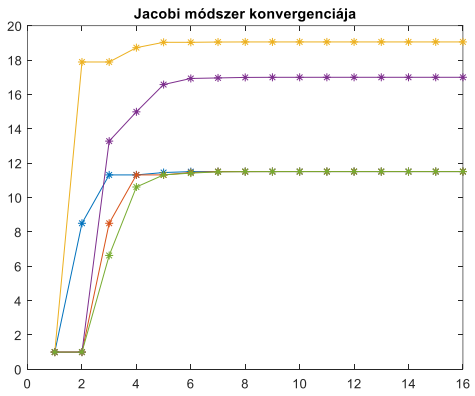
```
function [x,i,X]=iterativ(A,b,e,imax,method)
% A*x=b Lineáris egyenletrendszer megoldása iteratív módon,
% Jacobi vagy Gauss-Seidel módszerrel (alapértelmezett: Jacobi).
% Bemenet: A, b, e-hibahatár, imax - maximális iteráció szám
% method - 'Jacobi' vagy 'GaussSeidel' (nem kötelező megadni)
% Kimenet: x-megoldás, i -iteráció szám, X-iterációs lépések

% Módszer meghatározása
    if nargin==4; method='Jacobi'; end
    switch method
        case 'Jacobi'
            B = diag(diag(A));  % A mátrix főátlója mátrixban
        case 'GaussSeidel'
            B=tril(A);  % A mátrix alsó háromszög
        otherwise % Jacobi
            B = diag(diag(A)); % A mátrix főátlója mátrixban
    end
% Első iterációs lépés
Ai=eye(5)-inv(B)*A;
bi = inv(B)*b;
x0=ones(size(b)); % kiinduló érték
i=1; x1=Ai*x0+bi; % Első iteráció
X=[x0,x1];
% Iterációs ciklus
while i<=imax && norm(x1-x0)>e
    x0=x1;
    x1=Ai*x0+bi;
    X=[X x1];
    i=i+1;
end
x=X(:,end); % megoldás
end
```

Bemenet az *A* mátrix, a *b* vektor, *e* a megengedett eltérés két egymást követő iteráció között, a maximális iteráció szám (*imax*) és a módszer (method), ami vagy `'Jacobi'` vagy `'GaussSeidel'` lehet. Alapértelmezett módszer a Jacobi módszer, azaz nem kötelező megadni az utolsó bemenetet. Ezt vizsgálja a **nargin** változó (Number of function input arguments), a megadott bemenő paraméterek számát. Ha csak 4 változót adott meg valaki, akkor a módszer alapértelmezetten Jacobi módszer lesz. A *B* mátrix Jacobi módszer esetében az *A* mátrix főátlója, Gauss-Seidel módszer esetében pedig az *A* mátrix alsó háromszögmátrixa lesz!

A megoldáshoz elő kell állítani az *AI* mátrixot, a *bi* vektort és meg kell adni a kezdő x_0 vektort, ami most egységvektor lesz. Az első iteráció után addig folytatja a program az iterációkat, amíg két egymást követő iteráció között az eltérés egy megadott toleranciánál kisebb, vagy el nem értük a maximális iteráció számot. Három kimenetünk van, a megoldás – x, az iterációk száma – i, és az összes iterációs lépés során kapott megoldás vektorok összessége az X mátrixban, ahol egymás melletti oszlopokban vannak az egymást követő iterációs lépések elemei.

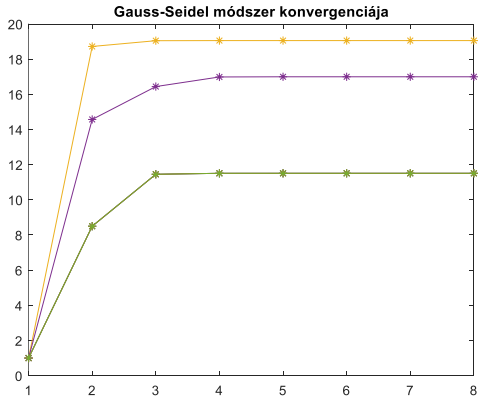
Először Jacobi módszerrel oldjuk meg a feladatot!

```
> %% Jacobi módszer
> [x,i,X] = iterativ(A,b,1e-6,100,'Jacobi')
> % Ellenőrzés (relatív hiba)
> h1 = norm(A*x-b);
> fprintf('Az iterációk száma: %d, relatív hiba: %g \n',i,h1)
> %Az iterációk száma: 15, relatív hiba:
  2.10763e-06
>
> %  Grafikus megjelenítés
> figure(1); plot(X','*-')
> title('Jacobi módszer konvergenciája')
```



Utána oldjuk meg Gauss-Seidel módszerrel is!

```
> %% Gauss-Seidel módszer
> [x,i,X] = iterativ(A,b,1e-
  6,100,'GaussSeidel')
> % Ellenőrzés (relatív hiba)
> h1 = norm(A*x-b);
> fprintf('Az iterációk száma: %d, relatív hiba: %g \n',i,h1)
> % Az iterációk száma: 7, relatív hiba: 1.28853e-08
>
> % Grafikus megjelenítés
> figure(2);plot(X','*-');
> title('Gauss-Seidel módszer
  konvergenciája')
```



A megoldásból látjuk, hogy míg a Jacobi módszernek 15 iterációra volt szüksége, addig a Gauss-Seidel módszernek elegendő volt 7.

A Matlab-nak egyébként van egy saját függvénye iteratív megoldásra, ez a **gmres** parancs. Lásd:

```
> [x, flags, relres, iter, resvec] = gmres(A,b)
> figure(3); plot(resvec,'b*-') % hibák kirajzolása az iteráció során
```

Illetve ritka mátrixok tárolására van egy saját formátuma, amit nagyméretű ritka mátrixoknál érdemes használni:

```
> AS=sparse(A)
```

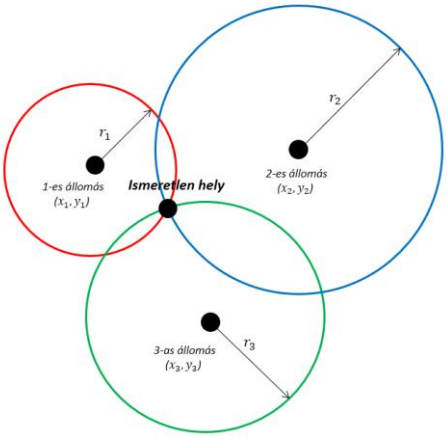
ÚJ FÜGGVÉNYEK A GYAKORLATON

qr	-	QR felbontás
svd	-	SVD felbontás
pinv	-	Pszeudoinverz számítás SVD felbontással
type	-	Szöveges fájl tartalmának képernyőre írása
tril	-	Egy mátrix alsó háromszögmátrixa
nargin	-	Függvény megadott bemenő paramétereinek a száma
gmres	-	Lineáris egyenletrendszer iteratív megoldása
sparse	-	Ritka mátrixok tárolása

7. NEMLINEÁRIS EGYENLETRENDSZEREK MEGOLDÁSA

Nemlineáris egyenletek gyakran előfordulnak az építőmérnöki feladatok során. Vannak olyan esetek is, amikor nem egy darab nemlineáris egyenlet zérushelyeit keressük, hanem több nemlineáris egyenlet metszéspontját, egy nemlineáris egyenletrendszer megoldását. Ilyenek példák például jellemzően a különböző geodéziai pontmeghatározási feladatok (ívmetszés, előmetszés, hátrametszés stb.), de a rúdszerkezetek egyensúlyi egyenletei és a súlytámfalak tervezése során is nemlineáris egyenletrendszereket kell megoldanunk. Most nézzünk meg egy mobiltelefonnal történő pozíció meghatározási feladatot fix mobil adótornyokhoz képest, amikor megmérjük a telefon és a környező adótornyok távolságait. Ez tulajdonképpen egy ívmetszési feladat.

A mobiltelefonok pozíciójának meghatározásakor rendelkezésre áll az eszköz és a bázisállomások távolsága. A távolságok egy-egy kört határoznak meg a bázisállomások körül (lásd ábra). A körök másodfokú egyenletek, és ezek metszéspontja adja a mobiltelefon pozícióját. Az egyenleteket a következő implicit alakban adhatjuk meg, két ismert bázis esetén:



$$(x - x_1)^2 + (y - y_1)^2 - r_1^2 = 0$$

$$(x - x_2)^2 + (y - y_2)^2 - r_2^2 = 0$$

Amennyiben legalább 3 távolság, azaz 3 kör, valamint a bázisok koordinátái ismertek, az ismeretlen hely egyértelműen meghatározható.

Két másodfokú egyenletből álló két ismeretlenes egyenletrendszer megoldása 4. fokú polinom gyökeinek megtalálására vezethető vissza. Ezt megoldhatnánk visszahelyettesítéssel egyedileg, vagy megpróbálhatjuk automatizálni a folyamatot numerikus módszerek használatával.

EGYENLETRENDSZER VEKTOROS JELÖLÉSMÓDJA

A nemlineáris egyenletrendszerek általános megoldásához vezessük be a vektoros jelölésmódot, így egyszerűbb lesz dolgozni az egyenletekkel és a Matlab beépített függvényei is ilyen alakban igénylik a megadást.

Általában akkor tudjuk megtalálni egy egyenletrendszer megoldását, amikor az ismeretlenek száma megegyezik az egyenletek számával. Az egyenletrendszereket a következő alakban szokás megadni:

$$f_1(x_1, x_2, x_3, \dots, x_n) = 0$$

$$f_2(x_1, x_2, x_3, \dots, x_n) = 0$$

$$f_3(x_1, x_2, x_3, \dots, x_n) = 0$$

$$\vdots$$

$$f_n(x_1, x_2, x_3, \dots, x_n) = 0$$

Az egyszerűség kedvéért a megoldások legyenek az $\boldsymbol{x}$ vektor elemei $\boldsymbol{x} = (x_1, x_2, x_3, \dots, x_n)$, és az $\boldsymbol{f}$ is legyen egy vektor, amelyben az egyenletek vannak $\boldsymbol{f} = (f_1, f_2, f_3, \dots, f_n)$. Ezzel a jelöléssel az egyenletrendszert egyszerűen le tudjuk írni:

$$\boldsymbol{f}(\boldsymbol{x}) = 0$$

Egyváltozós esetben egy x_0 kezdeti értéket kell felvennünk. Többváltozós esetben egy $\boldsymbol{x_0}$ vektorban adjuk meg az összes változó kezdeti értékeit. Minél több változónk van, annál több jó kezdeti értéket szükséges meghatároznunk, ami nem mindig könnyű feladat. Gyakorlatban azonban kihasználhatjuk az adott probléma mérnöki ismereteit és ezek alapján gyakran tudunk adni jó közelítő értékeket akár többváltozós esetre is. Ha tudjuk ábrázolni az egyenleteket (például kétváltozós esetben), akkor az ábra alapján is meghatározhatunk kiinduló adatokat.

POZÍCIÓMEGHATÁROZÁS MOBILTELEFONNAL

A feladatunk pozíció meghatározás lesz mobiltelefonnal. A példában tegyük fel, hogy csak 2 mobil adótoronytól ismerjük a távolságot, így a pozícióra két lehetséges helyet fogunk kapni. Az egyes bázisállomások koordinátáit és a távolságméréseket az alábbi táblázat foglalja össze (kilométer mértékegységben).

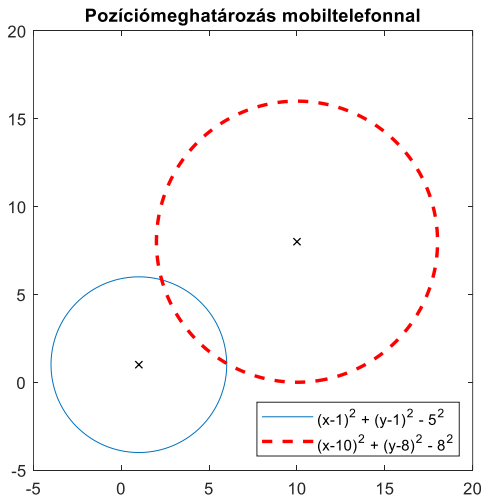
Bázis sorszám	**X koordináta** (x_i)	**Y koordináta** (y_i)	**Bázis-terminál távolság** (r_i)
1	1	1	5
2	10	8	8

Az egyenleteket a következő implicit alakban adhatjuk meg:

$$(x-1)^2 + (y-1)^2 - 5^2 = 0$$

$$(x-10)^2 + (y-8)^2 - 8^2 = 0$$

ahol az *x,y* koordinátákat keressük.



Először ábrázoljuk a két implicit alakban adott függvényt az **fimplicit** paranccsal. Az **fimplicit** paranccsal f(x,y) = 0 implicit alakban megadott függvényeket lehet ábrázolni. A színek, vonaltípus stb. megadása a plot parancshoz hasonlóan történhet. Egyelőre használható még az **ezplot** parancs is erre a célra (illetve Octave-ban csak ez), de a későbbi Matlab verziókból az **ezplot** ki fog kerülni.

```
> clear all; clc; close all;
> f1 =@(x,y) (x-1).^2 + (y-1).^2 - 5^2;
> f2 =@(x,y) (x-10).^2 + (y-8).^2 - 8^2;
> figure(1);
> g1=fimplicit(f1,[-5 20 -5 20]); hold on;
> fimplicit(f2,[-5 20 -5 20],'--r','LineWidth',2);
> axis equal;
> legend('(x-1)^2 + (y-1)^2 - 5^2','(x-10)^2 + (y-8)^2 -
  8^2','Location','SE')
> title('Pozíciómeghatározás mobiltelefonnal')
```

Hogyan tudjuk megtalálni a két egyenlet metszéspontjait? Egyváltozós esetben a nemlineáris függvény zérushelyeit kerestük, amihez használhattuk például a Newton-módszert. Többváltozós esetben is alkalmazható a Newton-módszer, némi átalakítással. Nézzük ezt meg!

TÖBBVÁLTOZÓS NEWTON MÓDSZER

Nézzük meg részletesen az egyik legismertebb módszert, a többváltozós Newton-módszert, a nemlineáris egyenletrendszerek megoldására, hogy könnyebben megértsük a megoldás egy lehetséges módját. Ez az egyváltozós eset megoldásából általánosítható. Egyváltozós esetben a Newton módszer a függvény linearizálásából volt levezethető x_0 helyen:

$$f(x) \approx f(x_0) + f'(x_0) \cdot (x - x_0)$$

Többváltozós esetben *f* lineáris közelítése x_0 helyen:

$$\boldsymbol{f}(\boldsymbol{x}) \approx \boldsymbol{f}(\boldsymbol{x_0}) + J(\boldsymbol{x_0}) \cdot \boldsymbol{\Delta x}$$

ahol $\boldsymbol{f}(\boldsymbol{x})$ az egyenletrendszer egy oszlopvektorban, $\boldsymbol{x_0}$ a vektorban megadott kezdőértékek minden változóhoz, $\boldsymbol{\Delta x} = \boldsymbol{x} - \boldsymbol{x_0}$, az $f'(x_0)$ helyett szereplő $J(\boldsymbol{x_0})$ pedig az n×n-es Jacobi mátrix értéke az x_0 helyen. A Jacobi mátrix elemei az egyenletek parciális deriváltjai:

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

Pl. az alábbi egyenletrendszer Jacobi mátrixa:

$$\begin{matrix} 3x^2 + 2y + 1 = 0 \\ -x^3 - 5y + 2 = 0 \end{matrix} \quad \rightarrow \quad J(x) = \begin{bmatrix} 6x & 2 \\ -3x^2 & -5 \end{bmatrix}$$

Szeretnénk megoldani az $\boldsymbol{f}(\boldsymbol{x}) = 0$ egyenletrendszert.

$$f(x_{i+1}) \approx f(x_i) + J(x_i) \cdot (x_{i+1} - x_i) = 0$$

A kérdés, hogyan tudjuk úgy megadni x_{i+1}-et, hogy a fenti feltétel igaz legyen? Rendezzük át az egyenletrendszert!

$$x_{\mathrm{i}+1} = x_i - J(x_i)^{-1} \cdot f(x_i)$$

Ez megoldható, amennyiben létezik $J(x_i)$ inverze. Ez az alak a korábban tanult egyváltozós Newton módszer ($x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$) többváltozós megfelelője. Mint az előző órákon láttuk, gyakorlatban nem szoktunk közvetlenül inverzet számítani. Vezessük be a $\Delta x = x_{i+1} - x_i$ jelölést az $f(x_i) + J(x_i) \cdot (x_{i+1} - x_i) = 0$ egyenlethez, és vigyük át a bal oldalra $f(x_i)$-t. Így egy lineáris egyenletrendszert kell megoldanunk:

$$J(x_i) \cdot \Delta x = -f(x_i)$$

ahol $J(x_i)$ egy *nxn*-es ismert mátrix, $-f(x_i)$ egy ismert *nx1*-es vektor. A lineáris egyenletrendszerek megoldására láttunk több hatékony módszert is, amelyek mind különböző mátrix felbontásokat használtak. Használjuk most az $x = A\backslash b$ alakú megoldást, ami négyzetes esetben LU felbontást alkalmaz. Miután Δx értéke ismert, így x_{i+1} számítható:$x_{i+1} = x_i + \Delta x$.

Az egyes iterációkban a megoldandó feladat tehát:

- $J(x_i) \cdot \Delta x_i = -f(x_i)$ lineáris egyenletrendszer megoldása Δx_i-re
- $x_{i+1} = x_i + \Delta x_i$ számítása addig, amíg $f(x) \approx 0$ vagy $\Delta x \approx 0$ (amíg kisebb nem lesz egy megadott tolerancia értéknél).

TÖBBVÁLTOZÓS NEWTON-MÓDSZER MATLAB-BAN

Írjunk függvényt, ami megvalósítja a többváltozós Newton módszert a fentiek alapján! Legyen a függvény neve newtonsys. (Mentsük a fájlt a **newtonsys.m** fájlba!) A leállási feltétel most az, hogy az iteráció két egymást követő lépésében meghatározott megoldások egy megadott kicsiny epszilon értéknél kevésbé térjenek el egymástól, illetve a ciklus leáll akkor is, ha elértük a maximális iteráció számot. Bemenet lesz az egyenlet rendszer, a Jacobi mátrix, a kezdőértékek és a leállási feltételt meghatározó maximális iterációszám, illetve az a kis epszilon szám, aminél kisebb Δx érték esetén elfogadjuk a megoldást. Két kimenete lesz a függvénynek, a megoldás és az iteráció szám.

```
> function [x1, n] = newtonsys (f, J, x0, eps, nmax)
>     dx = J(x0)\-f(x0)); % első iteráció
>     x1 = x0 + dx;       % első iteráció
>     n = 1;
>     while norm(x1-x0)>eps && n<=nmax
>         x0 = x1;
>         dx = J(x0)\-f(x0);
>         x1 = x0 + dx;
>         n = n + 1;
>     end;
> end
```

MEGOLDÁS NEWTON-MÓDSZERREL

a) Jacobi mátrix előállítása

A többváltozós Newton módszerhez, mint láttuk szükség lesz az eredeti egyenletek és a kezdőértékek mellett az egyenletek parciális deriváltjaira is a Jacobi mátrixhoz. Ezt szimbolikus számításokkal tudjuk előállítani. Lehet külön-külön a parciális deriváltakat is számítani a **diff** paranccsal, és abból összeállítani a Jacobi mátrixot, de a Matlab-ban van egy parancs a Jacobi mátrix közvetlen előállítására (**jacobian**). A jacobian parancs használatához definiáljuk most szimbolikusan az egyenletrendszert!

```
> %% Megoldás többváltozós Newton-módszerrel
> syms x y
> fs1 = (x-1).^2 + (y-1).^2 - 5^2
> fs2 = (x-10).^2 + (y-8).^2 - 8^2
> disp('Jacobi mátrix')
```

```
> js = jacobian([fs1; fs2])
> % [  2*x - 2,  2*y - 2]
> % [ 2*x - 20, 2*y - 16]
```

Definiáljuk függvényként a Jacobi mátrixot, szimbolikus kifejezés helyett! Ezt megtehetjük úgy, hogy bemásoljuk az egyes elemeket a megfelelő helyekre CTRL+C/CTRL+V használatával, vagy használhatjuk itt is a matlabFunction parancsot, ami szimbolikus kifejezésből függvényt állít elő (lehetnek olyan esetek, amikor ez nem működik).

```
> J=@(x,y) [2*x - 2,  2*y - 2; 2*x - 20, 2*y - 16] % vagy:
> J = matlabFunction(js)
```

b) Az egyenletrendszer és a Jacobi mátrix vektorizálása

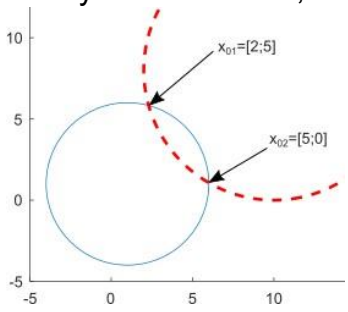
A megoldáshoz x,y változók helyett vektorváltozókat kell alkalmazzunk, ahol az ismeretlenek az ***x*** vektorban (x_1=x és x_2=y), az egyenletek pedig az ***f*** vektorban legyenek. A Jacobi mátrix esetében ugyanez szükséges! Vektorizáljuk az egyenleteinket és a Jacobi mátrixot is!

```
> % definiáljuk f, et és J-t is vektorváltozókkal
> f = @(x) [f1(x(1),x(2)); f2(x(1),x(2))];
> J = @(x) J(x(1),x(2));
```

c) Megoldás Newton-módszerrel

Oldjuk meg a problémát az általunk definiált **newtonsys** függvénnyel. Ehhez a **newtonsys.m** fájlnak ugyanabban a könyvtárban kell lennie ahová mentjük az aktuális script fájlt is. A numerikus megoldásokhoz, mint arról korábban már szó volt szükséges kezdőérték megadása. Minél közelebb van a kezdőérték a tényleges megoldáshoz annál gyorsabban meg tudjuk oldani a feladatot, annál valószínűbb, hogy konvergál a módszer. Az ábrából vegyünk kezdőértékeket, azokat az x és y koordinátákat, ahol a két görbe közelítőleg metszi egymást!

```
> % Oldjuk meg a problémát Newton módszerrel
> % kezdőértékek megadása az ábra alapján
> x01 = [2; 5] % egyik metszésponthozhoz
> x02 = [5; 0] % másik metszésponthoz
> [x1 i1] = newtonsys (f, J, x01, 1e-6, 100);
  % Az 1. megoldás
> [x2 i2] = newtonsys (f, J, x02, 1e-6, 100);
  % A 2. megoldás
```

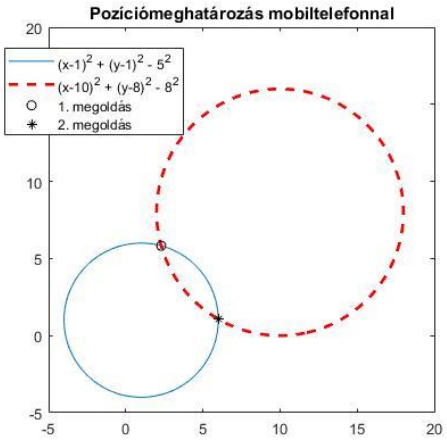


Az eredményeket írassuk ki formázva, és rajzoljuk is be az ábrába őket! Ellenőrzésnek helyettesítsük be a kapott eredményeket az implicit egyenletekbe, hogy tényleg 0-t kapunk-e (többváltozós esetben az eltérésvektor normáját szokás megadni, azaz az eltérésvektor hosszát)!

```
> disp('Lehetséges pozíciók Newton módszerrel:')
> fprintf('1. megoldás: %.4f,%.4f, iterációk: %d\n',x1, i1)
> fprintf('2. megoldás: %.4f,%.4f, iterációk: %d\n',x2, i2)
> % ellenőrzés
> norm(f(x1)) % 1.4648e-14
> norm(f(x2)) % 0
> plot(x1(1),x1(2),'ko')
> plot(x2(1),x2(2),'k*')
> legend('(x-1)^2 + (y-1)^2 - 5^2','(x-10)^2 + (y-8)^2 - 8^2',...
>     '1. megoldás','2. megoldás','Location','best')
```

A hiba a numerikus pontosságon belül 0-nak tekinthető.

Az eredmények:




```
Lehetséges pozíciók Newton módszerrel:
1. megoldás: 2.3005,5.8279, iterációk: 5
2. megoldás: 5.9995,1.0721, iterációk: 5
```

Nézzük meg a megoldást egy másik (x=1 és y=1) kezdőértékkel!

```
> % másik kezdőérték választása
> disp('Adjunk meg másik kezdőértéket!')
> x0 = [1; 1]
> [x3 i3] = newtonsys (f, J, x0, 1e-6,
  100);
> fprintf('3. megoldás: %.4f,%.4f,
  iterációk: %d\n',x3, i3)
```

Az eredmény:

```
Warning: Matrix is singular to working precision.
> In newtonsys at 2
  In mobiltornyok at 55
Warning: Matrix is singular, close to singular or badly scaled. Results may
be inaccurate. RCOND = NaN.
> In newtonsys at 7
  In mobiltornyok at 55

3. megoldás: NaN,NaN, iterációk: 2
```

Mi történt? Miért nincs megoldásunk? (NaN = Not a Number)

Azért, mert a Jacobi mátrix szinguláris. A geometriai ok, hogy a kezdeti érték egybe esik az egyik állomás koordinátájával, a kör középpontjával. Nem mindegy hogyan választjuk meg a kezdőértéket, ahogy az sem, milyen módszerrel oldjuk meg a feladatot, nem biztos, hogy konvergálni fog a megoldás.

MEGOLDÁS NUMERIKUSAN FSOLVE SEGÍTSÉGÉVEL

Természetesen a Matlab-nak is vannak beépített függvényei a nemlineáris egyenletrendszerek megoldásához. Nézzük meg először a numerikus eljárást használó **fsolve** parancsot! Az **fsolve**-ba több algoritmus is be van építve, ezek alapján próbálja meg az optimális megoldást megtalálni a program minimalizálva a maradék eltérések négyzetösszegeit. Az **fsolve** esetében több opció is megadható, amivel gyorsítani lehet az eljárást, például megadhatunk Jacobi mátrixot, ami a többváltozós Newton módszerhez is kell. A kezdőértékek maradjanak a korábbiak. Az egyenletrendszert itt is vektoros formában kell megadni, viszont a Jacobi mátrix megadása nem követelmény. A két különböző megoldás megtalálásához itt is kétszer kell lefuttatnunk az **fsolve** parancsot, egyszer az egyik megoldáshoz közeli kezdőértékekkel, egyszer a másik megoldáshoz közeli kezdőértékekkel.

```
> % numerikus megoldás - fsolve
> x1 = fsolve(f,x01) % x1 = [2.3005;  5.8279]
> x2 = fsolve(f,x02) % x2 = [5.9995;  1.0721]
```

Természetesen ugyanazokat ez eredményeket kaptuk mint korábban. Numerikus ellenőrzés végett helyettesítsük vissza a kapott eredményeket az eredeti egyenletrendszerbe és nézzük meg a 0-tól való eltéréseket.

```
> % ellenőrzés
> norm(f(x1)) % 7.4621e-12
> norm(f(x2)) % 7.0300e-08
```

A hiba a numerikus pontosságon belül itt is 0-nak tekinthető. Az **fsolve** alkalmazásakor megadható a kívánt pontosság az **optimset** változó használatával, hasonlóan az **fzero**-hoz, a *'TolFun'* illetve a *'TolX'* változókkal, előbbi a függvényértéket nézi, utóbbi az egymást követő x értékek változását. Az alapérték mindkettőnél 10^{-6}. Ha a feladatot 10^{-9}-en pontossággal akarjuk megoldani, akkor így tehetjük meg:

```
> x1 = fsolve(f,x01,optimset('TolFun',1e-9))
```

Az **fsolve**-t több kimenettel is meghívhatjuk, így olyan módon is, hogy rögtön megadja a függvényértékeket is, anélkül, hogy vissza kellene helyettesíteni az ellenőrzéshez, illetve az **optimset** használatával az egyes iterációs lépések is kiírathatók:

```
> opt = optimset('TolFun',1e-9,'Display','iter');
> [X,FVAL] = fsolve(f,x01,opt)
```

Az eredménye:

```
X = 2.3005
    5.8279
FVAL = 1.0e-11 *
    0.5059
    0.5485
```

Természetesen az `fsolve` parancs is használható egyváltozós nemlineáris egyenlet gyökeinek a megkeresésére is, mint az **fzero**, viszont ez utóbbi sokkal hatékonyabban működik egyváltozós esetben (de többváltozós esetben nem alkalmazható). Hasonló módon megoldhatunk **fsolve** segítségével nem csak algebrai polinomokat, hanem egyéb trigonometriai, logaritmus, exponenciális stb. függvényeket tartalmazó egyenlet rendszereket is.

MEGOLDÁS SZIMBOLIKUSAN SOLVE SEGÍTSÉGÉVEL

A példa, amit megoldottunk egy algebrai polinom, aminek létezik szimbolikus megoldása is, amikor nincs szükségünk kezdőértékek megadására sem, és egyszerre megkapjuk az összes megoldást, éppúgy, mint az egyváltozós esetben. Itt most a parancs is megegyezik az egyváltozós esetben megismerttel, ez a **solve** parancs. A **solve**, szemben az **fsolve**-val, csak algebrai polinomok esetében használható.

A **solve** használatához szimbolikusan kell megadni az egyenleteket, ezt már megtettük a Jacobi mátrix előállítása során (fs1 és fs2), használjuk most ezeket itt is! Az eredmény tartalmazza mindkét megoldást egyszerre, egzakt alakban, szimbolikus struktúra formában (*xs*). A tényleges változók értékeit a struktúra neve (*xs*) után ponttal megadva lehet lekérdezni. Célszerű ezeket utána számmá alakítani a **double** paranccsal.

```
> xs = solve(fs1, fs2)
> %    x: [2x1 sym]
> %    y: [2x1 sym]
> xs.x, xs.y % x,y változók értékei szimbolikusan
> %  (77*39^(1/2))/260 + 83/20         69/20 - (99*39^(1/2))/260
> %  83/20 - (77*39^(1/2))/260         (99*39^(1/2))/260 + 69/20
> xs = [double(xs.x) double(xs.y)] % x,y változók értékei numerikusan
> %     5.9995    1.0721
```

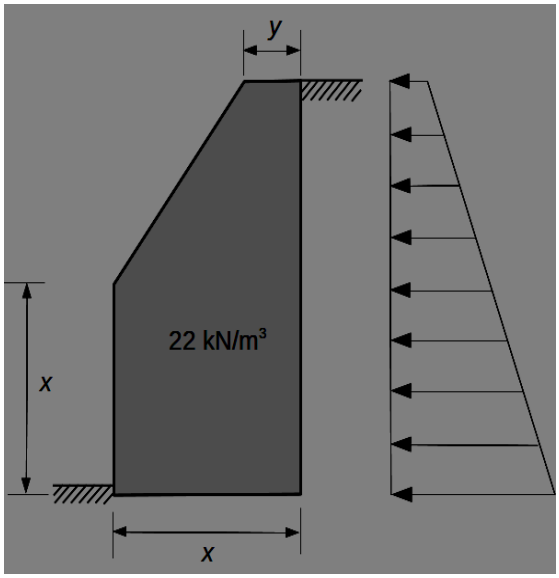
```
>  %     2.3005    5.8279
```

Mind a **solve**, mind az **fsolve** parancs ugyanazt a megoldást adta. Az `fsolve` esetében a vektorváltozós nullára rendezett egyenletrendszert kellett megadni bemenetként, és annyiszor kellett meghívni különböző kezdőértékkel, ahány megoldásunk van. A `solve` esetében a szimbolikusan definiált nullára rendezett egyenleteket kellett megadni. Ez utóbbi kezdőérték nélkül egy lépésben megadta az összes megoldást, viszont csak algebrai polinomoknál használható.

GYAKORLÓ FELADATOK

1) Támfal méretezése kicsúszás és felborulás ellen

Adott az ábrán látható súlytámfal, ahol a tervezés során szeretnénk az *x* és *y* méreteket úgy meghatározni, hogy a támfal mind az elcsúszás, mind a felborulás ellen biztosítva legyen. A támfal magassága 2.6 méter, a talajnyomás megoszló terhe pedig 1.6 kN/m^2-től 5.4 kN/m^2-ig változik. Az egyenletek felírásához szükség van az önsúly parciális tényezőjére (0.9), a talajnyomáséra (1.4), a talaj és a beton közötti csúszó súrlódás együtthatóra (0.3), és feltételezzük, hogy az elfordulás a támfal alsó élének 1/10-e körül jön létre. Kiszámítható, hogy a támfal az elcsúszás és felborulás ellen akkor még éppen biztosított, ha teljesülnek az alábbi egyenletek:

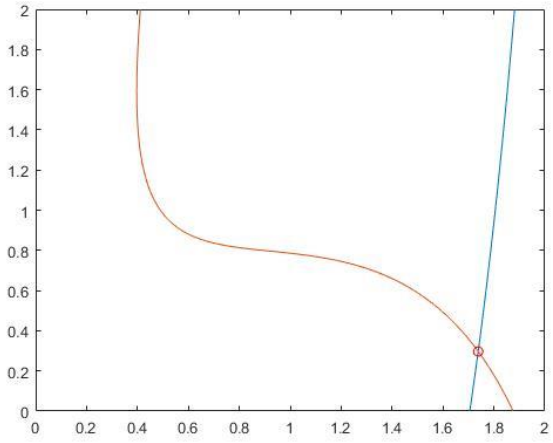


$$2x^2 - (2.6 - x)(x + y) = 4.29$$

$$6.24x^2 - (2.6 - x)(x - y)(7x - y) = 4.111$$

Ábrázoljuk az egyenleteket a [0,2]×[0,2] tartományon! Határozzuk meg az egyenletrendszer megoldását numerikusan és szimbolikusan is! A megoldást rajzoljuk be az ábrába, és ellenőrizzük az egyenletekbe behelyettesítve őket!

```
>  % Súlytámfal méretezése elcsúszás és kifordulás ellen
>  clc; clear all; close all
>  %% A függvények ábrázolása
>  f1 = @(x,y) 2*x.^2-(2.6-x).*(x+y)-4.29
>  f2 = @(x,y) 6.24*x.^2-(2.6-x).*(x-y).*(7*x-y)-4.111
>  figure(1);
>  fimplicit(f1,[0,2,0,2]); hold on;
>  fimplicit(f2,[0,2,0,2]);
>  %% numerikus megoldás
>  % a függvények vektorizálása
>  f=@(x) [f1(x(1),x(2));f2(x(1),x(2))]
>  % kezdőértékek az ábrából
>  x0=[1.8; 0.3];
>  [x, fval] = fsolve(f,x0)
>  % x =
>  %     1.7383
>  %     0.2969
>  % fval = 1.0e-10 *
```



```
>  %     0.0504
>  %     0.9412
>  % berajzoljuk az ábrába
>  %% szimbolikus megoldás
>  plot(x(1),x(2),'ro')
>  syms x y
>  fs1 = 2*x.^2-(2.6-x).*(x+y)-4.29
>  fs2 = 6.24*x.^2-(2.6-x).*(x-y).*(7*x-y)-4.111
>  xs = solve(fs1, fs2)
>  %    x: [4x1 sym]
>  %    y: [4x1 sym]
>  xs = [double(xs.x) double(xs.y)] % x,y változók értékei numerikusan
>  %   -0.4219 + 0.0407i  -0.8806 - 0.0810i
>  %   -0.4219 - 0.0407i  -0.8806 + 0.0810i
>  %    1.7383 + 0.0000i   0.2969 + 0.0000i
>  %    2.3294 + 0.0000i  21.9170 + 0.0000i
>  xs1 = real(xs(3,:))
>  % xs1 = 1.7383    0.2969
```

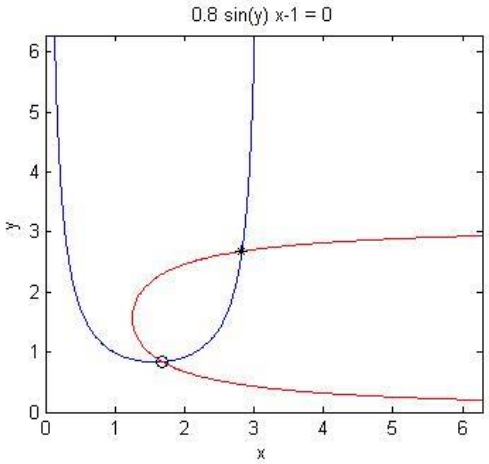
A szimbolikus megoldásnál 4 megoldást is kaptunk, azonban ebből kettő komplex megoldás, egy pedig kívül esik a megadott tartományon. Marad a harmadik sorban lévő megoldás (a 0 komplex rész elhagyható a **real** parancs használatával).

2) Gyakorlásképp oldjunk meg egy nem algebrai polinomokból álló egyenletrendszert is a Matlab beépített numerikus módszerével és a Newton módszerrel is!

$$1.2\sin(x)\cdot y = 1$$
$$0.8\sin(y)\cdot x = 1$$

Ábrázoljuk az egyenletrendszert az $x = 0\ldots 2\pi$, $y = 0\ldots 2\pi$ tartományon, majd határozzuk meg az egyenletrendszer megoldásait! Ellenőrizzük a megoldásokat visszahelyettesítéssel, és az ábrába is rajzoljuk be a megoldásokat!

```
>  clc; clear all; close all;
>  f1 = @(x,y) 1.2*sin(x).*y-1
>  f2 = @(x,y) 0.8*sin(y).*x-1
>  f = @(x) [f1(x(1), x(2));
   f2(x(1), x(2))];
>  h1 = fimplicit(f1, [0 2*pi]);
>  hold on;
>  h2 = fimplicit(f2, [0 2*pi]);
>  set(h1,'Color','b');
   set(h2,'Color','r')
>  %% Megoldás fsolve segítségével
>  % első megoldás
>  opt = optimset('Display',
   'iter');
>  x01 = [1.6; 0.8]
>  x1 = fsolve(f, x01, opt) % x1 = 1.6810    0.8384
>  norm(f(x1)) %  8.0708e-11
>  plot(x1(1), x1(2), 'ko')
>
>  % második megoldás
>  x02 = [2.8; 2.7]
>  x2 = fsolve(f, [2.8 2.7], opt) % x2 = 2.8258    2.6834
>  norm(f(x2)) %  1.1585e-08
>  plot(x2(1), x2(2), 'k*')
```



```
>
>  %% Megoldás Newton módszerrel
>  syms x y
>  f1s = 1.2*sin(x)*y-1
>  f2s = 0.8*sin(y)*x-1
>
>  js = jacobian([f1s; f2s])
>  % [ 1.2*y*cos(x),   1.2*sin(x)]
>  % [   0.8*sin(y), 0.8*x*cos(y)]
>
>  J=@(x,y) [1.2*y*cos(x),   1.2*sin(x);   0.8*sin(y), 0.8*x*cos(y)]
>  J = @(x) J(x(1),x(2));
>
>  [x1 i1] = newtonsys (f, J, x01, 1e-6, 100) % Az 1. megoldás
>  [x2 i2] = newtonsys (f, J, x02, 1e-6, 100) % A 2. megoldás
>
>  % x1 = [1.6810 0.8384], i1 = 4
>  % x2 = [2.8258 2.6834], i2 = 3
>  norm(f(x1)) %   1.1102e-16
>  norm(f(x2)) %   3.1402e-16
```

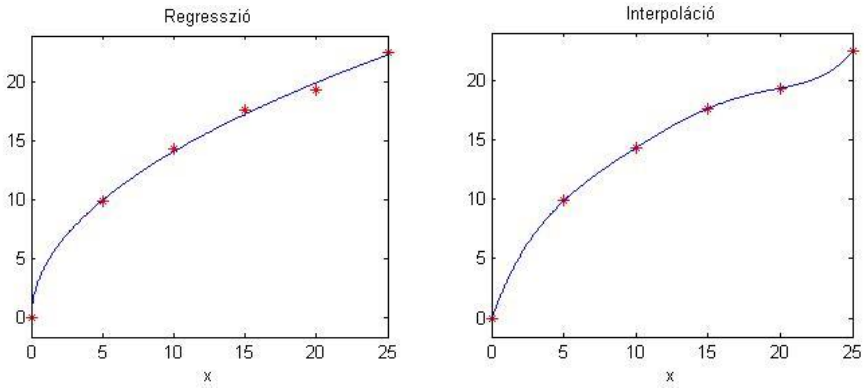
A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

fimplicit	-	f(x,y)=0 implicit alakban megadott függvények ábrázolása
axis equal	-	Egyenlő beosztás a tengelyeken
jacobian	-	Jacobi mátrix kiszámítása (egyenletet parciális deriváltjai)
fsolve	-	Nemlineáris egyenletrendszerek megoldása numerikusan
solve	-	Algebrai polinomokból álló egyenletrendszer megoldása szimbolikusan

8. REGRESSZIÓ

A mérnöki gyakorlatban sokféle fizikai mennyiséget mérnek műszerekkel, amiket manapság többnyire digitalizálnak és a mérési eredményeket diszkrét pontonként tárolják. Ilyen például szilárdságtanból a szakítóvizsgálat, ahol az anyagok húzó igénybevétellel szembeni ellenállását mérik vagy geodéziából a teljes hullámalakos lézerszkennerek mérései. A méréseket különböző módokon használják fel, van amikor tudják, hogy az adott fizikai mennyiség milyen alakú összefüggéssel írható le és függvényillesztéssel (regresszió) keresik ennek a függvénynek a paramétereit, máskor a mért pontok között lenne szükség a becsült értékekre (interpoláció) vagy meg kellene becsülni, hogy a mért tartományon kívül hogyan alakulnának az adatok (extrapoláció).

Regresszió (függvényillesztés) esetén meghatározzuk a pontokra legjobban illeszkedő függvény paramétereit, ilyenkor a meghatározott függvény többnyire nem megy át a mért pontokon, csak közel halad hozzájuk. Interpoláció esetén az ismert pontok közötti értékeket szeretnénk megbecsülni, úgy, hogy a görbe (általában valamilyen polinom) minden mért ponton áthaladjon.



REGRESSZIÓ MINŐSÍTÉSE

Regresszió esetében a 'legjobban' illeszkedő függvényt keressük. Hogyan tudjuk mérni, mi a legjobban illeszkedő? Ehhez meg kell határozzuk az eltéréseket a mért pontok y_i koordinátái és az illesztett függvény adott pontbeli $f(x_i)$ függvényértéke között. Ezeket maradék eltéréseknek is szokták nevezni (r_i): $r_i = y_i - f(x_i)$

A maradék eltéréseket kellene valamilyen módon minimalizálni, az összes pontra. Lehetne egyszerűen összeadni a hibákat, de ebben az esetben előfordulhatna, hogy nagyon nagy pozitív és nagyon nagy negatív hibák vannak, amelyek kiejtik egymást és hiába lesz nulla az összes eltérés, az illeszkedés rossz lesz. Lehetne a hibák abszolút értékeit venni, akkor nem ejthetnék ki egymást a hibák, viszont ebben az esetben nem egyértelmű a függvény illesztés, pl. adott ponthalmazra több egyenes is illeszthető ugyanakkora összes hibával. A megoldás erre, hogy a hibák négyzetösszegét minimalizálják. Ezzel jól lehet mérni az illeszkedés minőségét és egyértelmű megoldást ad a függvény paramétereire. Minimalizálandó összes hiba (S):

$$S = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

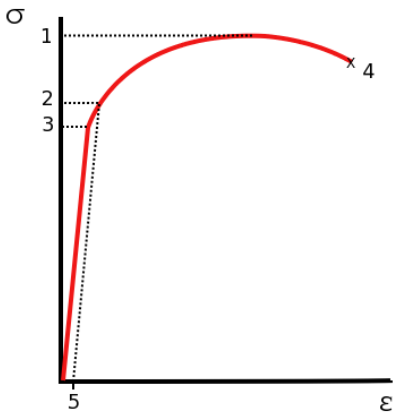
A regresszió lokális minősítése a maradék eltérések (r_i - rezídiumok) alapján történhet, a globális minősítés pedig ezeknek a korrigált tapasztalati szórásával:

$$\sigma = \sqrt{\sum_{i=1}^{n} \frac{(y_i - f(x_i))^2}{n - np}} = \sqrt{\sum_{i=1}^{n} \frac{r_i^2}{f}} = \sqrt{\frac{S}{f}}$$

ahol *n* a rendelkezésre álló mérések, *np* a becsült paraméterek, *f* pedig a fölös mérések száma (*f=n-np*).

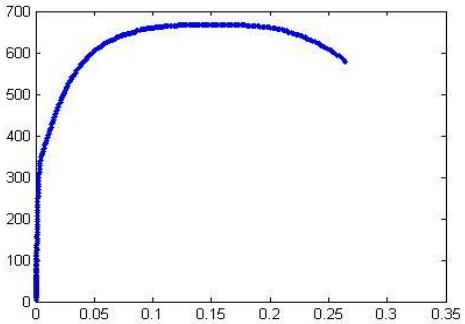
EGYENES ILLESZTÉS

Nézzünk egy konkrét példát szilárdságtanból! Egy jól megmunkálható, ötvözött acél szakítóvizsgálatából származó méréseket kell feldolgozni. A teszt során a terhelés folyamatosan növekszik egy maximum értékig, utána csökken, majd tönkremegy (eltörik) az anyag. A vizsgálatból meghatározható a rugalmassági modulus (vagy Young-modulus) (E), a szakítószilárdság (1), az egyezményes folyáshatár (2) (0.2% maradó alakváltozáshoz (5)) az arányossági határ (3)[9]. Most a rugalmassági határon belül érvényes Hooke-törvénye alapján meghatározzuk a rugalmassági modulust, az egyezményes folyáshatárt, a szakadás helyét és a szakítószilárdságot!



Ehhez először töltsük be a mérési adatainkat a **szakitovizsgalat.txt** fájlból! Ebben a mért fajlagos alakváltozásokhoz (ε - %) tartozó feszültség (σ - Mpa) értékek találhatóak meg.

```
> % Szakítóvizsgálat
> clc; clear all; close all;
> data =
  load('szakitovizsgalat.txt');
> x = data(:,1); % fajlagos
  alakváltozás, epszilon
> y = data(:,2); % feszültség, szigma
> figure(1)
> plot(x,y,'.')
```

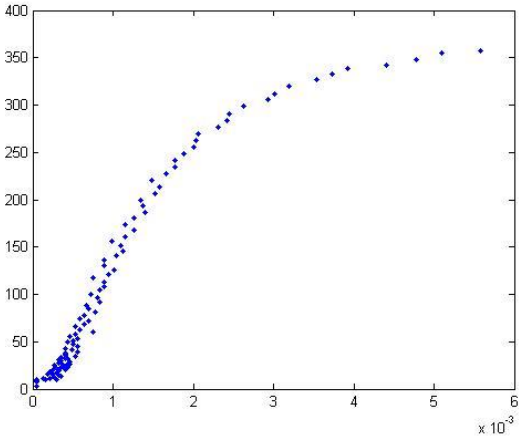


Azt, hogy hol megy tönkre az anyag, könnyű meghatározni, az utolsó mérési eredményt kell venni, ez a maximális ε érték is egyben. A szakítószilárdság meghatározásához a maximális feszültség (σ) értéket kell megkeresnünk:

```
> disp('Tönkremenetelhez tartozó fajlagos alakváltozás:')
> x(end) % ugyanaz, mint a max(x) => 0.2644 %
> disp('Szakítószilárdság:')
> max(y) %  668.3606 Mpa
```

[9] CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=647577

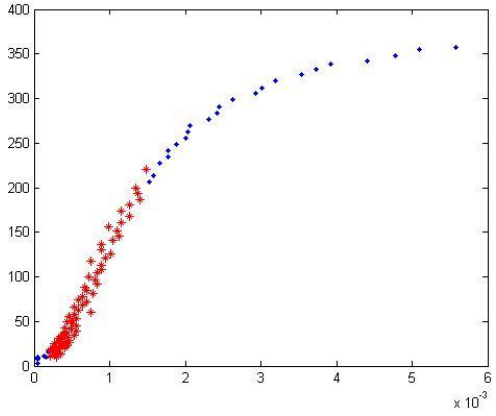
Tehát 26.4 % fajlagos alakváltozásnál szakadt el az anyag, és 668 MPa volt a maximális feszültség, a szakítószilárdság. A rugalmassági modulus meghatározásához azt a szakaszt kell megkeressük, ahol lineáris az összefüggés a fajlagos alakváltozás és a feszültség között, és erre kell egy egyenest illesztenünk. Az egyenes meredeksége lesz a rugalmassági modulus értéke. Ehhez nagyítsunk bele egy kicsit az ábrába, 0.6 % alakváltozásig és 400 MPa feszültségig!




```
> axis([0 0.006 0 400])
```

A fenti ábrán látszik, hogy a mérés elején, az origó közelében nem tekinthető lineárisnak a mérés a műszer bizonytalansága miatt, így az egyenes illesztéshez le kell vágni az adatok elejét, most vágjuk le, ami kisebb, mint 0.02 %. Meg kell keresni a rugalmassági határ (arányossági határ) felső végét is, vegyük ezt most 0.15 %-nak.

Megjegyzés: a lineáris szakasz végét az ábra alapján állapítottuk meg. Segítségül a 'data cursor' gombra  kattintva lekérdezhetjük az egyes pontok adatait. Válogassuk le logikai indexelést használva a lineáris szakasz pontjait, ahol az x koordináta nagyobb, mint 0.0002 és kisebb, mint 0.0015!




```
> feltetel=and(x>0.0002,x<0.0015);
> xl = x(feltetel);
> yl = y(feltetel);
> hold on;
> plot(xl,yl,'r*');
```

Vizsgáljuk meg, hogy a leválogatott pontok között tényleg lineáris kapcsolat áll-e fent. Ehhez számoljuk ki a lineáris korrelációs együtthatót:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

```
> xs = xl-mean(xl)
> ys = yl-mean(yl)
> r = sum(xs.*ys)/sqrt(sum(xs.^2)*sum(ys.^2)) %  0.9805
```

Vagy ugyanez egyszerűbben a matlab beépített **corr2** parancsával:

```
> r = corr2(xl,yl) %  0.9805
```

Minél jobban közelít a korrelációs tényező abszolút értéke az 1-hez, annál inkább lineáris a kapcsolat a két változó között. Mivel most 0.98 lett ez az érték, a kapcsolatot lineárisnak tekinthetjük és illeszthetünk rá egy egyenest. Ehhez nézzük meg az egyenes egyenletét: $y = m \cdot x + b$. Ebben két ismeretlen található m, az egyenes meredeksége és b eltolás paraméter, ahol az egyenes metszi az y tengelyt. A

leválogatott mérések alapján 75 összetartozó x,y értékünk van, ezek alapján 75 egyenletet tudunk felírni, amelyek lineárisak az ismeretleneket tekintve (m,b):

$$\begin{aligned} m \cdot x_1 + b &= y_1 \\ m \cdot x_2 + b &= y_2 \\ &\vdots \\ m \cdot x_{75} + b &= y_{75} \end{aligned}$$

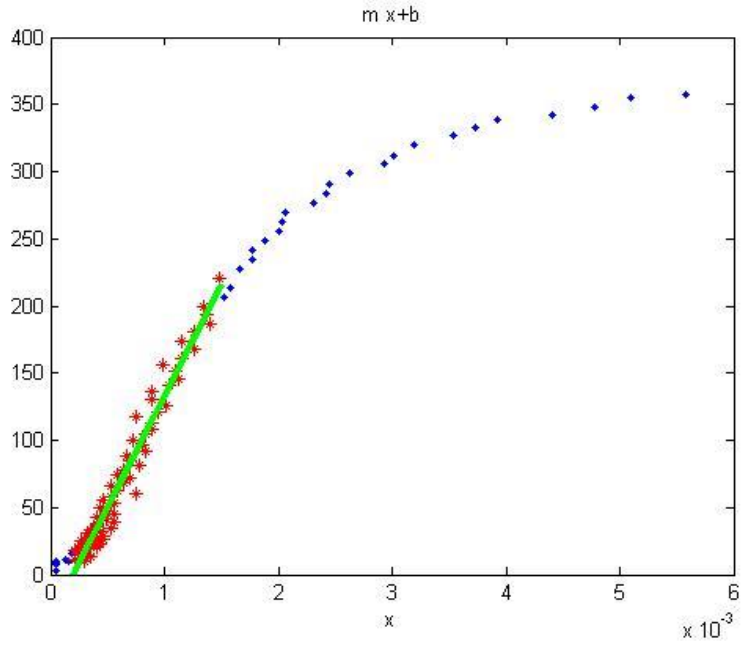
Mátrixos alakban ($A \cdot x = B$):

$$A = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_{75} & 1 \end{pmatrix}; B = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{75} \end{pmatrix}$$

Ez egy túlhatározott egyenletrendszer, ahol a maradék eltérések négyzetösszegének minimalizásával szeretnénk megkapni a legkisebb hibájú megoldást. A korábbi órák alapján ehhez használhatjuk például a túlhatározott esetben QR felbontást alkalmazó x=A\B alakú parancsot, vagy az SVD felbontást használó x=pinv(A)*B-t is.

Először elő kell állítanunk az A alakmátrixot az ismeretlenek együtthatóival. Az első oszlopban m együtthatói lesznek, vagyis x_i értékei (x_i^1), a másodikban pedig b együtthatója, ami mindig 1, ezt írhatjuk az egyszerűség kedvéért x_i^0 alakba is.

```
> A = [xl.^1 xl.^0]
> B = yl;
> mb = A\B
> % egyenes egyenlete
> m = mb(1)
> b = mb(2)
> f = @(x) m*x+b
> hold on;
> fplot(f,[0 0.0015],'g','LineWidth',3);
> axis([0 0.006 0 400])
```



Az illesztett egyenes meredeksége lesz a rugalmassági modulus értéke:

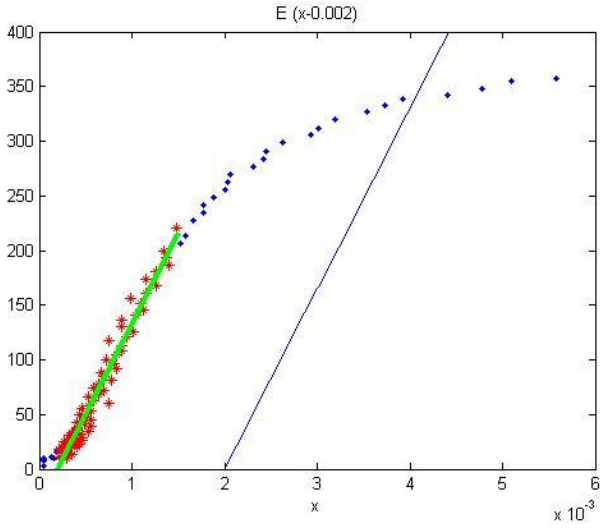
```
> format long
> E = m % E = 1.656261744954783e+05
```

Vagyis a mérés alapján meghatározott E rugalmassági modulus értéke: 165 626 N/mm^2 (MPa).

PARABOLA ILLESZTÉS

A folyáshatár megállapítása ebben az esetben nem egyértelmű az ábrából, ennek az anyagnak nincs jól látható folyáshatára. Ilyenkor az egyezményes folyáshatárt szokás használni, ami a 0.2% maradó alakváltozáshoz tartozó feszültség érték. Ezt a szakítódiagramból úgy lehet meghatározni, hogy 0.2% fajlagos nyúlás értékétől párhuzamos egyenest húznak a lineáris szakasszal, vagyis E meredekséggel kell berajzolni egy egyenest ebből a pontból, és ahol ez metszi a szakítógörbét, ott kell leolvasni a feszültséget. Definiáljuk ezt az egyenest és rajzoljuk be az ábrába!

```
> % 0.2%-os folyáshatár
> fhatar = @(x) E*(x-0.002)
> fplot(fhatar,[0 0.006])
> axis([0 0.006 0 400])
```



Hogyan határozzuk meg ennek az egyenesnek és a szakítógörbének a metszéspontját? Jó lenne illeszteni egy függvényt arra a szakaszra is, ahol a metszéspont is található, és ennek a metszéspontját megkeresni az egyenessel. Ehhez válogassuk le az előzőekhez hasonlóan a x>0.0015 és x<0.006 pontokat:

```
> % 0.0015 és 0.006 közötti
  szakasz
> feltetel2=and(x>0.0015,x<0.006);
> xp = x(feltetel2);
> yp = y(feltetel2);
> plot(xp,yp,'m*')
```

Erre most illesszünk egy másodfokú polinomot, egy parabolát $y = c_0 + c_1 \cdot x + c_2 \cdot x^2$ alakban!

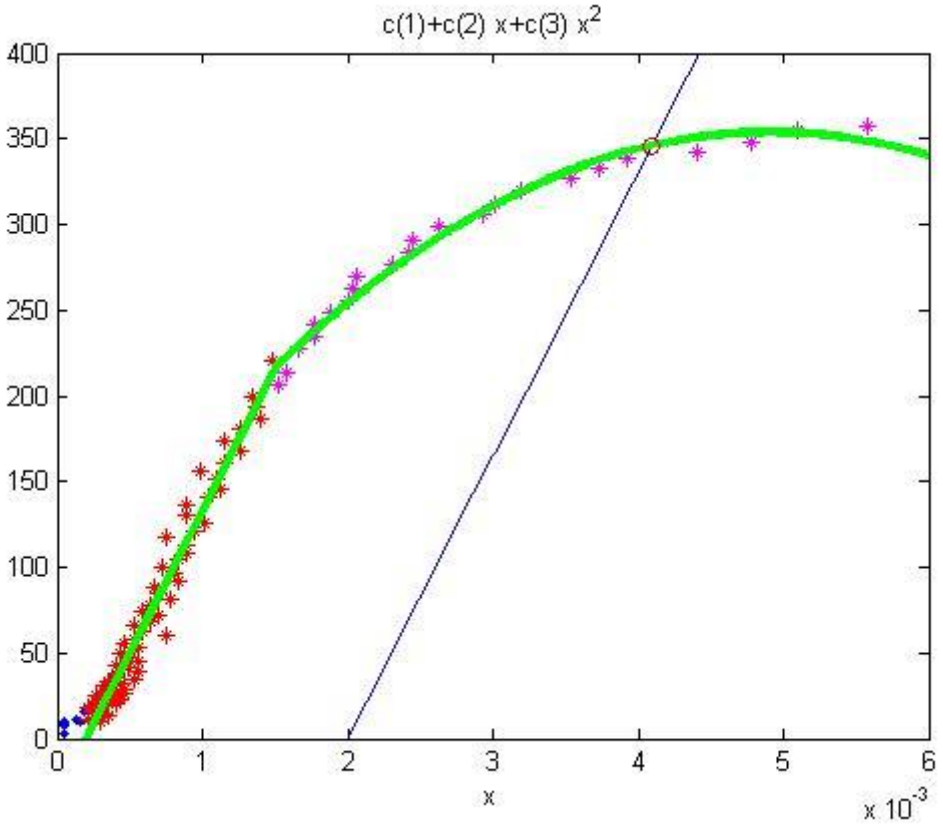
$$y_1 = c_0 + c_1 x_1 + c_2 x_1^2$$
$$y_2 = c_0 + c_1 x_2 + c_2 x_2^2$$
$$\vdots$$
$$y_n = c_0 + c_1 x_n + c_2 x_n^2$$

Mátrixos alakban:

$$A = \begin{pmatrix} 1 & x_1 & {x_1}^2 \\ 1 & x_2 & {x_2}^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & {x_n}^2 \end{pmatrix}; b = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

A parabola (másodfokú polinom) illesztéséhez elő kell állítanunk ismét a megfelelő alakmátrixot (az ismeretlen c_0, c_1 és c_2 együtthatóit), és megoldanunk egy túlhatározott lineáris egyenlet rendszert!

```
> A = [xp.^0 xp.^1 xp.^2]
> b = yp;
> % túlhatározott lin. egy. rsz. megoldása
> c = A\b
> % parabola egyenlete
> f2 = @(x) c(1) + c(2).*x + c(3)*x.^2
> fplot(f2,[0.0015 0.006],'g','LineWidth',3)
> axis([0 0.006 0 400])
```



Most már csak a metszéspontot kell megkeressük. Ezt a korábbi tanulmányaink alapján szintén könnyen megtehetjük. Egy f(x) és egy g(x) függvény metszéspontjában f(x)=g(x). Ezt nullára rendezve a h(x) = f(x)-g(x) = 0 nemlineáris egyenlet gyökeit kell megkeresni, amit megtehetünk például a beépített **fzero** függvényt alkalmazva!

```
> %% 0.2 %-hoz tartozó folyáshatár megállapítása
> % fhatar(x) = f2(x), vagyis h(x) = fhatar(x) - f2(2)=0
> h = @(x) fhatar(x) - f2(x)
> metszes = fzero(h,0.004)
> folyashatar = f2(metszes) % 345.818 MPa
> plot(metszes, folyashatar,'ro')
```

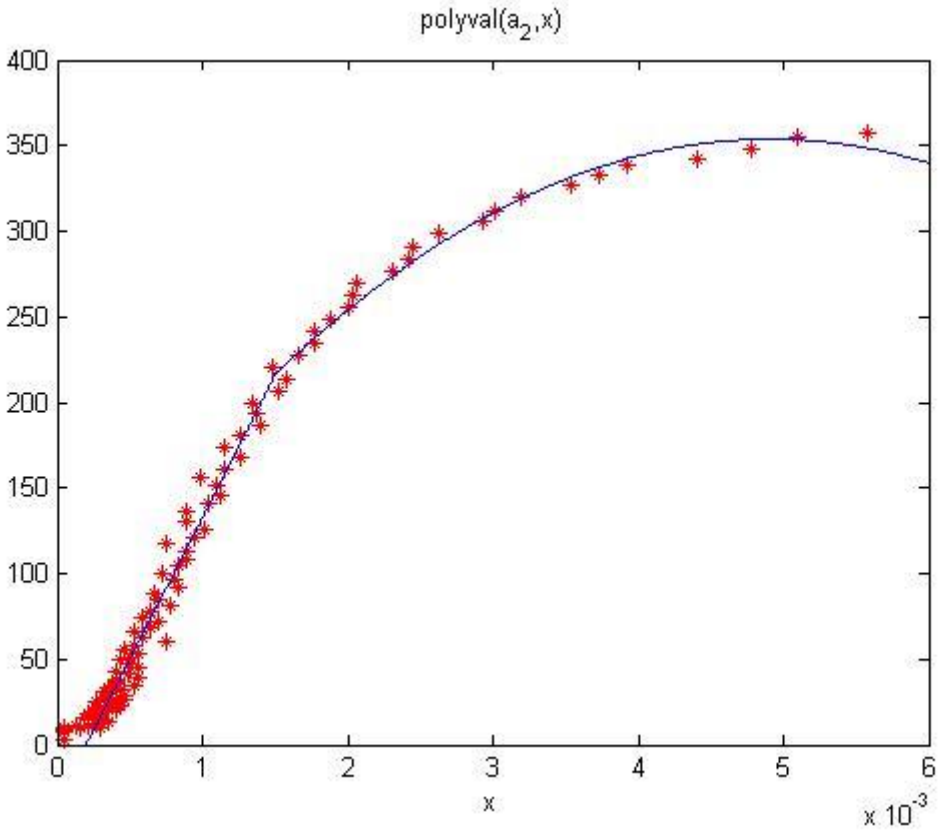
Az egyezményes, 0.2% maradék alakváltozáshoz tartozó folyáshatár a mérésünknél 346 MPa-ra adódott.

Hasonlóan az eddigiekhez harmad, negyed stb. fokú polinomokat is illeszthetünk az adatainkra. A magasabb fokú polinomoknál azonban vigyáznunk kell, mert az alakmátrixunk rosszul kondicionált lesz és bizonytalan lesz a megoldásunk, a mérési pontjainkra lehet, hogy tökéletesen fog illeszkedni a polinom, de közöttük oszcilláció léphet fel. Erre fogunk majd példát látni az interpolációnál.

POLINOM ILLESZTÉS MATLAB BEÉPÍTETT FÜGGVÉNYEIVEL (POLYFIT, POLYVAL)

A rugalmassági modulus meghatározásához a feladat első részében egy egyenes illesztésre volt szükség, ami megfelel egy elsőfokú polinomnak, a feladat második részében pedig másodfokú polinomot illesztettünk. Matlab-ban van egy parancs (**polyfit**), amivel összetartozó pontpárokhoz határozhatjuk meg tetszőleges fokszámú polinom együtthatóit. Az eredménye ennek egy vektor lesz, ami a legkisebb négyzetek módszerével illesztett polinom együtthatóit tartalmazza, a legnagyobb fokú tagtól kezdve visszafelé a konstans tagig. Ennek a parancsnak van egy párja is, a **polyval** parancs, ami kiszámolja egy tetszőleges pontban a polinom értékét, ha megadtuk azt a vektort, ami az együtthatókat tartalmazza. Ez utóbbit meghívhatjuk egy konkrét x értékre, vagy definiálhatjuk függvényként x független változóval. Nézzük meg, hogyan oldhattuk volna meg az előző feladat függvény illesztéseit ezekkel a parancsokkal!

```
> % egyenes illesztése (elsőfokú polinom)
> a1 = polyfit(xl,yl,1)
> p1 = @(x) polyval(a1,x)
> % parabola illesztése (másodfokú polinom)
> a2 = polyfit(xp,yp,2)
> p2 = @(x) polyval(a2,x)
>
> figure(2)
> plot(x,y,'r*'); hold on;
> fplot(p1,[0 0.0015]);
> fplot(p2,[0.0015 0.006])
> axis([0 0.006 0 400])
```



NEMLINEÁRIS REGRESSZIÓ LINEÁRIS ALAKBA ÍRÁSSAL

A valóságban nagyon sok olyan fizikai jelenség van, ahol a mennyiségek közötti kapcsolat nem lineáris. Például a légsűrűséget (ρ) a magasság (h) függvényében exponenciális függvénnyel lehet modellezni: $\rho = k \cdot e^{m\,h}$, egy elejtett tárgy v sebessége a megtett x út függvényében az alábbi függvénnyel írható le: $v^2 = 2\,g\,x$.

Nagyon sok nemlineáris függvény van, de most csak azokkal fogunk foglalkozni, amelyeket át lehet úgy alakítani, hogy egy lineáris egyenletrendszer megoldásával megtaláljuk a paramétereket legkisebb négyzetek módszerét alkalmazva. Ilyen többek között a

- hatványfüggvény: $y = k\,x^m$
- exponenciális függvény: $y = k\,e^{m\,x}$ vagy $y = k\,10^{m\,x}$
- reciprok függvény: $y = \frac{1}{m\,x+c}$

Az algebrai polinomok is ilyenek, ezeknek az illesztését már láttuk az előző példában.

Az a kérdés, hogyan tudjuk a fenti függvényeket lineáris alakba írni?

Általában új változók bevezetésével tudjuk átalakítani a kétváltozós nemlineáris egyenletet, hogy az új változók (amelyek az eredeti változókból levezethetőek) már lineáris kapcsolatban álljanak a keresett paraméterekkel. Nézzünk erre egy példát, hozzuk lineáris alakra a hatványfüggvényt, vegyük mindkét oldal természetes alapú logaritmusát!

$$\ln(y) = \ln(k\,x^m) = m\ln(x) + \ln(k)$$

Vezessünk be új változókat, hogy $Y = c_1\,X + c_2$ lineáris alakra hozzuk az egyenletet. Most legyen $Y = \ln(y)$, $X = \ln(x)$, $c_1 = m$, $c_2 = \ln(k)$:

$$\begin{array}{ccccc} \ln(y) & = & m\ln(x) & + & \ln(k) \\ Y & = & c_1\,X & + & c_2 \end{array}$$

A fenti formában már alkalmazhatjuk a lineáris regressziót, és amint megkaptuk c_1, c_2 értékét az eredeti összefüggés keresett paraméterei könnyen meghatározhatóak:

$$m = c_1, \qquad k = e^{c_2}$$

Sok más nemlineáris egyenlet is lineáris alakba hozható hasonlóan. Nézzük meg erre a bevezetőben említett egyik példát!

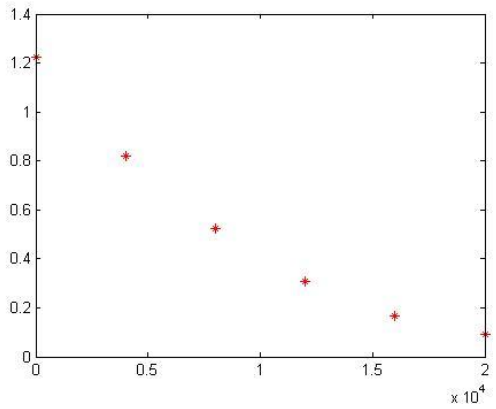
A légsűrűséget (ρ) a magasság (h) függvényében exponenciális függvénnyel lehet modellezni: $\rho = k \cdot e^{m\,h}$. A következő táblázatban különböző magasságokban mért légsűrűség értékek találhatóak:

$h\ [m]$	1	4000	8000	12000	16000	20000
$\rho\ [kg/m^3]$	1.225	0.820	0.525	0.309	0.168	0.092

Lineáris regressziót használva határozzuk meg a legjobban illeszkedő függvényhez a *k* és *m* együtthatókat! Az egyenletet használva mekkora lesz a 8850 m magas Csomolungmán a légsűrűség? Milyen magasan lesz 1 kg/m^3 a légsűrűség? Vizsgáljuk meg lokálisan és globálisan a görbeillesztés hibáit!

A megoldáshoz töltsük be a **legsuruseg.txt** állományt, amiben a fenti adatok vannak.

```
> % légsűrűség a magasság függvényében
> clear all; close all; clc;
> data = load('legsuruseg.txt')
> % p=k*e^(m*h) - exponential function,
  m,k = ?
> h = data(:,1) % magasság
> p = data(:,2) % legsűrűség
> figure(1)
> plot(h,p,'r*')
```



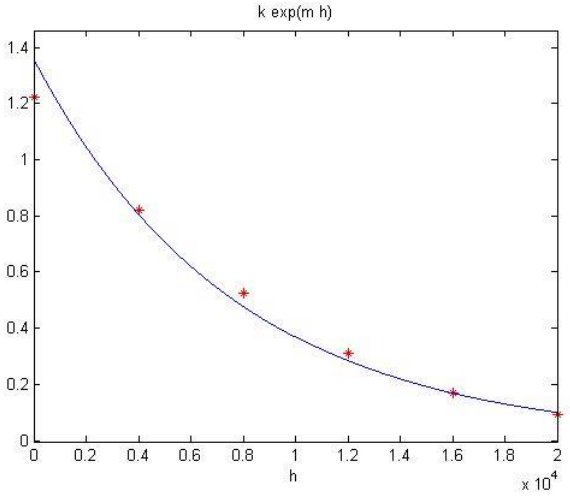
Hozzuk lineáris alakra az exponenciális függvényt! $\rho = k \cdot e^{m\,h}$

$$\begin{aligned} \ln(\rho) &= m\,h + \ln(k) \\ Y &= c_1\,X + c_2 \end{aligned}$$

```
> Y = log(p)
> X = h
> % A*x=b alakban felírva
> A = [X.^1 X.^0]
> b = Y
> % megoldás:
> c = A\b % c1 = m, c2 = ln(k)
```

A keresett paraméterek: $m = c_1,\ k = e^{c_2}$

```
> % a keresett paraméterek
> m = c(1)
> k = exp(c(2))
> % illesztett függvény
> f = @(h) k*exp(m*h)
> % felrajzolva
> hold on;
> fplot(f,[0 20000])
```



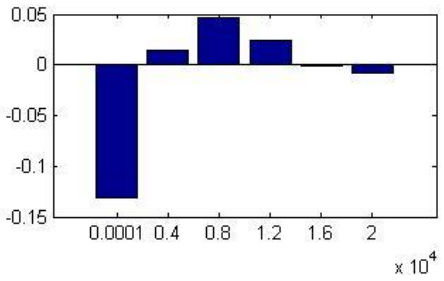
Az egyenletet használva mekkora lesz a 8850 m magas Csomolungmán a légsűrűség? Milyen magasan lesz 1 kg/m^3 a légsűrűség? Az első kérdést egy egyszerű behelyettesítéssel megválaszolhatjuk, a másodikhoz az $f(x) = 1$ egyenletet át kell alakítani $g(x) = f(x) - 1 = 0$ alakra és megkeresni ennek a nemlineáris egyenletnek a gyökeit. Ehhez szükséges egy kezdőértéket is megadni, amit vehetünk az ábrából körülbelül 2000-nek (0.2x10^4).

```
> % legnyomas 8850 m magasban
> p8850 = f(8850)  % 0.4285
>
> % Milyen magasan lesz 1 kg/m^3 a légsűrűség?
> g = @(h) f(h)-1
> h06 = fzero(g,2000) % 2.3415e+03
```

Tehát a Csomolungmán 0.4285 kg/m^3 a levegő sűrűsége, és 2342 m-en lesz pont 1 kg/m^3 a sűrűség.

Nézzük meg a maradék eltérések alakulását! Rajzoljuk fel őket egy oszlopdiagramra, és számítsuk ki az eltérések négyzetösszegét és a maradék eltérések korrigált tapasztalati szórását!

```
> % regresszió maradék hibái
> r = p - f(h)
> % maradék eltérések felrajzolása
> figure(2)
> bar(h,r)
> % hibák négyzetösszege
> S = sum(r.^2) %  0.0203
> % korrigált tapasztalati szórás
> n = length(h) % mérések száma
> np = 2 % becsült paraméterek száma: k, m
> szoras = sqrt(S/(n-np)) % 0.0712
```



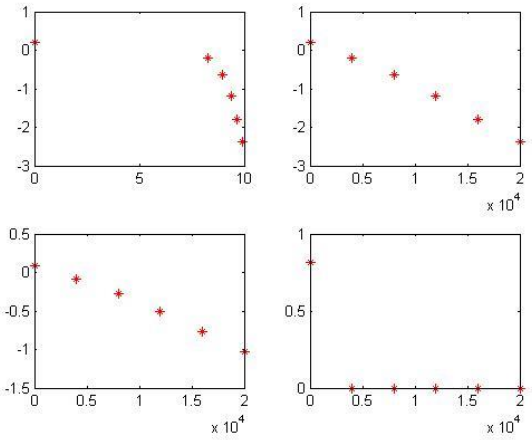
Foglaljunk össze egy táblázatban néhány nemlineáris egyenletet, amit hasonlóképpen megoldhatnánk lineáris regresszióval!

Nemlineáris egyenlet	Lineáris alak	$Y = c_1 X + c_2$ alakban	Keresett paraméterek	Értékek a lineáris regresszióhoz (képe egyenes)
$y = k\,x^m$	$\ln(y) = m\ln(x) + \ln(k)$	$Y = \ln(y), X = \ln(x),$ $c_1 = m,\ c_2 = \ln(k)$	$m = c_1$ $k = e^{c_2}$	$\ln(x), \ln(y)$
$y = k\,e^{m\,x}$	$\ln(y) = mx + \ln(k)$	$Y = \ln(y), X = \mathrm{x},$ $c_1 = m,\ c_2 = \ln(k)$	$m = c_1$ $k = e^{c_2}$	$\mathrm{x}, \ln(y)$
$y = k\,10^{m\,x}$	$\lg(y) = mx + \lg(k)$	$Y = \lg(y), X = \mathrm{x},$ $c_1 = m,\ c_2 = \lg(k)$	$m = c_1$ $k = 10^{c_2}$	$\mathrm{x}, \lg(y)$
$y = \dfrac{1}{m\,x + k}$	$\dfrac{1}{y} = mx + k$	$Y = {}^1\!/_y, X = x,$ $c_1 = m,\ c_2 = \mathrm{k}$	$m = c_1$ $k = c_2$	$x, \dfrac{1}{y}$
$y = \dfrac{m\,x}{x + k}$	$\dfrac{1}{y} = \dfrac{k}{m}\dfrac{1}{x} + \dfrac{1}{m}$	$Y = {}^1\!/_y, X = {}^1\!/_x,$ $c_1 = {}^k\!/_m,\ c_2 = {}^1\!/_m$	$m = {}^1\!/_{c_2}$ $k = c_1/c_2$	$\dfrac{1}{x}, \dfrac{1}{y}$

NEMLINEÁRIS EGYENLET TÍPUSÁNAK KIVÁLASZTÁSA[10]

Az előző feladatban rendelkezésünkre állt egy modell, hogy milyen alakú összefüggés áll fent a mennyiségek között. Előfordulhat azonban olyan eset is, amikor nem ismerjük a kapcsolatot leíró függvény alakját. Hogyan választhatjuk ki a megfelelő illesztendő nemlineáris egyenletet ilyen esetben? Célszerű felrajzolni a pontokat és a fenti táblázat utolsó oszlopában lévő értékeket. Ha van olyan, amelyik esetében a pontok nagyjából egy egyenes mentén helyezkednek el, akkor azt a függvény típust válasszuk a regresszióhoz! Néhányat ábrázoljunk az előző példához ezek közül! Matlab-ban a természetes alapú logaritmus a **log** függvény, 10-es alapú logaritmus a **log10** függvény, exponenciális függvény pedig az **exp** függvény. Az ábrázoláshoz használjuk a **subplot** függvényt, amivel egy ábrára több dolgot is fel tudunk rajzolni. A parancsot a **subplot(n,m,i)** formában hívhatjuk meg, ahol n a sorok, m az oszlopok száma, i pedig az adott rajz sorszáma balról jobbra és fentről le számolva.

```
> figure(3)
> x=h; y = p;
> subplot(2,2,1)
> plot(log(x), log(y), 'r*')
> subplot(2,2,2)
> plot(x,log(y),'r*')
> % egyenes lett!
> subplot(2,2,3)
> plot(x,log10(y),'r*')
> % egyenes lett!
> subplot(2,2,4)
> plot(x,1/y,'r*')
```



A fenti rajzon a második és a harmadik képen lett a pontok képe közelítőleg egyenes, amikor x és $\ln(y)$ illetve $\log(y)$ lett ábrázolva. A táblázatra ránézve látszódik, hogy abban az esetben, ha nem ismernénk a függvénykapcsolatot, akkor is exponenciális függvény illesztésével lenne érdemes próbálkozni.

A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

axis	-	Tengelyek minimális, maximális értékeinek megadása
mean	-	Vektor elemeinek számtani közepe, átlaga
sum	-	Vektor elemeinek összege
corr2	-	Lineáris korrelációs együttható
polyfit	-	Megadott fokszámú polinom illesztése az adatokra
polyval	-	Együttható vektorral megadott polinom értékének kiszámítása
bar	-	Ábrázolás oszlopdiagrammon
subplot	-	Egy grafikus ablakon belül több ábra

[10] Otthoni átnézésre

9. GYAKORLÓ FELADATOK 1.

LINEÁRIS EGYENLET RENDSZEREK

1. Határozza meg az alábbi egyenletrendszer megoldását SVD felbontással!

$$\begin{aligned} 2x + 3y + 4z &= 5 \\ 3x + 5y + 6z &= 15 \\ 4x + 6y + 9z &= 7 \end{aligned}$$

Ellenőrizze, hogy van-e megoldása a feladatnak és egyértelmű-e? Hozza létre az U,S,V mátrixokat SVD felbontással! Ellenőrizze a felbontás helyességét! Állítsa elő az inverzet az U,S,V mátrixokat használva és ezzel oldja meg az egyenletrendszert! Határozza meg megoldás abszolút hibáját skalár alakban!

```
> clear all; clc;
> A = [2 3 4;
>      3 5 6;
>      4 6 9]
> b = [5; 15; 7]
>
> % Van-e megoldás?
> rank(A) % 3
> rank([A b]) % 3, egyenlőek, tehát van megoldás
> % és r(A) egyenlő az oszlopok számával is, tehát egyértelmű a
  megoldás
>
> %  Az M=U*S*V' felbontásnak megfelelően inv(M)=V*inv(S)*U'
> [U,S,V]=svd(A)
>
> %  Ellenőrzés
> norm(A-U*S*V') % 4.5513e-15 - jó a felbontás
> U'*U-eye(3)
> V'*V-eye(3)
> % U és V ortonormáltak
>
> %  Az inverz előállítása
> invS=diag(1./S);
> invA=V*diag(invS)*U'
>
> %  A megoldás
> x1=invA*b % x1 = -14.0000, 15.0000, -3.0000
>
> %  Beépített SVD felbontás függvénnyel: pinv
> x2=pinv(A)*b %  x2 = -14.0000, 15.0000, -3.0000
>
> % Hiba
> norm(A*x1-b) % 5.1361e-14
```

2. Határozza meg az alábbi egyenletrendszer megoldását LU felbontással!

$$\begin{aligned} -x + y &= 3 \\ 2x - y + z &= 4 \\ x + 3y + 2z &= 5 \end{aligned}$$

Ellenőrizze, hogy van-e megoldása a feladatnak és egyértelmű-e? Hozza létre az L és U mátrixokat! Ellenőrizze a felbontás helyességét! Oldja meg az egyenletrendszereket, kihasználva azok mátrixainak speciális voltát! Határozza meg megoldás abszolút hibáját skalár alakban!

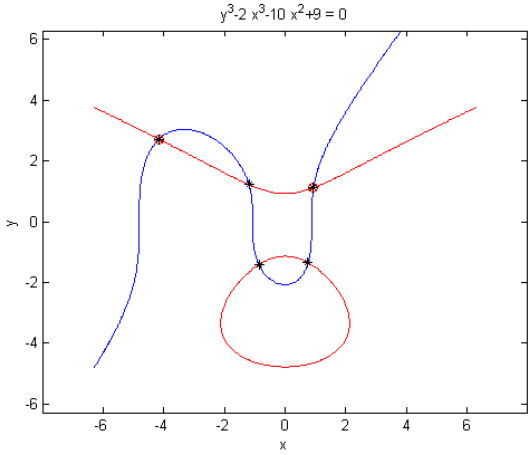
```
> clear all; clc;
> A = [-1 1 0;
>      2 -1 1;
>      1 3 2]
> b = [3; 4; 5]
>
> % Van-e megoldás?
> rank(A) % 3
> rank([A b]) % 3, egyenlőek, tehát van megoldás
> % és r(A) egyenlő az oszlopok számával is, tehát egyértelmű a
  megoldás
>
> %  LU felbontás alsó (L), felső (U) és permutációs (P) mátrix-ra
> [L U P]=lu(A)
>
> %  A felbontás helyességének ellenőrzése
> L*U-P*A
> norm(L*U-P*A)
>
> %  Az L*U*x=P*b egyenletrendszer megoldása 2 lépésben
> %  2.1 L*y=P*b megoldása y-ra
> %      úgy, hogy kihasználjuk L alsó mátrix tulajdonságát
> opts.UT=false;
> opts.LT=true;
> y1=linsolve(L,P*b,opts) %  y1 = 4.0000, 3.0000, 4.5714
>
> %  2.2 U*x=y megoldása x-re
> %      úgy, hogy kihasználjuk U felső mátrix tulajdonságát
> opts.UT=true;
> opts.LT=false;
> x1=linsolve(U,y1,opts) % x1 = -9, -6, 16
>
> % Hiba
> norm(A*x1-b)
```

3. Határozza meg az alábbi egyenletrendszer megoldásait!

$$y^3 - 3 \cdot x^2 + 5 \cdot y^2 - 5 = 0$$
$$y^3 - 2 \cdot x^3 - 10 \cdot x^2 + 9 = 0$$

Ábrázolja a feladatot grafikusan a gyökök közelítő helyének meghatározása érdekében! Határozza meg a legkisebb és a legnagyobb x koordinátájú valós gyökeit a Matlab beépített numerikus függvényével 10^{-6} pontossággal! Ábrázolja a megoldásokat a korábbi ábrán! Határozza meg az összes megoldást a Matlab beépített szimbolikus egyenletrendszer megoldó függvényével! Van komplex gyök? Ha igen mennyi? Ábrázolja az összes valós megoldást!

```
> %% 3. feladat
> clc; clear all; close all;
> % Az egyenletrendszer
> disp('Egyenlet rendszer')
> eq1 = @(x,y) y^3 - 3*x^2 + 5*y^2 - 5
> eq2 = @(x,y) y^3 - 2*x^3 - 10*x^2 + 9
> f = @(x,y) [eq1(x,y); eq2(x,y)];
> figure(1); clf;
> ezplot(eq1)
> hold on
> ezplot(eq2)
> axis equal
> F = @(x) f(x(1),x(2));
> x0 = [-4; 3]
> x1 = fsolve(F,x0,
  optimset('TolFun',1e-6))
> plot(x1(1),x1(2),'ro')
> % x1 = -4.1612
> %       2.7166
> x0 = [2; 2]
> x2 = fsolve(F,x0,
  optimset('TolFun',1e-6))
> % x1 = 0.9356
> %      1.1166
> plot(x2(1),x2(2),'ro')
> % Definiáljuk szimbolikusan az egyenletrendszert!
> fs = sym('[y^3 - 3*x^2 + 5*y^2 - 5;  y^3 - 2*x^3 - 10*x^2 + 9]')
> % Megoldás solve-val
> s=solve(fs)
> %     x: [9x1 sym]
> %     y: [9x1 sym]
> % Az eredmények kiíratása double típusú számként
> [double(s.x) double(s.y)]
> %   -4.1612 + 0.0000i   2.7166 + 0.0000i
> %   -1.1917 + 0.0000i   1.2202 + 0.0000i
> %   -0.8608 + 0.0000i  -1.4205 + 0.0000i
> %    0.7522 + 0.0000i  -1.3556 + 0.0000i
> %    0.9356 + 0.0000i   1.1166 + 0.0000i
> %   -4.4726 + 1.8307i  -2.8328 + 3.2770i
> %   -4.4726 - 1.8307i  -2.8328 - 3.2770i
> %    1.4855 + 3.3081i  -5.8058 + 0.6924i
> %    1.4855 - 3.3081i  -5.8058 - 0.6924i
> plot(s.x(1:5),s.y(1:5),'k*')
```

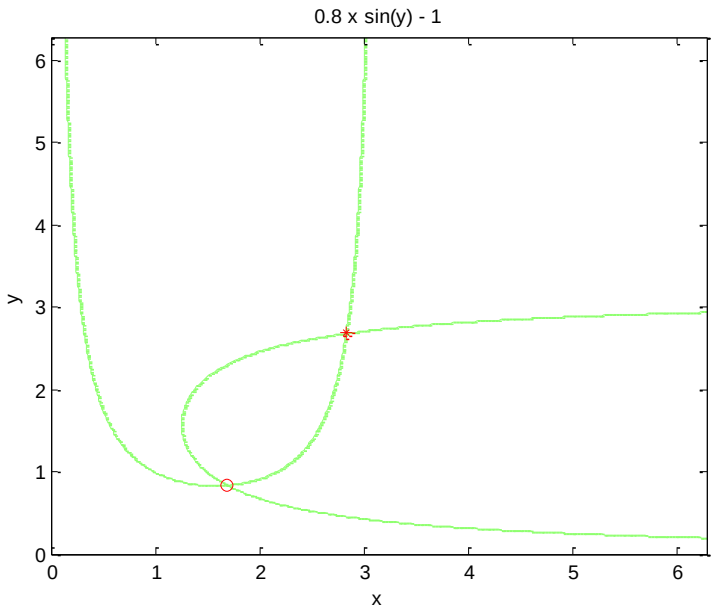


4. Adott az alábbi egyenletrendszer:

$$1{,}2\sin(x) \cdot y = 1$$
$$0{,}8\sin(y) \cdot x = 1$$

Ábrázolja az egyenletrendszert az $x = 0\ldots2\pi$, $y = 0\ldots2\pi$tartományon! Határozza meg az egyenletrendszer megoldásait a fenti tartományon a Matlab beépített numerikus módszerével! Ellenőrizze a megoldásokat visszahelyettesítéssel! Ábrázolja a megtalált megoldásokat!

```
> clear all; clc; close all;
>
> f1s = sym('1.2*sin(x)*y-1');
> f2s = sym('0.8*sin(y)*x-1');
>
> %f1 = @(x,y) 1.2*sin(x)*y-1
> %f2 = @(x,y) 0.8*sin(y)*x-1
>
> f1 = @(x,y) eval(f1s);
> f2 = @(x,y) eval(f2s);
>
> f = @(x) [f1(x(1), x(2));  f2(x(1), x(2))];
>
> clf;
> ezplot(f1s, [0 2*pi])
> hold on;
> ezplot(f2s, [0 2*pi])
>
> opt = optimset('Display', 'iter');
> x1 = fsolve(f, [1.6 0.8], opt);
> x2 = fsolve(f, [2.8 2.7], opt);
>
> plot(x1(1), x1(2), 'ro')
> plot(x2(1), x2(2), 'r*')
>
> x1
> x2
>
> % x1 =
> %     1.6810    0.8384
> % x2 =
> %     2.8258    2.6834
>
> f(x1)
> f(x2)
>
> % ans =
> %    1.0e-10 *
> %    -0.0152
> %    -0.8089
> % ans =
> %    1.0e-08 *
> %    -0.7832
> %    -0.8536
```



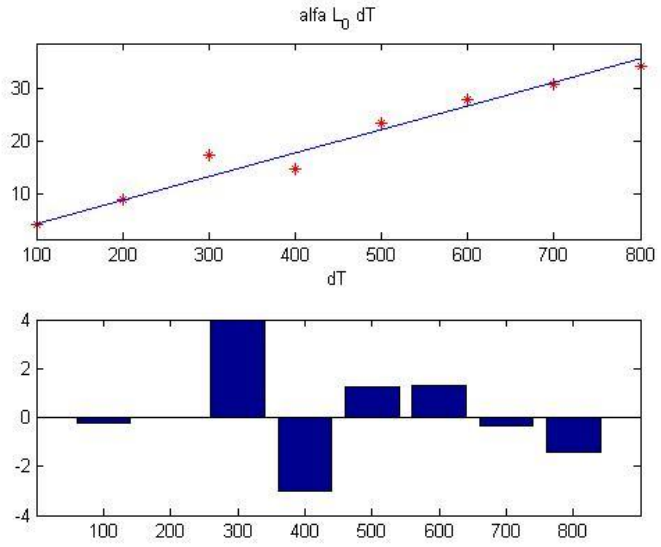
REGRESSZIÓ

5. Egy kísérletben méréseket végeztek a hőtágulási együttható meghatározására, ehhez egy 2.5 m hosszú rozsdamentes acélrudat sütőbe helyeztek. 20°C-tól kezdve 100°-onként növelték a hőmérsékletet egészen 820°C-ig, és mérték a hossz növekedését. ΔL hossznövekedés arányos ΔT hőmérséklet változással, az alábbi összefüggés szerint: $\Delta L = \alpha L_0 \Delta T$, ahol α a hőtágulási együttható, L_0 pedig a kezdeti hossz. Határozzuk meg lineáris regresszióval a hőtágulási együttható értékét az alábbi mérési eredmények alapján!

ΔT [°C]	100	200	300	400	500	600	700	800
ΔL [mm]	4.2	8.9	17.3	14.8	23.5	28	30.8	34.2

Rajzolja fel a regressziós egyenest és a maradék eltéréseket is egymás alá!

```
>  % hotagulas dL=alfa*L0*dT
>  clear all; close all; clc;
>
>  dT = 100:100:800; dT=dT'
>  dL = [4.2; 8.9; 17.3; 14.8; 23.5; 28; 30.8; 34.2]
>  L0 = 2500
>
>  figure(1)
>  subplot(2,1,1)
>  plot(dT, dL, 'r*')
>
>  % egyenletrendszer felírása
>  A = L0*dT
>  b = dL
>
>  alfa = A\b % 1.7800e-05
>  f = @(dT) alfa*L0*dT
>
>  hold on
>  ezplot(f,[min(dT),max(dT)])
>
>  e = dL-f(dT)
>  subplot(2,1,2)
>  bar(dT,e)
```

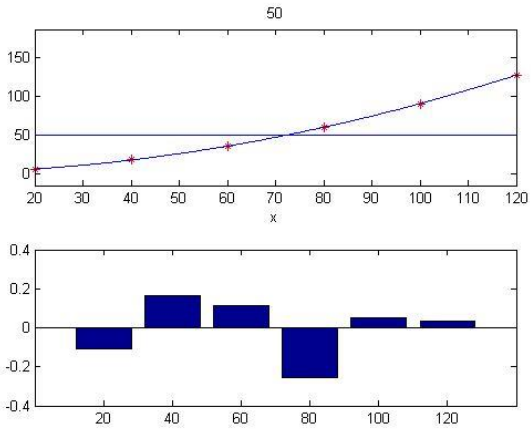


6. Nedves úton mérték egy autó féktávolságát a sebesség függvényében:

v [km/h]	20	40	60	80	100	120
d [m]	6	18	36	60	91	128

Határozza meg az adatokra legjobban illeszkedő másodfokú polinomot! Ábrázolja az adatokat és az illesztett polinomot egy ábrán, alatta egy külön ábrán pedig a maradék ellentmondásokat. Mekkora a maradék eltérések négyzetösszege, várható értéke és a korrigált empirikus szórása? Mennyi lesz a féktávolság 70 km/h-nál? Mekkora sebességnél lesz a féktávolság pont 50 m?

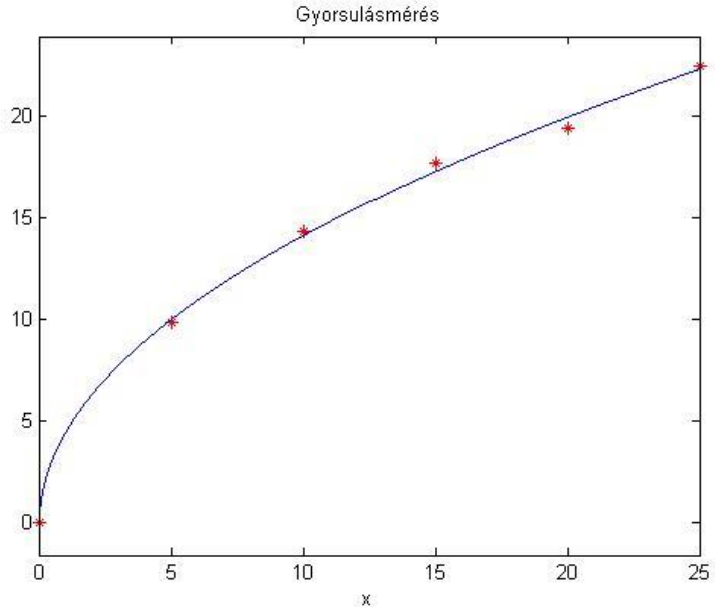
```
% Féktávolság nedves úton
clear all; clc; close all;

v = [20; 40; 60; 80; 100; 120]
d = [6; 18; 36; 60; 91; 128]

figure(1)
subplot(2,1,1)
plot(v,d,'r*')

% másodfokú polinom illesztése
% d = a0 + a1*v + a2*v^2
% A*x=b alakban
A = [v.^0 v.^1 v.^2]
b = d
x = A\b % 0.7000  0.1123  0.0079
f = @(v) x(1) + x(2)*v + x(3)*v.^2
hold on; ezplot(f,[0 140])
% más megoldás
a = polyfit(v,d,2) %  0.0079    0.1123    0.7000
f2 = @(v) polyval(a,v)

% féktávolság 70 km/h esetén
f(70) % 47.2813 m
% sebesség 50 m féktávolság esetén
ezplot('50',[min(v), max(v)])
x0 = 70 % kezdőérték az ábrából
f50 = @(x) f(x)-50;
v50 = fzero(f50,x0) %  72.1997

% hibák
r = d - f(v)
subplot(2,1,2)
bar(v,r)

% hibák négyzetösszege
S = sum(r.^2) %   0.1214
% korrigált tapasztalati szórás
n = length(v) % mérések száma
np = 3 % becsült paraméterek száma: a0,a1,a2
szoras = sqrt(S/(n-np)) %   0.2012
```



7. A nehézségi gyorsulás méréséhez a következő kísérletet végezték. Ledobtak egy labdát egy 30 m magas épületről, esés közben több helyen mérték a sebességét az épületre erősített szenzorokkal. A *v* sebesség a megtett *x* út között következő kapcsolat áll fenn: $v^2 = 2gx$, ahol g a nehézségi gyorsulás. A mért adatok a következőek:

x [m]	0	5	10	15	20	25
v [m/s]	0	9.85	14.32	17.63	19.34	22.41

```
> % nehézségi gyorsulás meghatározása
> % v^2 = 2*g*x
> % Új változók: Y=v^2, X=x; a1 = 2*g
> % Y = a1*X
> clear all; close all; clc;
> x = [0; 5; 10; 15; 20; 25]
> v = [0; 9.85; 14.32; 17.63; 19.34; 22.41]
> figure(1)
> plot(x,v,'r*')
> Y = v.^2; X = x;
> % lineáris egyenletrendszer megoldása
> a1 = X\Y % 19.8065
> % nehézségi gyorsulás
> g = a1/2 % 9.9032
> % tényleges érték: 9.81
> f =@(x) sqrt(2*g*x)
> hold on;
> ezplot(f,[0 25])
> title('Gyorsulásmérés')
```

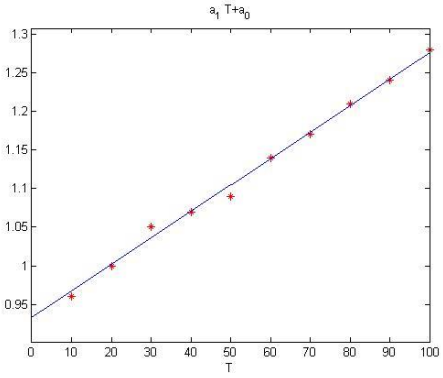


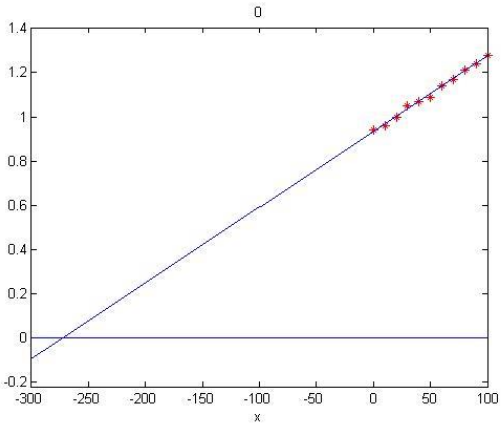
8. Az abszolút 0 fok meghatározására végeztek kísérletet. A kísérletben egy tartályban lévő gázt jeges vízbe merítettek (0°C), megmérték a gáz nyomását, majd 10 fokonként emelték a hőmérsékletet, mérve a nyomást is. A Gay-Lussac gáztörvény szerint állandó nyomáson egy adott tömegű gáz térfogata az abszolút hőmérsékletével egyenes arányban változik. Lineáris kapcsolat van a nyomás és a hőmérséklet között állandó térfogat esetén. A mért eredmények a következőek:

T [°C]	0	10	20	30	40	50	60	70	80	90	100
p [atm.]	0.94	0.96	1.0	1.05	1.07	1.09	1.14	1.17	1.21	1.24	1.28

Extrapolációval határozzuk meg az abszolút 0 fokot, ahol a nyomás nulla lesz!

```
clear all; close all; clc;
T = 0:10:100; T = T';
p = [0.94; 0.96; 1.0; 1.05; 1.07; 1.09; 1.14; 1.17; 1.21; 1.24; 1.28]
figure(1);
plot(T,p,'r*')
% p = a1*T + a0 (y=m*x+b egyenes egyenlete)
A = [T.^1 T.^0]; b = p
x = A\b
a1 = x(1) % 0.0034
a0 = x(2) % 0.9336

f = @(T) a1*T+a0
hold on;
ezplot(f,[min(T), max(T)])
% hibák
e = p - f(T)
norm(e)

% más megoldás beépített függvénnyel: polyfit, polyval
x2 = polyfit(T,p,1) % 0.0034 0.9336
e2 = p - polyval(x2,T)
norm(e2)
f2 = @(T) polyval(x2,T)

% Abszolút nulla fok megkeresése extrapolációval
figure(2)
plot(T,p,'r*'); hold on;
ezplot(f,[-300, 100])
ezplot('0',[-300, 100])

absz_nulla = fzero(f,-250)
%  -273.1383
% vagy átrendezve f=0-t: T = -a0/a1
absz_nulla = -a0/a1 %  -273.1383
% tényleges absz. nulla: 273.15°C

% ellenőrzés
f(absz_nulla)
f2(absz_nulla)
```





NUMERIKUS MÓDSZEREK MINTA ZÁRTHELYI[11]

A feladatokat Matlab (*.m vagy *.mlx) fájlban dolgozza ki, és lássa el kommentekkel, hogy más számára is követhető legyen és egyértelműek legyenek a kérdésekre a válaszok.

1. Határozza meg az alábbi egyenletrendszer megoldásait!

$$y^3 - 3 \cdot x^2 + 5 \cdot y^2 - 5 = 0$$
$$y^3 - 2 \cdot x^3 - 10 \cdot x^2 + 9 = 0$$

Ábrázolja a feladatot grafikusan a gyökök közelítő helyének meghatározása érdekében! Határozza meg a legkisebb és a legnagyobb x koordinátájú valós gyökeit a Matlab beépített numerikus függvényével 10^{-6} pontossággal! Ábrázolja a megoldásokat a korábbi ábrán! Határozza meg az összes megoldást a Matlab beépített szimbolikus egyenletrendszer megoldó függvényével! Van komplex gyök? Ha igen mennyi? Ábrázolja az összes valós megoldást!

2. Nedves úton mérték egy autó féktávolságát a sebesség függvényében:

v [km/h]	20	40	60	80	100	120
d [m]	6	18	36	60	91	128

Határozza meg az adatokra legjobban illeszkedő másodfokú polinomot! Ábrázolja az adatokat és az illesztett polinomot egy ábrán, alatta egy külön ábrán pedig a maradék ellentmondásokat. Mekkora a maradék eltérések négyzetösszege, várható értéke és a korrigált empirikus szórása? Mennyi lesz a féktávolság 70 km/h-nál? Mekkora sebességnél lesz a féktávolság pont 50 m?

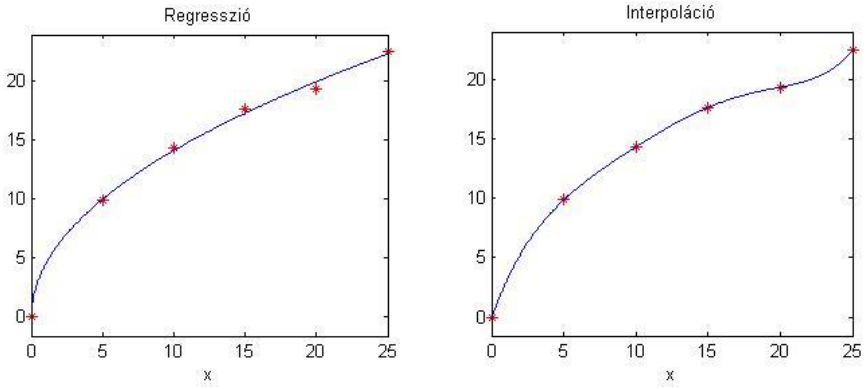
20 és 120 km/h között számolja ki kerek 5 méterenként a sebességekhez tartozó féktávolságokat és ezeket mentse ki egy szöveges állományba, egész számként a sebességeket és két tizedesjegy pontossággal a féktávolságokat!

[11] Megoldásokat lásd a gyakorló feladatok között

10. INTERPOLÁCIÓ

A korábbi gyakorlaton regresszióval, függvényillesztéssel foglalkoztunk, amikor meghatároztuk az adott pontokra legjobban illeszkedő egyenest, parabolát, exponenciális függvényt stb. Az illesztett függvény többnyire nem megy át a mért pontokon, de ideális esetben közel halad hozzájuk.

Az interpoláció egy olyan matematikai összefüggés, ami a megadott pontokat tökéletesen reprezentálja, visszaadja ezekben a pontokban a mérési értékeket, a pontok között pedig becslésre használható. Grafikusan ábrázolva a függvény átmegy a mérési pontokon.



INTERPOLÁCIÓ EGYETLEN POLINOMMAL

Egy polinom általános alakja:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

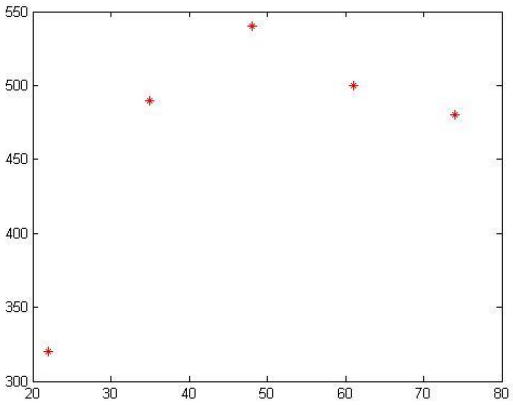
Az $a_n, a_{n-1}, \cdots, a_1, a_0$ együtthatók valós számok, n pedig egy nem negatív egész szám, a polinom fokszáma. Az elsőfokú polinom egy lineáris függvénykapcsolat, képe egyenes, a másodfokú polinom (parabola) és a magasabb fokú polinomok képei pedig valamilyen görbék, minél magasabb a fokszám, annál több 'kanyar', inflexiós pont lehet benne, annál bonyolultabb lehet a képe.

Ha *n* pontból álló adatállományunk van, akkor ezt különböző fokszámú polinomokkal lehet közelíteni, egészen *(n-1)* fokszámig. Alacsonyabb fokszámú polinom illesztése esetén regresszióról beszélhetünk, *(n-1)* fokú polinommal pedig interpolációt kapunk, ekkor a polinom minden ponton keresztül halad. Ez lesz a polinomiális interpoláció esete. Ez egy **globális interpoláció**, mivel az összes adatra egyetlen függvényt illesztünk. Nézzünk egy példát rá!

A szélerőművek teljesítménye a szélsebességgel változik. Egy kísérletben a következő 5 mérési eredményt kaptuk erre vonatkozóan:

Szélsebesség [km/h]	22	35	48	61	74
Teljesítmény [W]	320	490	540	500	480

Az 5 mérési eredményre negyedfokú interpolációs polinom illeszthető. Interpolációval határozzuk meg a szélerőmű teljesítményét 42 és 68 km/h-s szél esetén! Mekkora szélsebességnél lesz a teljesítmény 400 W? Ehhez töltsük be a szeleromu.txt állományt!




```
> % szélerőmű adatai
> clear all; close all; clc;
> data = load('szeleromu.txt')
> v = data(:,1) % szélsebesség
> p = data(:,2) % teljesítmény
> figure(1)
> plot(v,p,'r*')
```

A negyedfokú interpolációs polinom $f(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$ együtthatóit kiszámolhatjuk egy lineáris egyenletrendszert megoldva, a regressziónál már megismert módon, 5 egyenletet felírva az öt pontra. Az *A* együttható mátrixot **Vandermonde mátrix**nak is nevezik.

$$\begin{aligned} y_1 &= a_4x_1^4 + a_3x_1^3 + a_2x_1^2 + a_1x_1 + a_0 \\ y_2 &= a_4x_2^4 + a_3x_2^3 + a_2x_2^2 + a_1x_2 + a_0 \\ y_3 &= a_4x_2^4 + a_3x_2^3 + a_2x_2^2 + a_1x_2 + a_0 \\ y_4 &= a_4x_3^4 + a_3x_3^3 + a_2x_3^2 + a_1x_3 + a_0 \\ y_5 &= a_4x_4^4 + a_3x_4^3 + a_2x_4^2 + a_1x_4 + a_0 \end{aligned} \quad \rightarrow A = \begin{pmatrix} {x_1}^4 & {x_1}^3 & {x_1}^2 & x_1 & 1 \\ {x_2}^4 & {x_2}^3 & {x_2}^2 & x_2 & 1 \\ {x_3}^4 & {x_3}^3 & {x_3}^2 & x_3 & 1 \\ {x_4}^4 & {x_4}^3 & {x_4}^2 & x_4 & 1 \\ {x_5}^4 & {x_5}^3 & {x_5}^2 & x_5 & 1 \end{pmatrix}; b = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}$$

Vagy használhatjuk itt is a **polyfit, polyval** beépített Matlab függvényeket. Most az egyszerűség kedvéért használjuk ez utóbbiakat!

```
> a = polyfit(v,p,4) %  0.0001   -0.0171    0.5627   12.0190  -62.0517
```

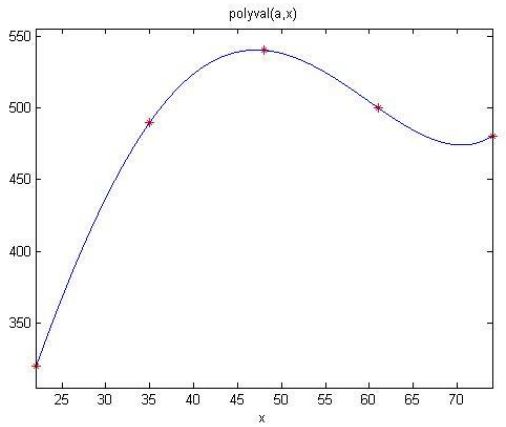
A polyfit-et használva megadhatjuk az illeszteni kívánt polinom fokszámát, és megkapjuk a polinom együtthatóit a legmagasabb fokú tagtól visszafelé a konstans tagig: $a_4 = 0.0001$; $a_3 = -0.0171$; $a_2 = 0.5627$; $a_1 = 12.0190$; $a_0 = -62.0517$.

A polyval parancs a megadott együtthatókból ki tudja számolni egy tetszőleges helyen a polinom értékét, pl. a kérdéses 42 km/h-nál:

```
> polyval(a,42) % 531.7853
```

Vagy definiálhatjuk az együtthatókkal a polinom függvényét is, és akkor ábrázolni is könnyen tudjuk:

```
> fp = @(x) polyval(a,x)
> fp(68) % 476.5008
> hold on;
> fplot(fp,[min(v) max(v)])
```



Ahhoz, hogy megkapjuk, hogy mekkora szélsebességnél lesz a teljesítmény 400 W, egy nemlineáris egyenletet kell megoldanunk! A megoldandó nemlineáris egyenletünk, nullára rendezve: $a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 - 400 = 0$. Szükség lesz egy kezdőértékre is, amit az ábrából vehetünk.

```
> h = @(x) fp(x)-400
> x200 = fzero(h,30) % 27.1296
```

Tehát 42 km/h-s szélnél 532 W a teljesítmény, 68 km/h-nál pedig 477 W. 400 W-os teljesítményt pedig 27 km/h-s szélnél kapunk.

A sztenderd alakba írt *n*-ed fokú polinom $f(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ illesztéséhez *(n+1)* lineáris egyenletből álló egyenletrendszert kell megoldani. Az egyenletekhez az összes rendelkezésre álló pont koordinátáit be kell helyettesíteni az általános alakba. Gyakorlatban, különösen magasabb fokú polinomok esetében nem hatékony megoldani ezt az egyenletrendszert, gyakran rosszul kondicionált lesz az együttható mátrixunk. Nézzük meg az előző példánk együttható mátrixának kondíciószámát! Ehhez írjuk fel az együttható mátrixot, ami a lineáris egyenletrendszer megoldásához kellene. A felírást megtehetjük a regressziónál megismert módon is, de a Matlabnak van egy külön parancsa a Vandermonde mátrix előállítására, azt is használhatjuk, ugyanaz lesz az eredmény:

```
> A = [v.^4 v.^3 v.^2 v.^1 v.^0] % hagyományos felírás
> A = vander(v) % másik megoldás: Vandermonde mátrixként
> cond(A) %  1.5378e+09
```

Ez a szám már most is nagyon nagy (10^9 nagyságrendű), pedig csak 4-ed fokú, vagyis nem is nagyon magas fokszámú polinomot illesztettünk. Ellenőrzésképp megnézhetjük, a lineáris egyenletrendszer megoldását, hogy ugyanazt adja-e, mint a polyfit!

```
> x = A\p % 0.0001; -0.0171; 0.5627; 12.0190; -62.0517
```

LAGRANGE ÉS NEWTON INTERPOLÁCIÓS POLINOMOK

Adott pontokon keresztül egy interpolációs polinom írható fel, ez viszont többféle alakban is megadható, nézzünk meg az általános alak mellett röviden két másikat, amit gyakran célszerűbb használni, könnyebb felírni és nem lesz rosszul kondicionált a feladat. Az egyik ilyen alak a Lagrange interpolációs polinom, a másik a Newton.

LAGRANGE INTERPOLÁCIÓS POLINOM

Ez a fajta polinom mindenféle számítás, egyenletrendszer megoldás nélkül, a pontok koordinátáiból felírható, a következő alakban *n* pontra:

$$f(x) = \sum_{i=1}^{n} y_i L_i(x) = \sum_{i=1}^{n} y_i \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

ahol $L_i(x) = \prod_{\substack{j=1 \\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)}$ -t Lagrange függvényeknek hívjuk. Két pontra felírva:

$$f(x) = \frac{(x - x_2)}{(x_1 - x_2)} y_1 + \frac{(x - x_1)}{(x_2 - x_1)} y_2$$

Három pontra felírva:

$$f(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3$$

- Látszik, hogy a pontok koordinátáit használva, előzetes számítások nélkül is felírható az interpolációs polinom.
- Nehézkes vele dolgozni, minden egyes interpolálandó pontnál fel kell írjuk újra az adott pontra az egész egyenletet, nem elég csak az együtthatókat behelyettesíteni, mint az általános alaknál.
- Ha a ponthalmazunk új ponttal bővül, akkor az összes Lagrange függvényt újra kell számolni, ebben különbözik a Newton formától, ahol új pontok esetén csak az új tagokat kell kiszámolni.

NEWTON-FÉLE INTERPOLÁCIÓS POLINOM[12]

A Newton-féle polinom az ún. osztott differenciák segítségével írható fel, rekurzív úton. Általános alakja a polinomnak:

$$f(x) = a_1 + a_2(x - x_1) + a_3(x - x_1)(x - x_2) + \cdots + a_n(x - x_1)(x - x_2)\cdots(x - x_{n-1})$$

Két pontra felírva az együtthatók:

$$a_1 = y_1; a_2 = \frac{y_2 - y_1}{x_2 - x_1}$$

Definiáljuk az elsőrendű osztott differenciákat a következőképpen:

$$f[x_{i+1}, x_i] = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

Ez tulajdonképpen az egyenes meredeksége, és két pontra megegyezik a_2 együtthatóval.

Három pontra felírható a másodrendű osztott differencia $f[x_3, x_2, x_1]$, ami két elsőrendű osztott differencia különbsége osztva $(x_3 - x_1)$ -gyel, ez lesz az a_3 együttható (az első kettő együttható megegyezik a korábbiakkal).

[12] Otthoni átnézésre

$$a_3 = f[x_3, x_2, x_1] = \frac{f[x_3, x_2] - f[x_2, x_1]}{x_3 - x_1} = \frac{\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1}}{x_3 - x_1}$$

Hasonlóképp négy pontra felírható a harmadrendű osztott differencia két másodrendű osztott differencia különbségét elosztva $(x_4 - x_1)$-gyel, és így tovább.

Általánosan a k-adrendű osztott differencia:

$$f[x_{i+k}, \cdots, x_i] = \frac{f[x_{i+k}, \cdots, x_{i+1}] - f[x_{i+k-1}, \cdots, x_i]}{x_{i+k} - x_i}, (k = 1,2, \cdots, n) \text{és } (i = 0, \cdots, n-k).$$

- Látszik, hogy ebben az esetben is a pontok koordinátáit használva, előzetes számítások nélkül is felírható az interpolációs polinom.
- Ezzel már nem olyan nehézkes dolgozni, ha egyszer már meghatároztuk $a_1 \cdots a_n$ együtthatókat, akkor ezeket felhasználhatjuk bármelyik pont interpolációjánál.
- Ha a ponthalmazunk új ponttal bővül, akkor nem kell mindent újraszámolni, csak az új együtthatót. Ezáltal könnyen bővíthető új ponttal a halmaz, és a pontoknak nem kell sorrendben lennie.

TÁROZÓ JELLEGÖRBE INTERPOLÁCIÓJA

Nézzünk interpolációra egy vízépítő mérnöki feladatot. A tározóméretezések egyik elengedhetetlen alapadata a tározó morfológiai jelleggörbéje, ami megadja, hogy egy adott vízálláshoz mekkora víztérfogat és felület tartozik. Adottak a következő adatok:

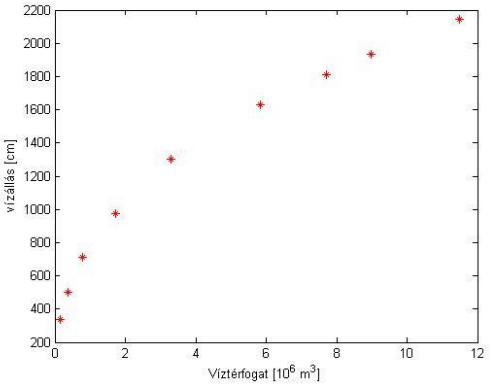
Vízállás H [cm]	Térfogat V [10^6 m^3]	Felület F [km^2]
336	0.16	0.05
504	0.36	0.09
714	0.79	0.19
976	1.73	0.37
1302	3.31	0.62
1628	5.83	0.90
1812	7.72	1.05
1932	8.98	1.16
2142	11.50	1.27

Ábrázoljuk a térfogatot leíró jelleggörbét, szokás szerint, úgy, hogy a vízszintes tengelyen a térfogat, a függőleges tengelyen pedig a vízállás legyen. Határozzuk meg ezek alapján interpolációval, hogy mekkora víztérfogat tartozik 15 m-es vízszinthez, illetve mekkora vízszint tartozik 12 millió m^3 térfogathoz!

Töltsük be a jelleggörbékhez tartozó adatokat a tarozo.txt állományból!

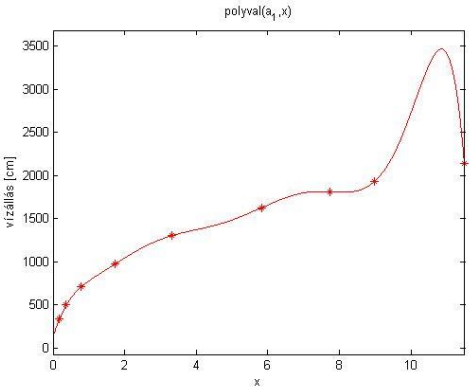
```
> clc; clear all; close all;
> adat = load('tarozas.txt')
```

```
> H = adat(:,1); % cm
> V = adat(:,2); % millió m^3
> F = adat(:,3); % km^2
>
> figure(1)
> plot(V,H,'*'); hold on;
> ylabel('vízállás [cm]')
> xlabel('Víztérfogat [10^6 m^3]')
```



Interpoláció esetén *n* pontra, *(n-1)*-ed fokú polinomot illeszthetünk. Nézzük meg, ez mit jelent a mostani esetben. Most 9 adatunk van, erre 8-ad fokú polinomot illeszthetünk!

```
> n = length(V) % 9
> a1 = polyfit(V,H,n-1)
> p1 = @(x) polyval(a1,x)
> fplot(p1,[0,max(V)])
```



Mi történt? Jó lehet a fenti görbe interpolációra? Nyilvánvaló hogy nem. Ez a túlzott illeszkedés esete, amikor a magas fokszámú polinom tökéletesen visszaadja a ismert pontok értékeit, de köztük (különösen a széleken) oszcillálni kezd, jelentősen eltér az adatok trendjétől, ezért nem alkalmazható megbízhatóan interpolációra, extrapolációra. Ezt a hullámzást, oszcillációs jelenséget **Runge jelenség**nek hívják. Kevés pont estén, amikor a polinom fokszáma alacsony, többnyire jól használhatóak az interpolációs polinomok, sok pont, magas fokszámú polinom esetén azonban más megoldást kell keresni!

A polyfit parancs futása után a Matlab egy figyelmeztető üzenetet is kiír:

```
Warning: Polynomial is badly conditioned. Add points with distinct X values,
reduce the degree of the polynomial, or try centering and scaling as described
in HELP POLYFIT.
```

Ha lekérdezzük a Vandermonde mátrix kondíció számát, akkor egy nagyon nagy számot kapunk (10^{10} nagyságrend), ami rosszul kondicionált mátrixot jelent.

```
> A = vander(V);
> cond(A) % 2.7260e+10
```

SPLINE INTERPOLÁCIÓ

Ha sok pont között szeretnénk interpolációt végezni, akkor jobb eredményt lehet elérni több, alacsony fokszámú polinom alkalmazásával, mint egy magas fokúval. Minden egyes alacsony fokszámú polinom csak egy adott szakaszra, intervallumra érvényes, két vagy több pont között. Általában minden polinomnak megegyezik a fokszáma, csak az együtthatókban térnek el egymástól. Amikor elsőfokú polinomokat használunk, akkor a pontokat egyenes vonalak kötik össze, másodfokú (négyzetes) vagy harmadfokú (köbös) esetben a pontok görbékkel vannak összekötve. Ez a szakaszonként eltérő paraméterű polinomiális interpoláció, amit **spline** interpolációnak neveznek. Ez tulajdonképpen egy fajta lokális interpoláció, mivel egy-egy szakaszra csak a környező pontokat vesszük figyelembe.

Az n-edfokú spline-ban legfeljebb n-edfokú polinom szakaszok csatlakoznak egymáshoz. A legegyszerűbb eset, ha az adott pontok közé lineáris függvényeket (elsőfokú polinomokat) illesztünk. Ez az Euler-féle töröttvonalak módszere. Gyakran szükséges, hogy a csatlakozási pontokban ne csak folytonos, hanem differenciálható is legyen a függvény. Ilyen lehet például egy másodfokú spline, ahol a csatlakozási pontokban a függvénynek jobbról és balról megegyeznek a deriváltjai is. Ezt négyzetes (kvadratikus) spline-nak is nevezik.

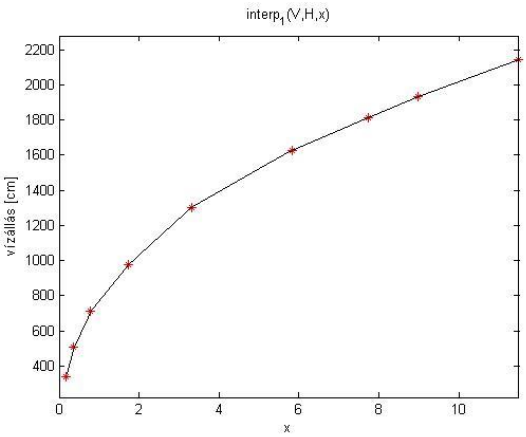
Általánosan k-ad fokú és m-ed rendű spline az, ahol szakaszonként k-ad fokú polinomokat illesztünk, és a közbülső pontokban m-ed rendig megegyeznek a deriváltak. Az egyik leggyakrabban használt spline a köbös, másodrendű spline interpoláció, amikor is az adott pontok közé harmadfokú polinomokat illesztünk úgy, hogy a csatlakozási pontokban a függvénynek megegyeznek a deriváltjai és második deriváltjai is. Létezik köbös elsőrendű spline interpoláció is, amikor szintén harmadfokú polinomokat illesztünk a pontokra, de csak az első deriváltak egyezését írjuk elő (pl. Hermite interpolációs polinomok).

LINEÁRIS SPLINE INTERPOLÁCIÓ

Illesszünk lineáris függvényeket a pontok közé! Adott *n* pont esetén *(n-1)* szakasz van, amire *(n-1)* egyenest kell meghatároznunk. Ezt a legegyszerűbben a két pontra felírt Lagrange-polinom segítségével tehetjük meg:

$$f_i(x) = \frac{(x - x_{i+1})}{(x_i - x_{i+1})} y_1 + \frac{(x - x_i)}{(x_{i+1} - x_i)} y_2$$

Matlab-ban az **interp1** függvényt használhatjuk általánosan spline interpolációra. Ennél egy paraméterben megadhatjuk, hogy milyen spline interpolációt szeretnénk használni, ha nem adunk meg semmit, akkor az alapértelmezett a lineáris interpoláció. Lehetséges interpolációs módszerek az **interp1**-nél: 'linear' - alapértelmezett, 'nearest' - legközelebbi szomszéd, 'pchip' - köbös elsőrendű spline, 'spline' - köbös másodrendű spline.




```
> figure(2);
> plot(V,H,'r*');hold on;
> ylabel('vízállás [cm]');
> xlabel('Víztérfogat [10^6 m^3]');
> sp = @(x) interp1(V,H,x)
> fplot(sp,[0,max(V)])
```

Lineáris függvényillesztésnél folytonos lesz ugyan a függvényünk, de a meredekségekben törések lesznek. Ha simább függvényt szeretnénk kapni, akkor magasabb fokú spline-t kell használni.

NÉGYZETES SPLINE INTERPOLÁCIÓ[13]

Négyzetes (kvadratikus, másodfokú) spline esetében az adott pontok közé másodfokú polinomokat ($y = a \cdot x^2 + b \cdot x + c$) illesztünk úgy, hogy a csatlakozási pontokban a függvénynek jobbról és balról megegyeznek az első deriváltjai is.

Egy másodfokú polinomnak 3 ismeretlen együtthatója van, és *n* pont esetén *(n-1)* szakaszra kell ezeket meghatározzuk, vagyis *3*(n-1)=3n-3* ismeretlenünk van, ezekre kell egyenleteket felírnunk, feltételeket megadnunk.

- Minden polinomnak át kell mennie a szakasz végpontjain, ebből szakaszonként két egyenlet írható fel a következő alakban: $f_i(x) = a_i \cdot x^2 + b_i \cdot x + c_i$, vagyis összesen *2*(n-1)=2n-2* egyenlet.
- A közbülső *(n-2)* pontban a deriváltak ($f'(x) = 2a_i x + b_i$) is megegyeznek, erre *(n-2)* egyenletet lehet felírni a következő alakban: $2\ a_{i-1}x_i + b_{i-1} = 2\ a_i x_i + b_i$
- Mivel a fenti két feltétel még csak *(3n-4)* egyenletet jelent (*3n-3)* ismeretlen meghatározására, még egy feltételt meg kell adni. Itt több lehetőség is van, lehet ez például az, hogy a második derivált értéke legyen 0 az első (vagy az utolsó) pontnál: $f_1''(x) = 2a_1$, vagyis $a_1 = 0$. Ez tulajdonképpen azt jelenti, hogy az első két pontot (vagy az utolsó két pontot) egyenessel kötjük össze.

A négyzetes spline-k illesztéséhez *(3n-4)* egyenletet kell felírjuk. A köbös, másodrendű spline-k használata elterjedtebb, részben mivel a második deriváltak (görbületek) is megegyeznek és emiatt simább a függvény, részben pedig amiatt, hogy levezethető egy olyan alakja is, ahol csak *(n-2)* egyenletet kell megoldani. A Matlab beépített függvényei között is csak köbös spline szerepel, négyzetes nem.

KÖBÖS MÁSODRENDŰ SPLINE INTERPOLÁCIÓ

Nézzük most az egyik leggyakrabban használt spline a köbös, másodrendű spline interpoláció levezetését az általános harmadfokú polinom képlete alapján!

Ebben az esetben az adott pontok közé úgy illesztünk harmadfokú polinomokat ($y = a \cdot x^3 + b \cdot x^2 + c \cdot x + d$) , hogy a csatlakozási pontokban a függvénynek megegyeznek az első és a második deriváltjai is. Adott *n* pont esetén *(n-1)* szakaszunk van, ezekre kell meghatároznunk harmadfokú polinomokat. Minden egyes harmadfokú polinomnak 4 ismeretlen együtthatója van, vagyis *4*(n-1)=4n-4* ismeretlenünk van, ezekre kell egyenleteket felírnunk, feltételeket megadnunk.

- Minden polinomnak át kell mennie a szakasz végpontjain, ebből szakaszonként két egyenlet írható fel a következő alakban: $y_i = a_i \cdot x^3 + b_i \cdot x^2 + c_i \cdot x + d_i$, vagyis összesen *2*(n-1)=2n-2* egyenlet.
- A közbülső *(n-2)* pontban megegyeznek a deriváltak ($f'(x) = 3ax^2 + 2bx + c$) is, erre *(n-2)* egyenletet lehet felírni a következő alakban: $3\ a_{i-1}{x_i}^2 + 2b_{i-1}x_i + c_{i-1} = 3\ a_i{x_i}^2 + 2b_i x_i + c_i$
- A közbülső *(n-2)* pontban a második deriváltak ($f''(x) = 6ax + 2b$) is megegyeznek, erre is *(n-2)* egyenletet lehet felírni a következő alakban: $6\ a_{i-1}x_i + 2b_{i-1} = 6\ a_i x_i + 2b_i$

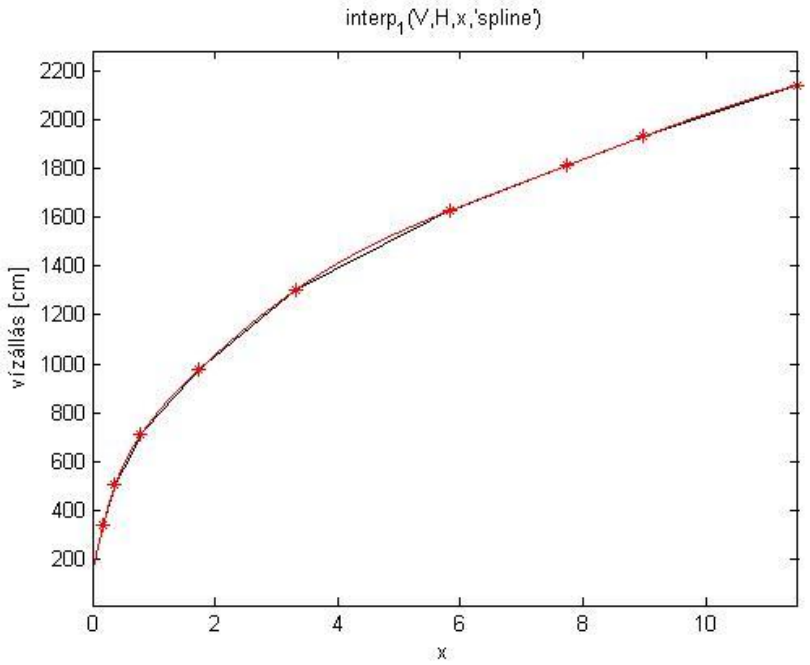
[13] Otthoni átnézésre

- Összesen eddig *(4n-6)* egyenletet írtunk fel *(4n-4)* ismeretlenre, tehát még két feltételt meg kell adni, amit többféleképpen lehet. Az egyik lehetőség, hogy a végpontokban a második deriváltak legyenek 0-k, ezt a fajta spline-t hívják természetes köbös spline-nak. Egy másik módszer, amit a Matlab beépített függvénye is használ, a „not-a-knot” feltétel. Ezt azt jelenti, hogy a második és az utolsó előtti pontban a harmadik deriváltak is megegyeznek.

Megjegyzés: levezethető egy másik alakja is ennek a köbös spline-nak, ahol nem *(4n-4)* egyenletet kell megoldani, hanem elég egy (n-2) lineáris egyenletből álló rendszert megoldani. Ez a levezetés a második deriváltakból indul ki és a Lagrange alakot használja. Lásd pl.: Amos Gilat, Vish Subramaniam (2011): Numerical Methods, An Introduction with Applications Using MATLAB, John Wiley & Sons

Illesszünk harmadfokú köbös spline-t Matlab-ban a tározó jelleggörbéjére! Ehhez használjuk ismét az **interp1** parancsot, csak most adjunk meg neki egy módszert is, az alapértelmezett 'linear' helyett legyen 'spline'!

```
> sp2 = @(x) interp1(V,H,x,'spline')
> fplot(sp2,[0,max(V)])
```



Látjuk, hogy ez egy simább lefutású függvényt eredményezett. Mint korábban szó volt róla, ez nem a természetes spline-t használja, hanem a not-a-knot”, vagyis 'nem csomópont' feltételt. Ezt azt jelenti, hogy a második és az utolsó előtti pontban a harmadik deriváltak is megegyeznek. Mivel itt harmadfokú polinomokról van szó, ezért az első kettő és az utolsó kettő szakaszon is teljesen megegyeznek a paraméterek, tehát tulajdonképpen az első 3 és az utolsó 3 pontra egy-egy polinomot illesztünk. Innen származik a neve, hogy a második és az utolsó előtti pont nem igazi csomópont.

Mivel a leggyakrabban ezt a köbös, másodrendű spline interpolációt alkalmazzák, ezért erre van egy külön parancs is, **spline** parancsként, aminek a működése egyenértékű az **interp1** 'spline' módszerével:

```
> sp2 = @(x) spline(V,H,x)
```

Térjünk vissza az eredeti kérdésre és az illesztett spline alapján határozzuk, hogy mekkora vízszint tartozik 12 millió m^3 térfogathoz, illetve mekkora víztérfogat tartozik 15 m-es (1500 cm-es) vízszinthez!

```
> % Mekkora vízállás tartozik 12 millió m^3 vízhez?
> H15 = sp2(12) % 2174 cm
> % 1500 cm-es vízállás mekkora víztérfogatot jelent?
> f = @(x) sp2(x)-1500
> V1500 = fzero(f,5) % 4.6699 millió m^3
```

KÖBÖS ELSŐRENDŰ SPLINE INTERPOLÁCIÓ

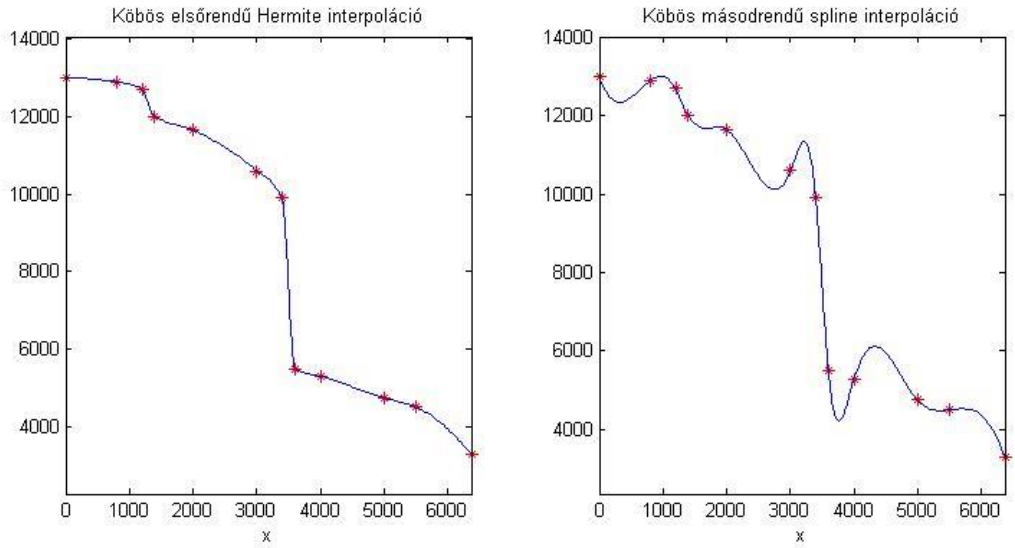
Matlab-ban van egy másik köbös spline interpoláció is. Az **interp1** parancsot meghívhatjuk 'pchip' módszerrel is (piecewise hermite interpolation). Ez egy köbös elsőrendű interpoláció, ahol ugyan harmadfokú polinomokat illesztünk az egyes szakaszokra, de csak az első deriváltak folytonosságát kötjük ki. Ez egy köbös Hermite interpoláció, ahol a deriváltakat is meghatározza a Matlab numerikus differenciálásból 3 pontra a közbülső pontok esetében és 2 pontból a függvény elején/végén. Nézzük meg mikor lehet hasznos ez a módszer!

Vegyünk most egy felsőgeodéziához köthető példát. A Föld gravitációs erőterének számításaihoz szükséges ismerni a Föld sűrűségének változásait. A Föld sűrűsége (ρ) a sugarával (R) együtt változik, megközelítően az alábbiak szerint:

Sugár [km]	0	800	1200	1400	2000	3000	3400	3600	4000	5000	5500	6370
Sűrűség [kg/m^3]	13000	12900	12700	12000	11650	10600	9900	5500	5300	4750	4500	3300

Illesszünk spline görbét a sugár-sűrűség értékekre, majd az interpolációs görbe alapján számítsuk ki mekkora lesz a Föld sűrűsége 3200 km-es sugárnál, illetve mekkora sugárnál lesz a sűrűség értéke pont 4000 kg/m^3? Ehhez töltsük be a fold_surusege.txt fájlt, majd illesszünk a pontokra köbös másodrendű és köbös elsőrendű spline-t is! Nézzük meg melyik illeszkedik jobban!

```
> data = load('fold_surusege.txt')
> r = data(:,1) % kilométerben a sugár
> ro = data(:,2) % sűrűség kg/m^3
>
> % köbös elsőrendű Hermite interpoláció
> fsp1 = @(x) interp1(r,ro,x,'pchip') % vagy 'pchip'
> fsp1(3200) % 10361
> figure(1);subplot(1,2,1);
> plot(r,ro,'r*'); hold on;
> fplot(fsp1,[0,max(r)]);
> title('Köbös elsőrendű Hermite interpoláció')
>
> % köbös másodrendű spline interpoláció
> fsp2 = @(x) spline(r,ro,x)
> fsp2(3200) %  11350
> subplot(1,2,2)
> plot(r,ro,'r*'); hold on;
> fplot(fsp2,[0,max(r)]);
> title('Köbös másodrendű spline interpoláció')
```



A két módszer elég nagy eltéréseket adott a 3200 km-nél lévő sűrűség értékekre, az első esetben 10361 kg/m^3 lett az eredmény, a másodikban pedig 11350 kg/m^3.

Az ábra alapján egyértelmű, hogy most jobb illeszkedést kaptunk a harmadfokú elsőrendű spline-ra ('pchip'), mint a harmadfokú másodrendű esetben ('spline'). Azért jobb most az előbbi módszer, mivel az adatokban erős törések, ugrások vannak. Sima függvények esetén a másodrendű 'spline' ad jobb közelítést.

A feladat második felére, hogy mekkora sugárnál lesz a sűrűség értéke pont 4000 kg/m^3 éppen ezért használjuk az elsőre meghatározott spline-t!

```
> % Mekkora sugárhoz tartozik 4000 kg/m^3 sűrűség?
> f = @(x) fsp1(x)-4000
> R4000 = fzero(f,5500) % 5958.9 km
```

Tehát kb. 6000 km-es sugárnál lesz a sűrűség értéke 4000 kg/m^3.

A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

vander	-	Vandermonde mátrix
interp1 (method: linear, nearest, spline, pchip)	-	Egydimenziós interpoláció (módszer: lineáris, legközelebbi szomszéd, köbös másodrendű, köbös elsőrendű Hermite interpoláció)
spline	-	Egydimenziós, köbös másodrendű spline interpoláció

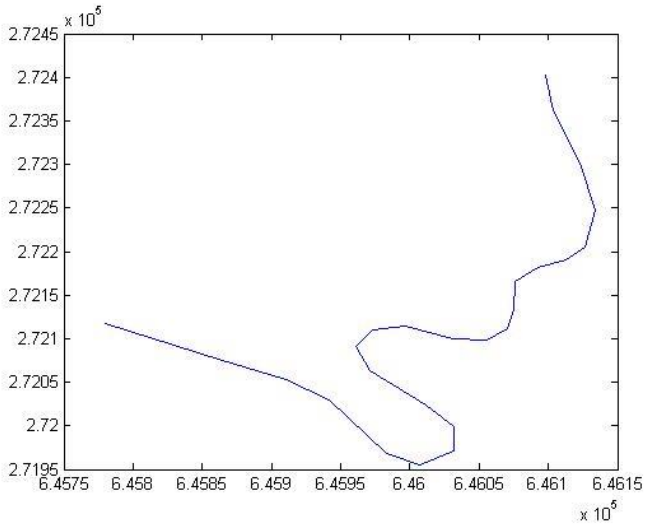
11. KÉTVÁLTOZÓS INTERPOLÁCIÓ, REGRESSZIÓ

TÖBBÉRTÉKŰ GÖRBÉK INTERPOLÁCIÓJA

A spline interpoláció jól alkalmazható olyan görbék közelítésére is, amelyeknél a függvény egy adott x pontban több y értéket is felvehet (pl. körvonal esetén). Ilyen esetben a görbe paraméteres alakját használjuk. Paraméter lehet például a pont sorszáma, vagy a görbe ívhossza is. Nézzünk most ez utóbbira egy példát!

Felmérték a Visegrádi várba vezető szerpentinút közel egy kilométeres szakaszán a tengelyvonalat. Ábrázoljuk a felmérés eredményeit, majd illesszünk spline görbét a pontokra és határozzuk meg a kezdőponttól számított 500 m-es útszelvény EOV koordinátáit! Az interpolációhoz válasszuk paraméternek a görbe pontjainak a távolságát (ívhosszát). Az adatokat töltsük be a ***visegrad.txt*** fájlból. Ebben EOV vetületben vannak a felmért pontok Y,X koordinátái.

```
> % Többértékű görbék interpolációja
> clc; close all; clear all;
> adat = load('visegrad.txt')
> x = adat(:,1); y = adat(:,2);
> figure(1)
> plot(x,y)
```



Az ábrán láthatjuk, hogy vannak olyan helyek, ahol egy adott x koordinátához több y érték is tartozik, hagyományos függvény interpolációval nem tudnánk rá görbét illeszteni. Paraméteres alakban viszont megoldható a feladat. Olyan paramétert kell választanunk, ami minden pont esetében egyértelmű megoldást ad. Legyen most ez a paraméter a görbe ívhossza, amit a felmért pontok közötti távolsággal fogunk közelíteni. Ezután két függvényillesztést hajtunk végre, egyet az x koordinátára az ívhossz függvényében, egyet pedig hasonlóképp az y koordinátára.

Számítsuk ki először a pontok x,y irányú koordináta különbségeit ($\Delta x, \Delta y$)! Ezt egyszerűen megtehetjük a numerikus **diff** parancsot használva. Ez egy vektor esetében a szomszédos elemek különbségeit adja vissza (értelemszerűen *n* elem esetén *n-1* értéket). Ebből már számíthatunk távolságokat Pitagorasz-tétellel (a

kezdőpontnak saját magától mért távolsága 0), majd ezeket folyamatosan összegezve a **cumsum** paranccsal megkapjuk a kezdőponttól mért távolságokat.

```
>  %  Paraméter: a görbe ívhossza, távolságokkal közelítve
>  % x,y irányú koordináta különbségek a pontok között:
>  dx = diff(x)
>  dy = diff(y)
>  % távolságok:
>  t0 = [0; sqrt(dx.^2+dy.^2)]
>  % folyamatosan összegezve:
>  t = cumsum(t0)
```

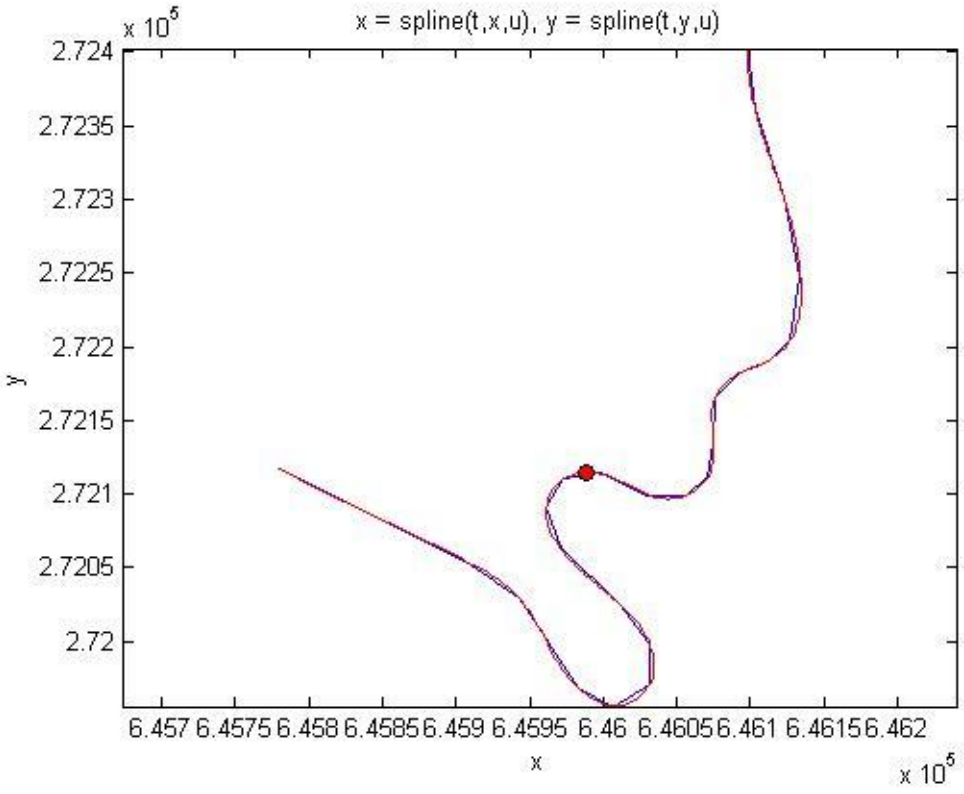
Illesszünk spline görbét az x és y értékekre az ívhossz függvényében!

```
>  %  Az interpolációs függvények x-re és y-ra
>  xt=@(u) spline(t,x,u);
>  yt=@(u) spline(t,y,u);
>  %  Ábrázoljuk az eredményt
>  hold on;
>  fplot(xt,yt,[0,t(end)],'r');
```

Mi lesz az EOV koordinátája az 500-as szelvény tengelypontjának?

```
>  % Milyen koordináta tartozik az 500 m-es szelvényhez?
>  format long
>  x500 = xt(500) % 6.459890032622117e+05
>  y500 = yt(500) % 2.721148370593938e+05
>  format short
>  plot(x500,y500,'ko','MarkerFaceColor','r')
```

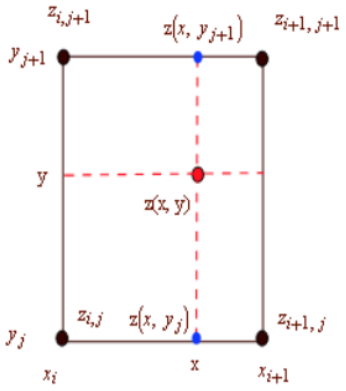
Tehát a felmérés kezdőpontjától az 500-as szelvény EOV Y koordinátája 645 989.00 m, az EOV X koordinátája pedig 272 114.84 m.



KÉTVÁLTOZÓS INTERPOLÁCIÓ SZABÁLYOS RÁCSON ADOTT PONTOK ESETÉN[14]

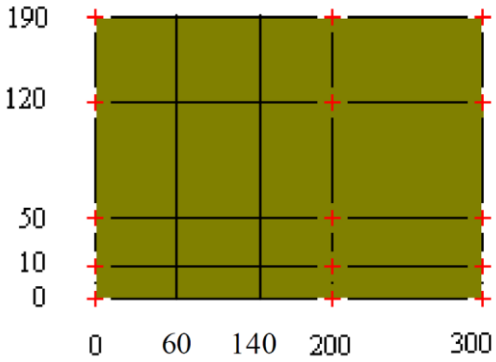
Kétváltozós esetben a legegyszerűbb interpolációs módszer a szabályos rácshálóban adott pontoknál alkalmazható bilineáris interpoláció. Az interpoláció három egyváltozós interpolációs lépésből állhat:

1) Interpoláció az $y = y_j$mentén vízszintesen: $z(x, y_j)$,
2) interpoláció az $y = y_{j+1}$ mentén vízszintesen: $z(x, y_{j+1})$,
3) végül a harmadik interpoláció az $x = állandó$mentén függőlegesen, felhasználva az előbbi két interpoláció eredményét: *z*(*x*, *y*).



Megjegyzés**:** A fentiek csak a bilineáris interpoláció lépései, azonban a *j-1*, *j+2* stb. pontok bevonásával magasabb fokú interpoláció is lehetséges.

Nézzünk egy tereprendezési példát a fentiekre! A terep magasságait egy szabályos rács pontjaiban mérésekkel határozták meg. A mérési pontok koordinátái és a mért magasságok az alábbiak (a magasságok a ***terep.txt*** fájlból beolvashatók):



$$Z = \begin{bmatrix} 135.0 & 131.7 & 136.4 & 139.8 & 119.8 \\ 134.8 & 133.0 & 139.7 & 135.5 & 120.0 \\ 124.4 & 130.0 & 140.0 & 139.2 & 122.3 \\ 131.1 & 133.8 & 138.1 & 137.7 & 121.1 \\ 133.2 & 137.1 & 143.0 & 135.0 & 123.3 \end{bmatrix}$$

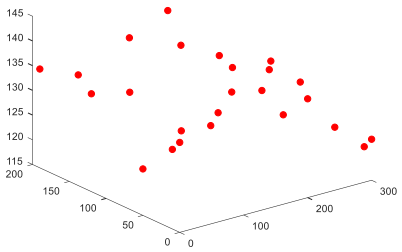
Mekkora a terep magassága a (100,100) pontban? Ábrázoljuk ebben a pontban a Ny-K irányú metszetet! Mennyi földet kell hozni vagy elvinni, ha a fenti mérési eredményekkel rendelkező domborzatot 135 m magasságú sík tereppé szeretnénk átalakítani?

A megoldáshoz elő kell állítanunk a rácspontok x,y koordinátáit is és beolvasni a hozzájuk tartozó z értékeket. Figyeljünk arra, hogy a matematikai koordináta rendszer kezdőpontja a bal alsó sarokba esik, a mátrixok kezdőpontja (első sor, első oszlop) pedig a bal felső sarokba! A koordináta rendszerek eltérése miatt itt az y koordinátákat fentről lefelé kell megadjuk!

```
> % Tereprendezes
> clc; clear all; close all;
> % A magasság értékek beolvasása
> Z = load('terep.txt')
> %  A rácspontok koordinátái (figyeljünk az adatok sorrendjére!)
```

[14] Felhasználva: Paláncz Béla (2012): Numerikus módszerek példatár + előadás fóliák

```
> x=[0 60 140 200 300];
> y=[190 120 50 10 0];
> %  Rácsháló előállítása
> [X,Y]=meshgrid(x,y)
```

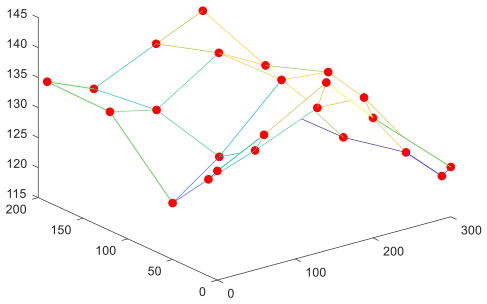


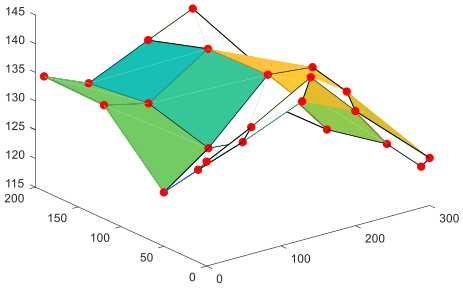
Ezzel előállt egy rácsháló x,y,z koordinátákkal. Jelenítsük meg a mért pontokat 3 dimenzióban (**plot3**)!

```
> figure(1)
> plot3(X,Y,Z,'ro','MarkerFaceColor','r')
```

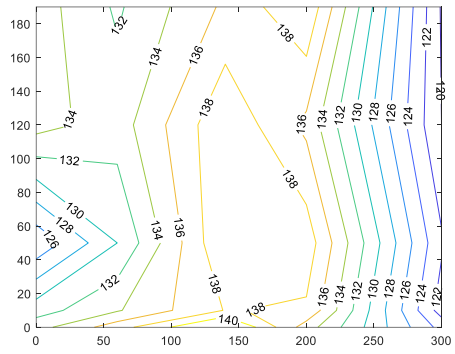
Szemléletesebben is meg lehet jeleníteni a domborzatot rácshálóként (**mesh**), vagy felületként (**surf**). Nézzük meg ezeket is!

```
> hold on; mesh(X,Y,Z)
> surf(X,Y,Z)
```





Másik gyakran használt lehetőség a szintvonalas ábrán történő megjelenítés (**contour**), ahol opcióként megadhatjuk azt is, hogy melyik szintvonalak jelenjenek meg (pl. most állítsuk be, hogy 120 és 140 méter között 2 méterenként legyenek szintvonalak), illetve bekapcsolhatjuk a szintvonalak feliratozását is a **set** paranccsal, ha két kimenettel hívjuk meg a **contour**-t, és a második kimenet tulajdonságait módosítjuk.



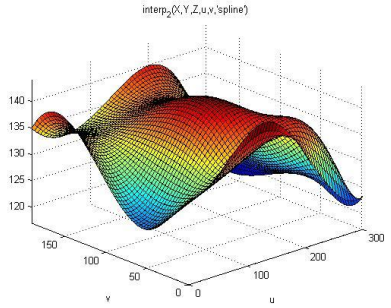
```
> figure(2)
> [C,h] = contour(X,Y,Z,120:2:140)
> set(h,'ShowText','on')
```

A fenti parancsok csak megjelenítésre szolgálnak, ebből még nem tudjuk meghatározni egy tetszőleges pontban a terep magasságát, ahhoz interpolációra (vagy regresszióra) lesz szükség.

Illesszünk a pontokra interpolációs felületet! Az interpolációhoz egy független változó esetén eddig az **interp1** parancsot használtuk, most 2 független változó esetén az **interp2** parancs használható. Az **interp2** csak rácshálóban megadott pontok/adatok esetén alkalmazható (a rácspontok egyenlő távolsága nem követelmény, csak a 'cubic' módszer alkalmazásakor). Több különböző módszerrel is meghívható: 'nearest' - legközelebbi szomszéd, 'linear' - bilineáris interpoláció, 'spline' - spline interpoláció, 'cubic' - 2D köbös spline interpoláció (bicubic), csak egyenletes rácstávolság esetén alkalmazható, ellenkező esetben spline interpoláció kerül helyette alkalmazásra.

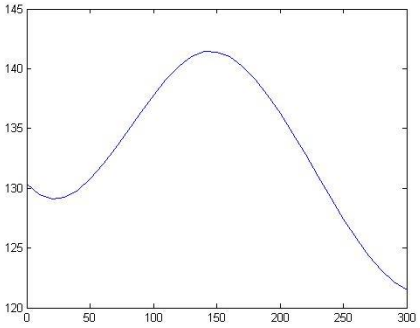
Rajzoljuk fel az interpolációs felületet az illesztett függvényt felhasználva (**fsurf** vagy **ezsurf**)!

```
>  %  spline interpoláció
>  F=@(u,v) interp2(X,Y,Z,u,v,'spline');
>  %  A terep megjelenítése
>  figure(3);
>  fsurf(F,[0,300,0,190])
>  %  Ellenőrzés
>  F(0,0) % 133.2000
>  F(300,190) % 119.8000
>  % Terep magassága a 100,100 pontban?
>  F(100,100) % 137.8079
```



A (0,0) pontban a magasság 133.2 volt, a (300,190)-ben pedig 119.8, az interpoláció eredményeképpen visszakaptuk ezeket az értékeket. A (100,100) pont magassága spline interpolációval 137.8 m-re adódott. Próbaképp cseréljük le a 'spline' módszert a 'nearest'-re, 'linear'-ra, hogy lássuk, mi történik ezekben az esetekben!

Rajzoljuk fel a Ny-K irányú metszetet ebben a pontban. Ehhez vegyünk fel 50 pontot az y=100 keresztmetszetben a $P_1(0,100)$ és a $P_2(300,100)$ pontok között! Használjuk az előbb meghatározott függvényt a magasságok kiszámítására!



```
>  % Ny-K irányú terepmetszet
>  x0 = linspace(0,300,50)
>  y0 = linspace(100,100,50)
>  % vagy: y0 = ones(1,50)*100
>  z0 = F(x0,y0)
>  figure(4); plot(x0,z0);
```

Oldjuk meg a feladat második részét is, hogy mennyi földet kell hozni vagy elvinni, ha a fenti mérési eredményekkel rendelkező domborzatot 135 m magasságú sík tereppé szeretnénk átalakítani? Ehhez számítsuk ki a terep és a tengerszint által határolt földtömeg mennyiségét és a 135 méteres vízszintes sík és a tengerszint által határolt földtömeg mennyiségét a megadott téglalapra. A kettő különbsége fogja megadni a tereprendezéshez szükséges föld mennyiségét. A terep által határolt földtömeg mennyiségét az alábbi kettős integrállal (**integral2**) számíthatjuk:

$$V_t = \int_0^{190} \int_0^{300} F(x,y)\ dx\ dy$$

A vízszintes z=áll. sík által határolt földtömeg mennyisége egy téglatest térfogata: $V_{sz} = z \cdot 300 \cdot 190$.

```
>  %  Terep alatti földtérfogat téglalap tartományon kettős integrállal
>  Vt=integral2(F,0,300,0,190) % 7.6140e+06
>  %  Vízszintes sík által határol földtérfogat
>  Vs=135*300*190 % 7695000
>  %  A tereprendezéshez szükséges földtérfogat
>  Vr=Vs-Vt % 8.0983e+04
```

Mivel a tervezett vízszintes sík által határol térfogat nagyobb volt, mint a terep alatti, ezért feltöltésre van szükség, méghozzá 80983 m^3 földre.

KÉTVÁLTOZÓS REGRESSZIÓ[15]

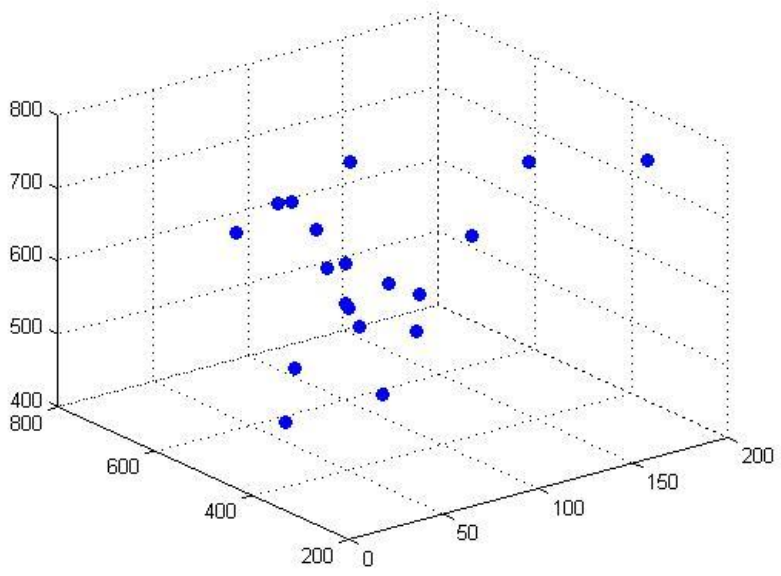
Szabálytalan elrendezésű pontok esetében jóval bonyolultabb interpolációt végezni. Ellenben a polinomiális regressziónak nem követelménye a pontok szabályos elhelyezkedése. Ez utóbbi esetben viszont ügyelni kell, hogy nagyon magas fokszámú polinomot nem szabad alkalmazni a túltanulás miatt. Az illesztő pontokban a hibaösszeg csökkenése ugyanis nem jelent feltétlenül megbízhatóbb modellt, sokszor a pontok között a felületben nagyfokú hullámzás, oszcilláció lép fel.

Nézzünk egy példát kétváltozós regresszióra, interpolációra! A magyar Duna vízgyűjtőjének területei Ausztriában és Bajorországban találhatók. Ha itt jelentős mennyiségű csapadék esik, akkor a Dunán árhullám vonul le. Feladatunk a budapesti tetőző vízállás előrejelzése a következő két adat alapján:

- x: az árhullámot kiváltó csapadék, amely 15 bajor illetve osztrák csapadékjelző állomás adatainak középértéke [mm];
- y: a Duna vízállása Budapestnél az árhullámot kiváltó esőzések kezdetekor [cm];
- z: a becsülendő érték Budapestnél, az árhullám tetőzéséhez tartozó vízszint [cm].

Az eddig mért adatokat az alábbi táblázat tartalmazza (az adatok az arhullam.txt file-ban elérhetőek), ebben egy sor egy adott időponthoz tartozó x, y, z értékeket jelenti:

58	405	590
52	450	660
133	350	780
179	285	770
98	330	710
72	400	640
72	550	670
43	480	520
62	450	660
67	610	690
64	380	500
33	460	460
57	425	610
62	560	710
54	420	620
48	620	660
68	390	620
74	350	590
95	570	740



2017. augusztus 3-án az osztrák és bajor vízgyűjtő területen átlagosan 100 mm csapadék esett. Ugyanekkor a Duna vízszintje Budapesten 400 cm-nél volt. Mekkorára várható az árhullám tetőzése Budapesten?

[15] Felhasználva: Paláncz Béla (2012): Numerikus módszerek példatár + előadás fóliák

Illesszünk egy másodfokú regressziós polinomot a pontokra, és ennek felhasználásával adjunk becslést a várható tetőzési magasságra! Illesszünk felületet a pontokra interpolációval is, lineáris és spline interpolációt alkalmazva és ezek alapján is becsüljük meg az árhullám várható tetőzését!

Először olvassuk be és ábrázoljuk az adatokat! 3D szórt pontokat a **plot3** vagy a **scatter3** paranccsal tudunk megjeleníteni.

```
> clc; clear all; close all;
> xyz=load('arhullam.txt')
> x=xyz(:,1); % árhullámot kiváltó csapadék [mm]
> y=xyz(:,2); % Duna vízállása Budapestnél az esőzés kezdetekor [cm]
> z=xyz(:,3); % a becsülendő érték, árhullám tetőzése Budapestnél [cm]
> % Ábrázolás
> figure(1);clf;
> scatter3(x,y,z,'filled')
```

Kétváltozós esetben regressziós síkot a következő módon írhatunk fel:

$$z = f(x,y) = p_1 + p_2 x + p_3 y$$

A másodfokú regressziós polinom általános felírása pedig a következő:

$$z = f(x,y) = p_1 + p_2 x + p_3 y + p_4 x^2 + p_5 xy + p_6 y^2$$

Harmadfokú regressziós polinom:

$$z = f(x,y) = p_1 + p_2 x + p_3 y + p_4 x^2 + p_5 xy + p_6 y^2 + p_7 x^3 + p_8 x^2 y + p_9 x y^2 + p_{10} y^3$$

Hasonlóképp negyed-, ötöd- stb. fokú polinomokat is felírhatnánk, de a túltanulás jelensége miatt 5. fokúnál magasabbat nem szoktak alkalmazni.

Most a feladatban másodfokú polinomiális felületet kell illesztenünk. n pont esetén n darab lineáris egyenletet írhatunk fel, és 6 meghatározandó paraméterünk van. Ez mátrixos alakban a következő lesz ($A \cdot p = b$):

$$A = \begin{pmatrix} 1 & x_1 & y_1 & {x_1}^2 & x_1 y_1 & {x_1}^2 \\ 1 & x_2 & y_2 & {x_2}^2 & x_2 y_2 & {x_2}^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & y_n & {x_n}^2 & x_n y_n & {x_n}^2 \end{pmatrix}; p = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_6 \end{pmatrix}; \; b = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{pmatrix}$$

Mivel a feladatban jóval több, mint 6 adatunk van, ezért egy túlhatározott lineáris egyenletrendszert kell megoldanunk, a legkisebb négyzetek módszerével minimalizálva a maradék ellentmondásokat. A feladat megoldása lényegében ugyanúgy megy, mint egyváltozós esetben.

```
> n = length(x) % 19
> A = [ones(n,1) x y x.^2 x.*y y.^2]
> p = A\z
```

A megoldást megkaphatjuk a matlab egy másik beépített függvényét, a **regress**-t használva is, ekkor az együtthatók mellett megkaphatjuk a 95 százalékos konfidencia intervallumukat és a maradék ellentmondásokat is

```
> [p int95 err]=regress(z,A)
```

Definiáljuk a regressziós felületet függvényként, és rajzoljuk be az illesztett felület az ábrába! Ábrázoljuk szintvonalakkal is a felületet! Becsüljük meg a várható árhullám tetőzését 100 mm csapadék és 400 cm-es vízszint esetén!

```
> f=@(x,y) p(1)+p(2)*x+p(3)*y+p(4)*x.^2+p(5)*x.*y+p(6)*y.^2
> hold on;
> fsurf(f,[min(x),max(x),min(y),max(y)])
```

Forgassuk el kicsit az ábrát a Rotate 3D paranccsal a grafikus ablakban, hogy jobban látszódjon az illeszkedés. Ábrázoljuk az illesztett *f* függvényt szintvonalas ábrán is! Függvény esetében a **contour** parancs nem használható, hanem vagy az **ezcontour** vagy az **fcontour**. A Matlab 2017-es verziójától az **ezcontour** használata nem javasolt, hanem helyette az **fcontour**, viszont ez utóbbi esetében egyelőre nem működik a szintvonalak feliratozása, úgyhogy egyelőre maradjunk az előbbinél. Ha a feliratokat be szeretnénk kapcsolni, akkor a szintvonalas ábrát el kell mentin egy változóba, és utána a **set** parancs segítségével állíthatjuk be a feliratozást ('Show','on'), illetve itt adhatjuk meg, hogy melyik szintvonalak jelenjenek meg ('LevelList',450:50:800). Figyeljünk, hogy a sima contour parancs esetében a ('ShowText','on') kapcsolót kellett beállítani, itt ez eltér ettől.

```
> figure(2); hold on;
> h = ezcontour(f, [min(x) max(x) min(y) max(y)])
> set(h,'Show','on','LevelList',450:50:800)
```

Az utolsó paranccsal beállítottuk, hogy a szintvonal feliratai is jelenjenek meg, és, hogy az alap 100 m helyett 50 m-ként rajzoljon szintvonalakat.

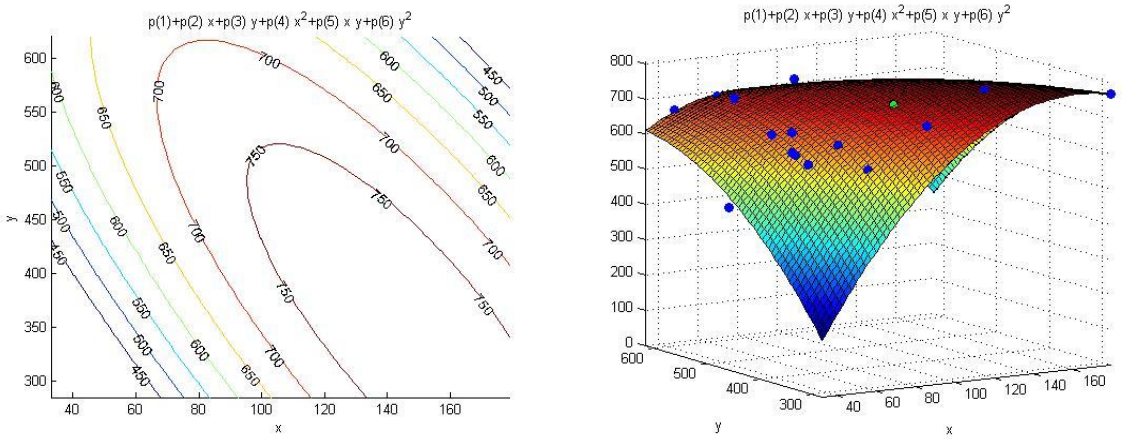
Megj. A legújabb Matlabnál javasolják az ezcontour helyett az fcontour használatát. Itt sajnos egyelőre nincs lehetőség feliratozni a szintvonalakat.

```
> fcontour(f, [min(x) max(x) min(y) max(y)],'LevelList',450:50:800)
```

Az árhullám várható tetőzése 100 mm csapadék és 400 cm-es vízszint esetén a regressziós polinommal végzett becslés alapján 738 cm lesz:

```
> % Előrejelzés
> zpol = f(100,400) % 737.8771
```

Az első ábránkba, ahol az adatokat megjelenítettük rajzoljuk be az illesztett felületet 3D-ben és az előre jelzett értéket is. Forgassuk el kicsit az ábrát a Rotate 3D paranccsal a grafikus ablakban, hogy jobban látszódjon az illeszkedés.

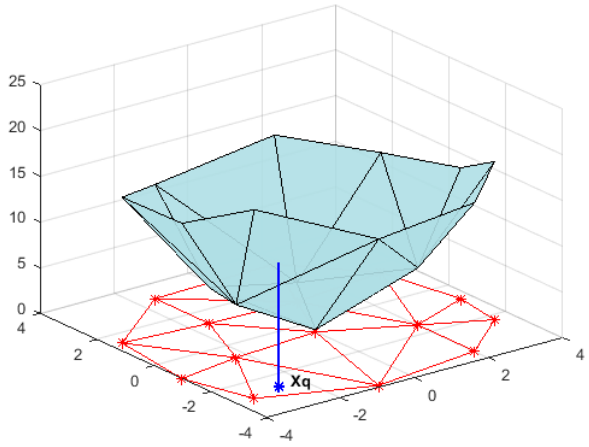


KÉTVÁLTOZÓS INTERPOLÁCIÓ SZABÁLYTALAN ELRENDEZÉSŰ PONTOK ESETÉN

Szabálytalan elrendezésű pontok esetében többféle interpolációs módszert szoktak alkalmazni. Lehet ez pl. a legközelebbi szomszéd módszere, amikor a pont a hozzá legközelebb eső pont értékét kapja, történhet az interpoláció krigeléssel, távolsággal fordított súlyozással, szabályos geometriai elemekre bontással vagy radiál–bázisfüggvény módszerét alkalmazva.

A Matlab beépített függvényei között a szabályos geometriai elemekre bontást találjuk meg. A **griddata** függvény Delaunay háromszög hálózatot vesz fel, és a háromszögek alapján végez interpolációt. Az interpoláció típusa lehet legközelebbi szomszéd interpoláció ('nearest'), háromszög alapú lineáris interpoláció, ekkor minden háromszögre egy síkot illesztünk ('linear'), vagy háromszög alapú köbös interpoláció ('cubic' - háromszögelésen alapuló 'bicubic spline' interpoláció). Illetve használható még a 'v4' módszer is, ami biharmonikus spline interpolációt végez. Ez utóbbi módszer nem háromszögelésen alapul.

A módszerek alapja a Delaunay háromszögelés, ahol úgy veszik fel a szórt pontok alapján a háromszögeket, hogy egy háromszög köré írt körbe ne kerüljön másik pont. Előnye ennek a háromszögelésnek, hogy mivel maximalizálja a háromszög legkisebb szögét, ezáltal kerüli a keskeny háromszögeket. Néhány alkalmazási területe: domborzatmodellezés (segítségével könnyen számíthatók az esés-, kitettség viszonyok, meghatározhatóak a szintvonalak), illetve a végeselem-módszerben is gyakran használják, mivel könnyen előállítható.



Határozzuk meg a háromszög alapú lineáris és köbös interpolációval is az árhullám várható tetőzését, ha a lehullott csapadék 100 mm és a vízállás Budapestnél 400 cm!

```
> zlin = griddata(x,y,z,100,400,'linear') % 724.7377
> zkob = griddata(x,y,z,100,400,'cubic') % 735.3939
```

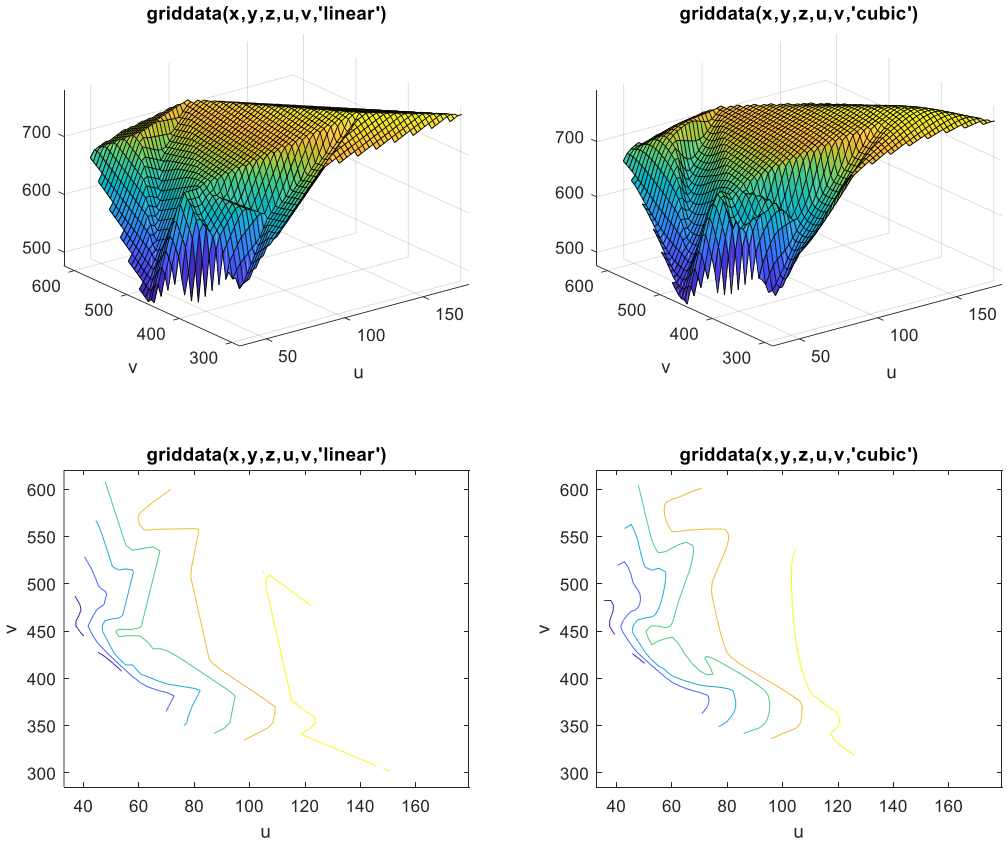
A polinom illesztéssel végzett előrejelzés alapján a tetőzés 738 cm-en várható, a lineáris interpoláció alapján 725 cm-en, a köbös alapján pedig 735 cm-en. Függvényként is definiálhatjuk a fentieket, és akkor tetszőleges pontra könnyen meghívhatjuk, pl. a köbös interpolációra:

```
> % Interpolációs függvény definiálása
> flin = @(u,v) griddata(x,y,z,u,v,'linear')
> fkob = @(u,v) griddata(x,y,z,u,v,'cubic')
> flin(100,400) % 724.7377
> fkob(100,400) % 735.3939
```

Ábrázoljuk a két függvényt térben, illetve szintvonalakkal, egy ábra 4 részterületén. Sajnos a griddata-ból előállított függvényt az **fplot** nem jeleníti meg, csak az **ezplot**.

```
> figure(3);
```

```
> subplot(1,2,1); ezsurf(flin,[min(x) max(x) min(y) max(y)])
> subplot(1,2,2); ezsurf(fkob,[min(x) max(x) min(y) max(y)])
> subplot(2,2,3); ezcontour(flin,[min(x) max(x) min(y) max(y)])
> subplot(2,2,4); ezcontour(fkob,[min(x) max(x) min(y) max(y)])
```



A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

diff	-	vektor elemeinek különbsége, közelítő numerikus derivált, szimbolikus derivált számítása
cumsum	-	vektor elemeinek folyamatos összegzése
meshgrid	-	2-3 dimenziós rács előállítása vektorban tárolt x,y(,z) koordinátákból
plot3	-	Pontok 3D megjelenítése
mesh	-	Rácshálóban adott 3D pontok megjelenítése térbeli rácsként
surf	-	Rácshálóban adott 3D pontok megjelenítése színezett felületként (kitöltött térbeli rács)
contour	-	Rácshálóban adott 3D pontok alapján szintvonalak rajzolása
set	-	grafikus objektum (pl. h) megadott tulajdonságainak beállítása, pl. szintvonalak feliratozása (set(h,'ShowText','on') contour parancs esetén, vagy set(h,'Show',on) ezcontour esetén)
interp2	-	2D interpoláció rácshálóban adott pontokból tetszőleges pontra (módszer: lineáris – 'linear', legközelebbi szomszéd - 'nearest', spline inetrpoláció – 'spline', 2D köbös spline (bicubic) – 'cubic')

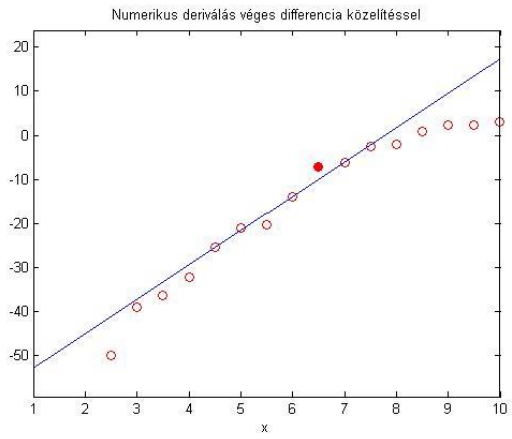
integral2	-	Kettős integrál számítása numerikusan, szabályos téglalap tartományon
scatter3	-	Szórt pontok 3D megjelenítése
regress	-	Többváltozós lineáris regresszió legkisebb négyzetek módszerével
fsurf, ezsurf	-	3D felületek kirajzolása megadott függvény alapján
fcontour, ezcontour	-	Szintvonalak kirajzolása függvény alapján
griddata	-	Interpoláció szórt pontok alapján tetszőleges pontra vagy rácsra (módszer: háromszög alapú lineáris interpoláció (TIN modell) – ’linear’, legközelebbi szomszéd - ’nearest’, háromszög alapú köbös interpoláció – ’cubic’, biharmonikus spline inetrpoláció – ’v4’)

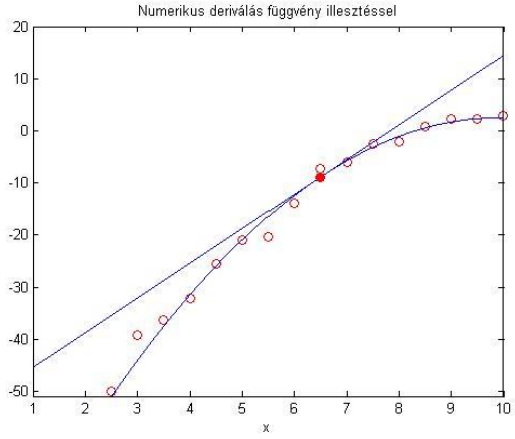
12. NUMERIKUS DERIVÁLÁS

A deriválttal (vagy differenciálhányadossal) egy mennyiség megváltozásának az ütemét tudjuk jellemezni. Nagyon sok helyen találkozhatunk vele a mérnöki gyakorlatban, tudományokban. Talán a legismertebb fizikai példa az út, sebesség gyorsulás összefüggése. Ha a megtett utat, pozíciót (x) idő függvényében (t) írjuk fel: $x = f(t)$, akkor a tárgy sebessége $v(t)$ az út idő szerinti első deriváltja lesz (az idő-út grafikon meredeksége): $v = \frac{df(t)}{dt}$, a gyorsulás pedig az út idő szerinti második deriváltja, vagy a sebesség első deriváltja: $a = \frac{dv(t)}{dt}$. Ugyancsak a deriváltat használhatjuk egy függvény minimumának vagy maximumának megkeresésére, mint azt láttuk korábban.

A deriválandó függvény lehet analitikus kifejezés vagy diszkrét pontokban megadott értékek. Egyszerű matematikai kifejezéssel megadott függvény deriváltja számítható analitikusan. Bonyolult matematikai kifejezések, vagy diszkrét pontok esetében azonban numerikus deriválást szoktak alkalmazni. Ugyancsak fontos szerepe van a numerikus deriválásnak a differenciál egyenletek megoldásakor.

A numerikus differenciálás egyik fontos módszere a derivált közelítése véges differencia hányadossal. Egy másik megközelítés, amikor a pontokat közelítjük valamilyen analitikusan felírható függvénnyel (pl. polinomiális regresszió, interpoláció) és az analitikus függvényt deriváljuk.





VÉGES DIFFERENCIA KÖZELÍTÉS

Tegyük fel, hogy csak diszkrét pontokban ismerjük a deriválandó függvény értékeit. A deriváltat közelíthetjük a szomszédos pontokat összekötő egyenes meredekségével. Erre több lehetőség is adódik. Az egyik a **jobb oldali vagy előremutató differencia**:

$$f'(x_i) = y_i' \approx \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

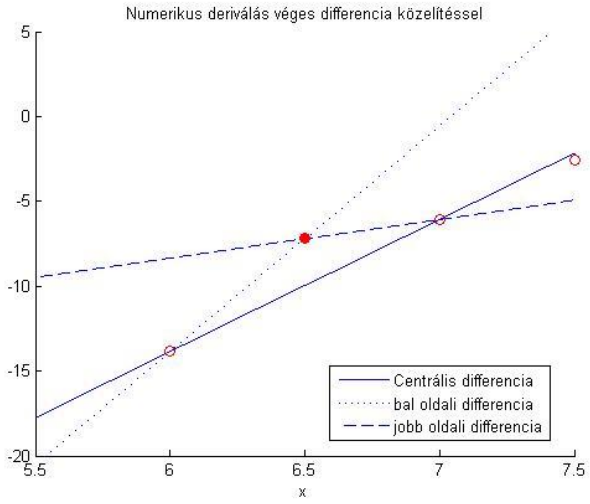
Egy másik lehetőség a **baloldali vagy hátramutató differencia**:

$$f'(x_i) = y_i' \approx \frac{y_i - y_{i-1}}{x_i - x_{i-1}}$$

Miután a jobb és baloldali differenciák hibáinak előjele gyakran ellentétes, jobb megoldást adhat a kettő átlaga. Amennyiben a pontok felosztása egyenletes ($x_{i+1} - x_i = x_i - x_{i-1} = h$) ez a **centrális differencia** formulához vezet:

$$f'(x_i) = y_i' \approx \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}} = \frac{y_{i+1} - y_{i-1}}{2h}$$

Általában a centrális differencia formula pontosabb közelítését adja a deriváltnak, mint a jobb vagy baloldali differencia. A pontok közötti távolság csökkenése is pontosabb eredményhez vezet.



A VÉGES DIFFERENCIA KÖZELÍTÉSEK HIBÁI

A Taylor sorokat használhatjuk a jobb, bal és centrális differenciák csonkítási hibabecsléséhez.

$$f(x+h) = f(x) + h\,f'(x) + \frac{h^2}{2} f"(c)$$

ahol *c* egy ismeretlen szám *x* és *x+h* között. Legyen $x = x_i,\ x + h = x_{i+1}$ és fejezzük ki f'-t az egyenletből!

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} - \frac{h}{2} f"(c)$$

A fenti formula a jobboldali (előremutató) differencia képlete, amelynek a hibája $-\frac{h}{2} f"(c)$, vagyis a csonkítási hiba nagyságrendje $O(h)$ (nagy ordó h). A fenti képletben h-t $(-h)$-ra cserélve megkapjuk a baloldali differencia képletét. A centrális differencia formula hiba becslését a harmadrendű Taylor sorból kaphatjuk meg:

$$f(x_{i+1}) = f(x_i + h) = f(x_i) + h\,f'(x_i) + \frac{h^2}{2} f''(x_i) + \frac{h^3}{3!} f'''(c_1)$$

$$f(x_{i-1}) = f(x_i - h) = f(x_i) - h\,f'(x_i) + \frac{h^2}{2} f''(x_i) - \frac{h^3}{3!} f'''(c_2)$$

ahol $x_i < c_1 < x_{i+1}$ és $x_{i-1} < c_2 < x_i$. Vonjuk ki egymásból a két egyenletet és fejezzük ki f'-t!

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_{i-1})}{2\,h} - \frac{h^2}{3!}\,\frac{f'''(c_1) + f'''(c_2)}{2}$$

A fentiek alapján a centrális differencia formula $O(h^2)$ nagyságrendű, ami sokkal jobb közelítést eredményez, így amikor csak lehet célszerű ezt használni.

Több pontra is felírhatjuk a deriváltat a pontosabb közelítés érdekében, például 5 pont bevonásával a centrális differencia formula hibája $O(h^4)$ lesz.

$$f'(x_i) = y_i' = \frac{y_{i-2} - 8\,y_{i-1} + 8\,y_{i+1} - y_{i+2}}{12\,h} + O(h^4)$$

A jobb és baloldali differencia hibája is csökkenthető több pont bevonásával. Az első derivált esetén a jobb és bal oldali differencia 3 pontra felírva:

$$f'(x_i) = y_i' \approx \frac{-3y_i + 4\,y_{i+1} - y_{i+2}}{2\,h} + O(h^2)$$

$$f'(x_i) = y_i' \approx \frac{y_{i-2} - 4y_{i-1} + 3y_i}{2\,h} + O(h^2)$$

MAGASABB RENDŰ DIFFERENCIA HÁNYADOSOK[16]

Magasabb rendű deriváltakra is felírható centrális differencia formula, ezeknél mind $O(h^2)$ lesz a hiba nagyságrendje. Centrális differencia formula második deriváltra 3 pontra:

$$f''(x_i) = y_i'' \approx \frac{y_{i+1} - 2\,y_i + y_{i-1}}{h^2} + O(h^2)$$

Centrális differencia formula harmadik deriváltra 4 pontra:

$$f'''(x_i) = y_i''' \approx \frac{1}{2\,h^3}(y_{i+2} - 2\,y_{i+1} + 2y_{i-1} - y_{i-2}) + O(h^2)$$

Centrális differencia formula negyedik deriváltra 5 pontra:

$$f^{(4)}(x_i) = y_i{}^{(4)} \approx \frac{1}{h^4}(y_{i+2} - 4\,y_{i+1} + 6y_i - 4y_{i-1} + y_{i-2}) + O(h^2)$$

A második derivált esetén a jobb és baloldali differencia 3 pontra:

$$f''(x_i) = y_i'' \approx \frac{y_i - 2\,y_{i+1} + y_{i+2}}{h^2} + O(h)$$

$$f''(x_i) = y_i'' \approx \frac{y_{i-2} - 2\,y_{i-1} + y_i}{h^2} + O(h)$$

A második derivált esetén a jobb és baloldali differencia 4 pontra:

$$f''(x_i) = y_i'' \approx \frac{2y_i - 5\,y_{i+1} + 4\,y_{i+2} - y_{i+3}}{h^2} + O(h^2)$$

$$f''(x_i) = y_i'' \approx \frac{-y_{i-3} + 4y_{i-2} - 5\,y_{i-1} + 2y_i}{h^2} + O(h^2)$$

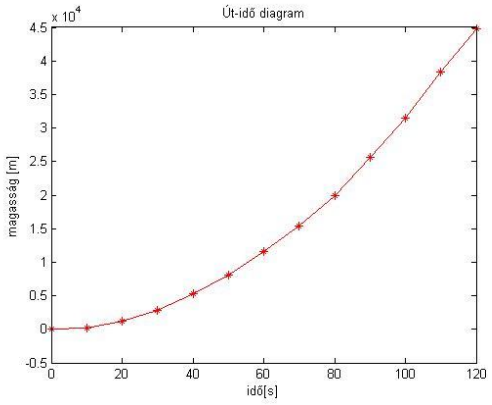
[16] Otthoni átnézésre

DIFFERENCIA HÁNYADOSOK ALKALMAZÁSA

Adottak egy űrsikló helyzetének magasság adatai a kilövés utáni első két percben:

t(s)	0	10	20	30	40	50	60	70	80	90	100	110	120
h(m)	-8	241	1244	2872	5377	8130	11617	15380	19872	25608	31412	38309	44726

Határozzuk meg az űrsikló sebességét az idő függvényében! Ahol lehet használjunk centrális differencia formulát! Töltsük be az adatokat az ursiklo.txt fájlból és ábrázoljuk a magassági pozíciót a idő függvényében!

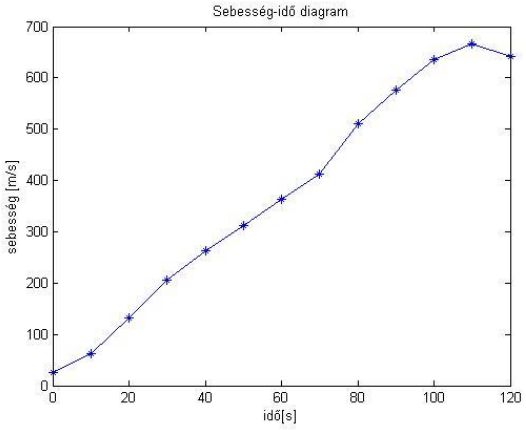



```
> clc; close all; clear all;
> adat = load('ursiklo.txt')
> t = adat(:,1);x = adat(:,2);
> figure(1); plot(t,h,'r*-')
> xlabel('idő[s]')
> ylabel('magasság [m]')
> title('Út-idő diagram')
```

A sebesség az idő szerinti első derivált. Centrális formulát csak a második ponttól az utolsó előtti pontig tudunk használni. Az első pontban csak jobboldali (előremutató) differencia formulát, az utolsó pontban pedig csak baloldali (hátramutató) formulát használhatunk. Írjunk egy függvényt, ami kiszámolja minden pontban az első deriváltat, ahol lehet centrális differencia formulát alkalmazva!

```
> function dx = derivalt(x,y)
> % Numerikus deriválás differencia hányadosok alkalmazásával
>     n = length(x);
>     dx(1) = (y(2)-y(1))/(x(2)-x(1)); % jobboldali differencia
>     for i = 2:n-1
>        dx(i) = (y(i+1)-y(i-1))/(x(i+1)-x(i-1)); % centrális diff.
>     end
>     dx(n) = (y(n)-y(n-1))/(x(n)-x(n-1)); % baloldali differencia
> end
```

Számoljuk ki az első deriváltat a fenti függvényt használva!



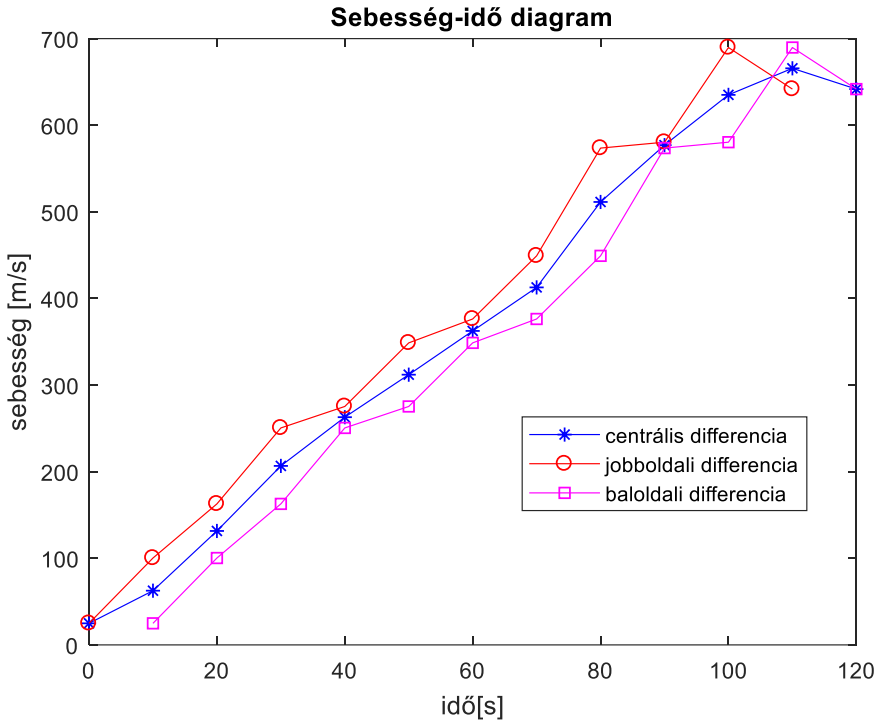

```
> v = derivalt(t,x)
> figure(2); plot(t,v,'b*-')
> xlabel('idő[s]');
> ylabel('sebesség [m/s]')
> title('Sebesség-idő diagram')
```

A Matlab beépített függvénye a **diff** is használható derivált számítására, mind numerikus, mind szimbolikus esetben. A **diff** parancsot numerikusan használva azonban csak egyoldali differenciákat számolhatunk vele, centrális differenciákat nem, ugyanis a **diff** parancs csak annyit tesz, ha egy vektor elemire meghívjuk, hogy kiszámolja a szomszédos elemek különbségét. Az eredménynek eggyel kevesebb eleme lesz, mint a bemenetnek volt. Határozzuk meg

a sebességeket és gyorsulásokat jobb és baloldali differencia különbségeket használva a **diff** paranccsal!

```
> % egyoldali differenciával beépített függvénnyel (diff)
> dx = diff(x);
> dt = diff(t);
> dxdt = dx./dt
> figure(2); hold on
> plot(t(1:end-1),dxdt,'ro-') % jobboldali/előremutató differencia
> plot(t(2:end),dxdt,'ms-') % baloldali/hátramutató differencia
> legend('centrális differencia','jobboldali differencia',...
>     'baloldali differencia','Location','best')
```

Az előremutató (jobboldali) és a hátramutató (baloldali) differenciák számítása lényegében megegyezik, jobboldali differenciákat az első ponttól az utolsó előttiig tudunk számítani, baloldalit pedig a 2. ponttól az utolsóig. A megjelenítésnél láthatjuk a különbséget, az értékek megegyeznek, csak eggyel el vannak csúsztatva a két esetben. Az ábra alapján látható, hogy a centrális differenciák használata simább görbét eredményezett, mint az egyoldali differenciáké. Hasonlóképp számíthatnánk a gyorsulásokat is a sebesség deriválásával!

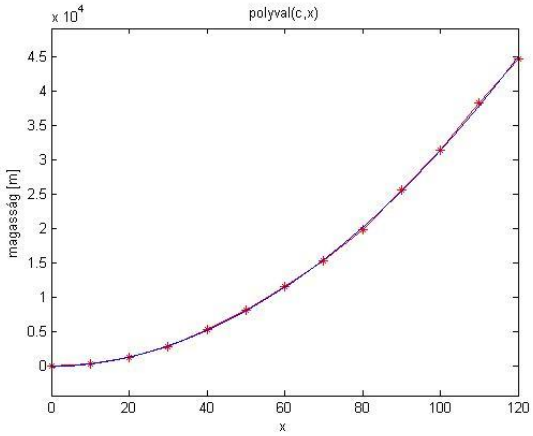


Az alkalmazott **deriválť.m** függvény a 2. ponttól az utolsó előttiig alkalmaz $O(h^2)$ hibájú centrális differencia formulát, a két szélső pontban pedig $O(h)$ hibájú egyoldali differencia formulát (az ábrán látjuk, hogy a két szélső pontban az értéke megegyezik a jobb ill. baloldali formuláéval). Lehetne pontosítani a végpontokban is, $O(h^2)$ hibájú megoldást alkalmazva 3-3 pont bevonásával, a korábban ismertetett képletek alapján:

```
> % Az első és az utolsó pontban is O(h^2) hibájú közelítést alkalmazva
> n=numel(v)
> v1 = (-3*x(1) + 4*x(2)-x(3))/(t(3)-t(1)) %   -12.8000
> vn = (x(n-2) - 4*x(n-1) + 3*x(n))/(t(n)-t(n-2)) % 617.700
```

DERIVÁLÁS FÜGGVÉNYILLESZTÉSSEL (SZIMBOLIKUS DERIVÁLÁS, POLINOM DERIVÁLÁSA)

Nézzük meg a másik megközelítést is, amikor a pontokra egy közelítő függvényt illesztünk. Legyen most ez egy másodfokú polinom!

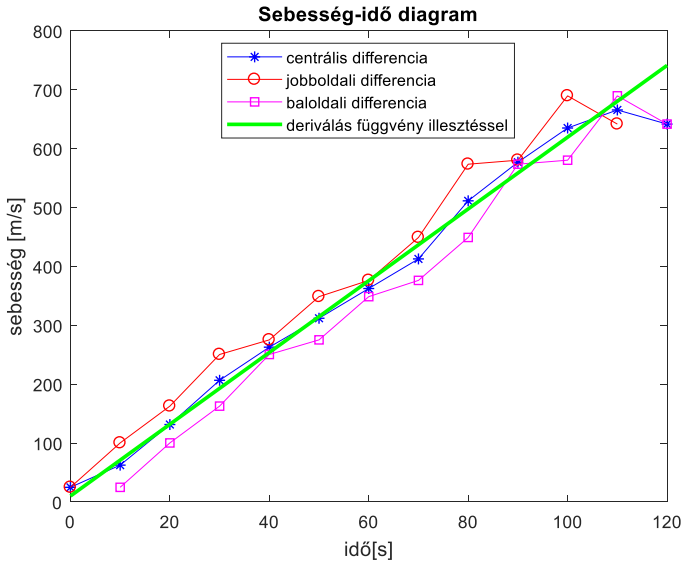



```
> % deriválás függvény illesztéssel
> c = polyfit(t,x,2)
> % c = 3.0470   10.1151  -89.3626
> p = @(x) polyval(c,x)
> figure(1); hold on;
> fplot(p,[min(t),max(t)]);
```

Az ábra alapján látjuk, hogy nagyon jól illeszkedik a pontokra a parabola. Ha az együtthatókkal definiálnánk szimbolikus alakban a polinomot (**syms**), akkor deriválásra használhatnánk a **diff** parancsot, mint azt már korábban láttuk. Azonban algebrai polinomok esetében van egy egyszerűbb megoldás is. A **polyder** parancsot használva nincs szükség szimbolikus formába alakítani a megoldást. Nézzük meg mindkét megoldást!

```
> % sebesség szimbolikus deriválással
> syms x
> ps = c(1)*x.^2 + c(2)*x + c(3)
> % (3050*x^2)/1001 + (50626*x)/5005 - 6288336567612965/70368744177664
> v2 = diff(ps,x)
> % (6100*x)/1001 + 50626/5005
> v2 = matlabFunction(v2)
> % vagy egyszerűbben polinom deriválás beépített Matlab paranccsal
> c1 = polyder(c) % c1 = 6.0939   10.1151
> v2 = @(x) polyval(c1,x)
> figure(2)
> fplot(v2,[min(t),max(t)],'g','LineWidth',2)
> legend('centrális differencia','jobboldali differencia',...
>     'baloldali differencia','deriválás függvény
illesztéssel','Location','best')
```

A derivált ebben az esetben még simább lefutású lett, egy egyenes!

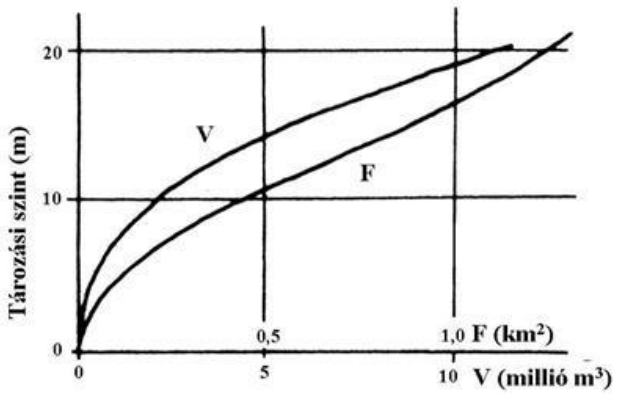


ÖNTÖZŐVÍZ TÁROLÓ PÁROLGÁSI, SZIVÁRGÁSI VESZTESÉGEI

Nézzünk meg egy jellegzetes építőmérnöki feladatot numerikus deriválásra! Becsüljük meg, hogy egy öntözővíz-tározónak egy hónap során hogyan alakultak a párolgási és elszivárgási veszteségei ($Q_{loss}(t)$), ha ismert a hozzáfolyás ($Q_{in}(t)$) és a vízkivétel ($Q_{out}(t)$) vízhozamának idősora, a vízállás idősora ($z(t)$), valamint a tározó térfogati jelleggörbéje. Az alapegyenletet a következő: a hozzáfolyásból kivonva a vízkivételt és a párolgási/elszivárgási veszteséget megkapjuk a térfogat változását:

$$Q_{in}(t) - Q_{out}(t) - Q_{loss}(t) = A(z)\frac{\mathrm{d}z}{\mathrm{d}t}$$

ahol $\frac{\mathrm{d}z}{\mathrm{d}t}$ a vízszint megváltozása, $A(z)$ pedig a tározó adott vízszinthez tartozó felszíne, amit a tározó jelleggörbéjéből határozhatunk meg.



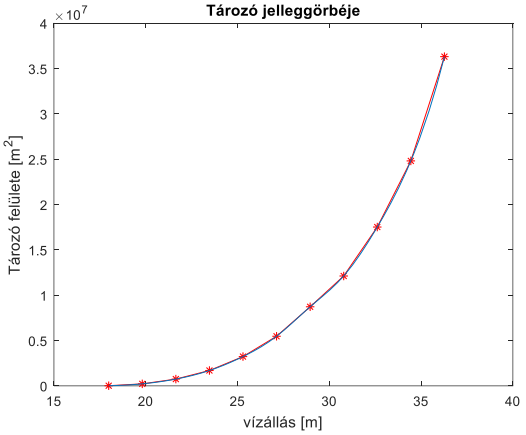
Az ábrán egy minta tározó jelleggörbéje látható, amiről leolvasható, hogy egy adott vízszint (tározási szint) esetén mekkora lesz a tárolt víztérfogat, illetve a tározó nyílt vízfelszíne.

A kérdéses tározó esetén rendelkezésünkre áll a tározó jelleggörbéje a felszínre vonatkozóan, hogy adott vízszinthez [m] mekkora tározó felszín [m^2] tartozik.

vízszint *h* [m]	17.983	19.812	21.641	23.470	25.298	27.127	28.956	30.785	32.614	34.442	36.271
felszín *A* [m^2]	16177	222431	748178	1694521	3229775	5471806	8723345	12120476	17519486	24815707	36316941

A fenti adatok megtálalhatóak a **jelleggorbe.txt** fájlban is. Olvassuk be az adatokat, majd válasszuk szét a változókat. Tüntessük fel egy új ábrában a pontokat és illesszünk rá egy köbös másodrendű spline-t, függvény alakban! Az x tengelyen most a vízszint értékek legyenek.

```
> %% Víztározó
> clc; clear all; close all;
> % Jelleggörbe
> jelleg = load('jelleggorbe.txt');
> h = jelleg(:,1), A = jelleg(:,2),
> figure(1); plot(h,A,'r*-')
> F = @(x) spline(h,A,x)
> hold on; fplot(F, [min(h), max(h)])
> title('Tározó jelleggörbéje')
> xlabel('vízállás [m]');
> ylabel('Tározó felülete [m^2]')
```

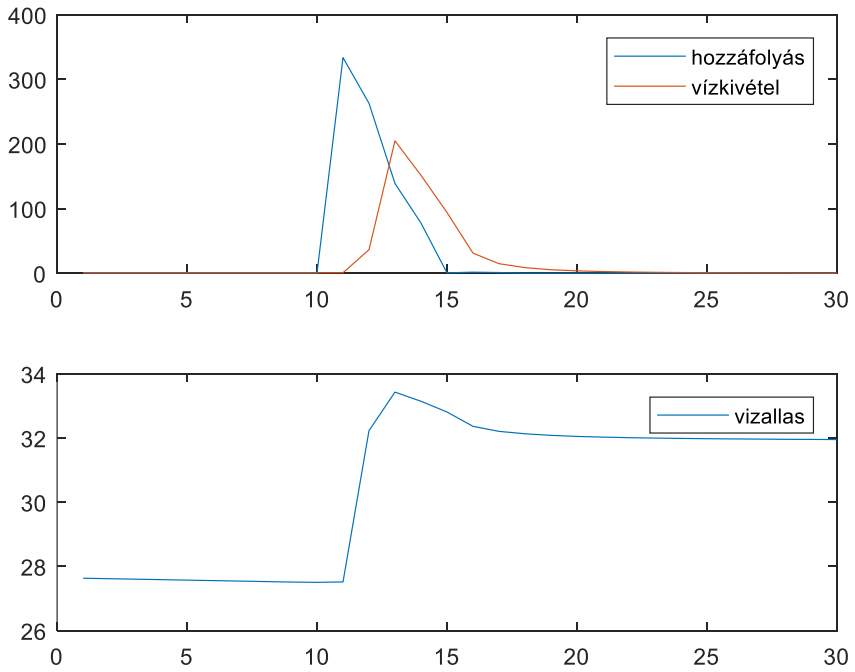


Rendelkezésünkre állnak mérési adatok 30 napra vízállásra ($z(t)$), a hozzáfolyás ($Q_{in}(t)$) és a vízkivétel ($Q_{out}(t)$) mennyiségére. Ezek az **idosor.txt** fájlban találhatóak, ahol négy oszlopban a következő adatok vannak: időpont - [nap], Q_{in}- [m^3/s], vízállás - [m], Q_{out} - [m^3/s].

időpont - t [nap]	hozzáfolyás - Q_{in} [m^3/s]	vízállás - z [m]	vízkivétel - Q_{out} [m^3/s]
1.	0.08	27.630	0.70
2.	0.08	27.616	0.70
⋮	⋮	⋮	⋮

Válasszuk szét az adatokat és a subplot parancsot használva készítsünk egy ábrát két egymás alatti grafikonnal, a felsőben feltüntetve a hozzáfolyás és a vízkivétel idősorát, az alsón pedig a vízszint megváltozását!

```
> % Idősor adatok
> idosor = load('idosor.txt');
> t=idosor(:,1), Qin=idosor(:,2),z=idosor(:,3), Qout=idosor(:,4)
> figure(2); subplot(2,1,1);
> plot(t,Qin,t,Qout); legend('hozzáfolyás','vízkivétel')
> subplot(2,1,2); plot(t,z); legend('vizallas')
```



A jelleggörbe alapján számoljuk ki az egyes vízállásokhoz tartozó felszín adatok idősorát $A(z)$!

```
> % Az adott idősorhoz tartozó tározó felszínek kiszámítása
> A = F(z)
```

Az alapegyenletünk a következő: $Q_{in}(t) - Q_{out}(t) - Q_{loss}(t) = A(z)\frac{\mathrm{dz}}{\mathrm{dt}}$.

Ebből ismerjük a $Q_{in}(t), Q_{out}(t)$, $A(z)$ idősorokat és a vízszint idősorát is $(z(t))$. Ahhoz, hogy ki tudjuk számolni a párolgási/elszivárgási veszteség idősorát $(Q_{loss}(t))$, a vízszint idő szerint változását vagyis az első deriváltat kellene meghatározzuk. Egy

adott napon mért hozzáfolyás, vízkivétel hatása mindig a másnap reggel mért vízszintekben fog csak megjelenni, ezért itt előremutató, vagy jobboldali differenciákra lesz szükségünk!

Határozzuk meg a vízállás idő szerinti első deriváltját ($\frac{dz}{dt}$), jobboldali (előremutató) differencia formulát használva ahol lehet (az utolsó pontnál baloldali differencia formulával)! Ehhez módosíthatjuk a korábban a centrális differencia formulára megírt **derivalt.m** függvényünket, vagy megoldhatjuk a Matlab numerikus deriválás parancsát alkalmazva is. Figyeljünk oda, hogy az idő mértékegységek megegyezzenek! A végeredmény a további számítások érdekében oszlopvektor legyen!

A jobb oldali vagy előremutató differencia képlete: $f'(x_i) = y_i' \approx \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$

A baloldali vagy hátramutató differencia: $f'(x_i) = y_i' \approx \frac{y_i-y_{i-1}}{x_i-x_{i-1}}$

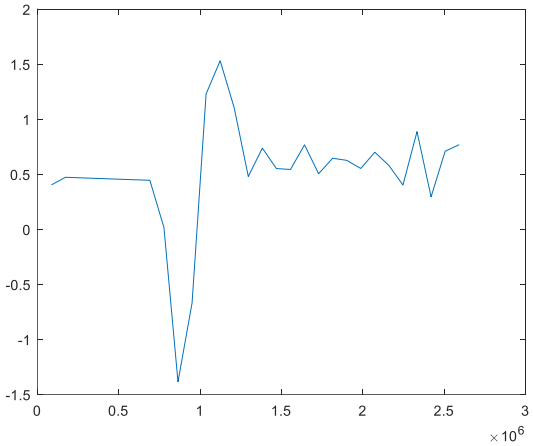
A módosított függvény neve legyen **jobbderivalt.m**!

```
function dx = jobbderivalt(x,y)
% Numerikus deriválás differencia hányadosok alkalmazásával
% Utolsó előtti pontig jobb, utolsó pontban baloldali differencia
formulával
    n = length(x);
    for i = 1:n-1
       dx(i) = (y(i+1)-y(i))/(x(i+1)-x(i));
    end
    dx(n) = (y(n)-y(n-1))/(x(n)-x(n-1));
end
```

Számítsuk ki a vízállás változását a fenti függvénnyel! Ne felejtsük el a napokban megadott időket átváltani másodpercekre!

```
% vízállás változás
ts = t*24*3600;
dz = jobbderivalt(ts,z);
dz=dz'
% vagy beépített diff függvényt használva
dz2 = diff(z)./diff(ts); dz2 = [dz2; dz2(end)]
```

Számoljuk ki a tárolt víz térfogat változásának idősorát is: $\frac{dV}{dt} = A(z)\frac{dz}{dt}$, majd az alapegyenletet használva számolja ki a párolgási/szivárgási veszteség idősorát (Q_{loss}) [m^3/s]-ban!



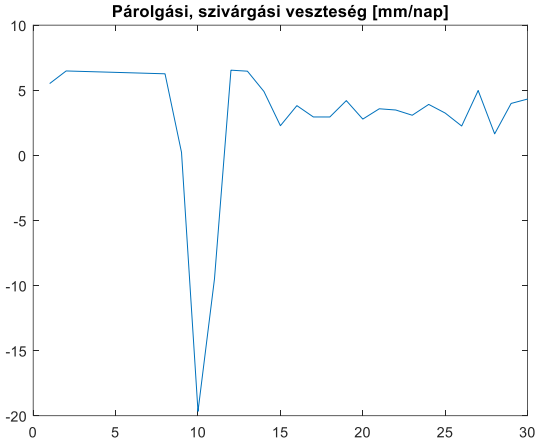

```
% Térfogat változás, párolgási,
szivárgási veszteség [m^3/s]
dV = A.*dz
Qloss = Qin - Qout - dV;
figure(3); plot(ts,Qloss)
```

A párolgás a vízfelszínnel arányos (h_v), értékét mm/nap mennyiségben szokták megadni. Számoljuk át a [m^3/s]-ban megkapott értékeket mm/nap mennyiségre az alábbi képletet használva:

$$h_v = \frac{Q_{loss} \cdot 86400 \cdot 1000}{A}$$

ahol Q_{loss} [m^3/s]-ban adott, 86400 a másodpercek száma egy napban, A pedig az adott időponthoz tartozó vízfelszín [m^2]. Az 1000-es szorzó azért van benne, mert a végeredményt nem méterben, hanem mm-ben kapjuk (/nap). A kapott értékeket ábrázoljuk is! (Megj: A párolgási veszteség max. napi 10 mm szokott lenni, értéke lehet negatív is, csapadék esetén)

```
> % Párolgási, szivárgási veszteségek mm-ben
> veszt_mm = Qloss*86400*1000./A
> figure(3); plot(t,veszt_mm);
> title('Párolgási, szivárgási veszteség [mm/nap]')
```



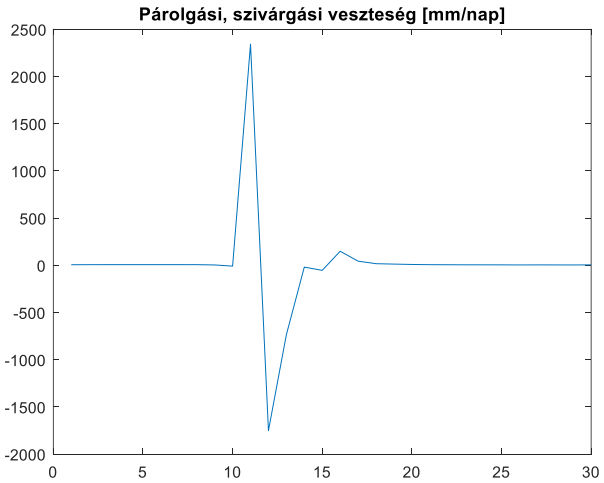
Mi történt volna, ha nem előremutató differencia formulákat használunk (aminek itt fizikai tartalma van), hanem itt is centrális differencia formulákat alkalmazzuk, mondván, hogy pontosabb? Cseréljük ki a

```
> dz = jobbderivalt(ts,z);
```

parancsot a következőre (ami centrális differencia formulát használ):

```
> dz = derivalt(ts,z);
```

Az eredmény szemmel láthatóan képtelenség, hiszen egy nap alatt nem szokott 2.3 m magasságú vízoszlop elpárologni, se 1.8 m csapadék esni.



DERIVÁLÁS TÖBBVÁLTOZÓS ESETBEN

A **gradiens** a deriválás általánosítása többváltozós esetre. Egy függvény deriváltjának értékét egyváltozós esetben számíthatjuk, értéke egy skalár lesz. Több változó esetén a gradiens veszi át a helyét, mely ellentétben a deriválttal, már vektort ad vissza, nem skalárt, eredménye egy vektormező lesz. Kétváltozós esetben egy *f* függvény gradiens vektora a következő:

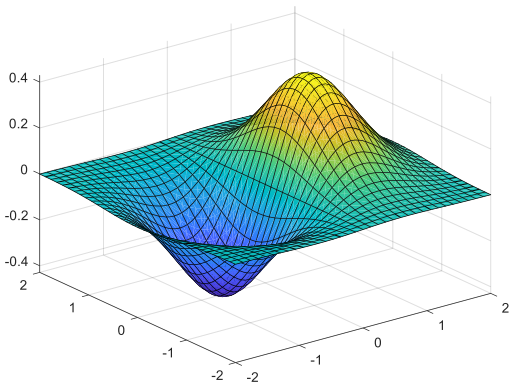
$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix};$$

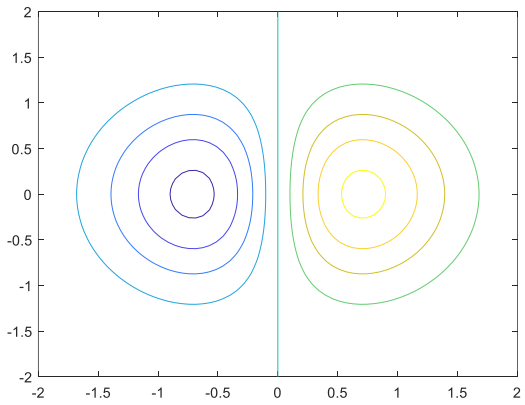
Ugyanúgy, ahogy a derivált az érintő meredekségét adja meg, a gradiens a felület legnagyobb meredekségű irányába mutat, megadja az adott pontban a függvény legnagyobb megváltozásának irányát, értékét.

Határozzuk meg a következő kétváltozós függvény gradiensét, és ábrázoljuk a vektormezőt az $x \in [-2,2]$ és $y \in [-2,2]$ tartományon!

$$f(x,y) = x\, e^{-(x^2+y^2)}$$

```
>  %% Kétváltozós felület definiálása
>  clc; clear all; close all;
>  f = @(x,y) x .* exp(-(x.^2 + y.^2));
>  figure(1)
>  fsurf(f,[-2,2,-2,2])
>  figure(2)
>  fcontour(f,[-2,2,-2,2])
```



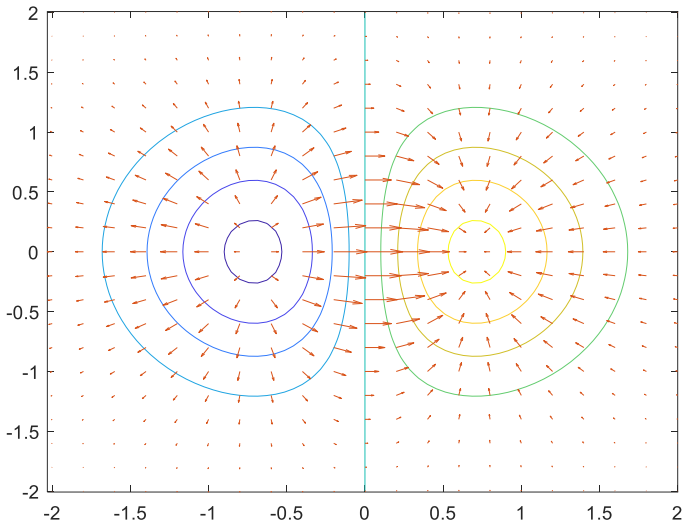


Számítsuk ki a gradiensvektor elemeit, vagyis az x és y szerinti parciális deriváltakat szimbolikusan! Ehhez használhatnánk szimbolikusan a **diff** parancsot is, vagy egy lépésben a **gradient** parancsot.

```
>  %% Gradiens számítás szimbolikusan
>  syms x y
>  fs = x .* exp(-(x.^2 + y.^2));
>  G = gradient(fs, [x, y])
>  %  exp(- x^2 - y^2) - 2*x^2*exp(- x^2 - y^2)
>  %                   -2*x*y*exp(- x^2 - y^2)
```

Ábrázoljuk a vektormezőt a szintvonalas ábrán, úgy, hogy kiszámoljuk egy rácsháló pontjaiban a gradiens értékeit! Vektormezőt a **quiver** paranccsal jeleníthetünk meg.

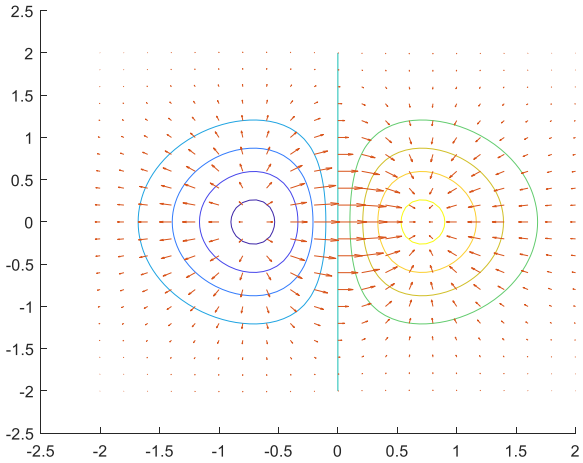
```
> [X, Y] = meshgrid(-2:0.2:2,-2:0.2:2);
> gx = matlabFunction(G(1))
> gy = matlabFunction(G(2))
> GX = gx(X,Y); GY = gy(X,Y);
> hold on;
> quiver(X,Y,GX,GY)
```



A **gradient** parancsot használhatjuk numerikusan is a gradiensvektor értékeinek kiszámítására.

```
> %% Gradiens számítás numerikusan
> Z = f(X,Y);
> [px,py] = gradient(Z);
> figure(3); hold on;
> fcontour(f,[-2,2,-2,2])
> quiver(X,Y,px,py)
```

Itt először kiszámítottuk az X,Y rácspontokban a függvény értékét (Z), majd egyenletesnek feltételezve a rácsháló osztását, a gradiens vektor értékeit (px,py).



A második parciális deriváltakat tartalmazza a Hesse mátrix, ami kétváltozós esetben így írható fel:

$$H(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x\,\partial y} \\ \frac{\partial^2 f}{\partial y\,\partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

Egy *f* függvény *x* változó szerinti második parciális deriváltját számíthatjuk a **diff(f,x,2)** paranccsal, vagy az egész Hesse mátrixot is megadhatjuk egyben szimbolikusan a **hessian** parancs alkalmazásával. Határozzuk meg az előző függvény Hesse mátrixát és számoljuk ki a második deriváltak értékét az x=0.5 és y = 0.7 helyen!

```
> %% Hesse mátrix
> d2x = diff(fs,x,2)
> % 4*x^3*exp(- x^2 - y^2) - 6*x*exp(- x^2 - y^2)
> H = hessian(fs)
> % [   4*x^3*exp(- x^2 - y^2) - 6*x*exp(- x^2 - y^2), 4*x^2*y*exp(-
  x^2 - y^2) - 2*y*exp(- x^2 - y^2)]
> % [ 4*x^2*y*exp(- x^2 - y^2) - 2*y*exp(- x^2 - y^2), 4*x*y^2*exp(-
  x^2 - y^2) - 2*x*exp(- x^2 - y^2)]
> Hfv = matlabFunction(H)
> Hfv(0.5,0.7)
> %    -1.1928   -0.3340
> %    -0.3340   -0.0095
```

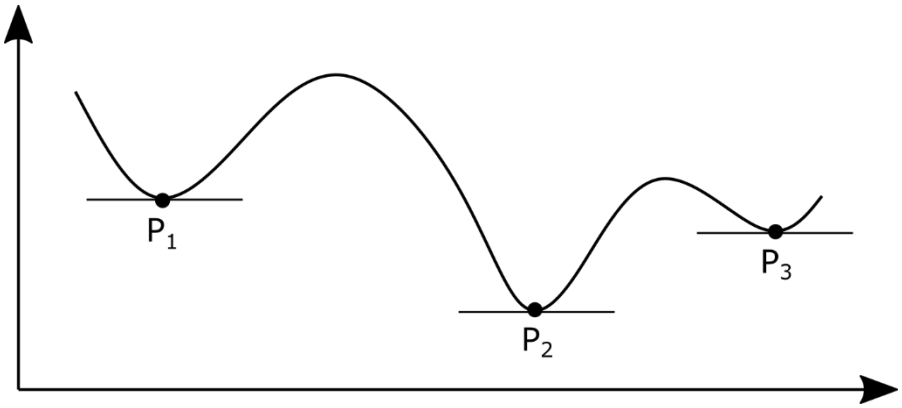
A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

polyder	-	Algebrai polinom deriváltjának számítása
diff(f,x,2)	-	f szimbolikus kifejezés 2. deriváltja x szerint
gradient	-	Gradiensek számítása numerikusan, szimbolikusan
quiver	-	vektormező megjelenítése
hessian	-	Hesse-mátrix, az f(x) függvény második parciális deriváltjainak a mátrixa

13. FELTÉTEL NÉLKÜLI OPTIMALIZÁCIÓ

Az optimalizáció, egy függvény szélsőérték helyének a meghatározása. Ez a feladat a mérnöki gyakorlatban is sokszor előfordul, meg kell határozni például egy tartószerkezet maximális elmozdulásának a helyét, geodéziai mérések kiegyenlítésekor egy pont legkisebb hibával rendelkező helyzetét, vízminőség vizsgálatnál a maximális szennyeződés mértékét.

A sokféle felmerülő feladat megoldására sok módszert dolgoztak ki. A kidolgozott numerikus eljárások többnyire minimum hely keresésére vonatkoznak, amennyiben a meghatározandó szélsőérték nem minimum hanem maximum, akkor azt a függvény (-1)-szeresének minimumával lehet megtalálni, $max(f(x)) = min(-f(x))$. A szélsőértéket mindig egy adott intervallumban, tartományban vizsgáljuk. Az adott tartományon belül lehetnek lokális minimumok, ahol a pont akármilyen kicsiny környezetében a függvényérték nagyobb, mint ebben a pontban (P_i pontok az ábrán). Amennyiben egy tartományban több lokális minimum is van, eltérő függvényértékekkel, akkor a legkisebb függvényértékhez tartozó pont a globális minimum (az ábrán P_2).

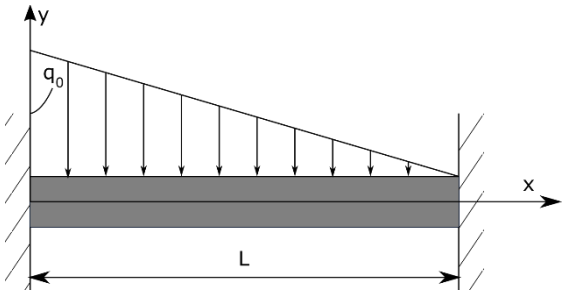


Beszélhetünk feltétel nélküli és feltételes szélsőértékről is (vagy megkötés nélküli és megkötéses optimalizációról). A feltételes szélsőérték feladatok esetében úgy keressük a függvény minimumát, hogy közben a pontoknak ki kell elégíteniük valamilyen megkötést, feltételt is. Itt lehet egy vagy több feltétel, ezek lehetnek egyenletekkel vagy egyenlőtlenségekkel megadva, lehetnek lineárisak vagy nemlineárisak is. A különböző esetekben más-más módszert lehet alkalmazni (pl. Lagrange-módszer, büntetésfüggvény módszer, Karush-Kuhn-Tucker-feltételek, lineáris programozás). Most idő hiányában csak a feltétel nélküli szélsőérték feladatokkal fogunk foglalkozni. Ezeknek a megoldására is számos módszer létezik.

EGYVÁLTOZÓS FÜGGVÉNY SZÉLSŐÉRTÉK KERESÉSE

LEHAJLÁS VIZSGÁLAT

Nézzünk meg most rugalmasságtanból egy példát, ahol a maximális lehajlás helyét és értékét keressük! Egy két végén befogott `I` keresztmetszetű gerendára lineárisan megoszló terhelés jut az ábra szerint. Az y tengely irányú lehajlás a következő összefüggéssel számítható:



$$y = \frac{q_0}{120LEI}(x^5 - 5Lx^4 + 7L^2x^3 - 3L^3x^2),$$

ahol q_0=15kN/m=15N/mm, E=70000 N/mm^2, L=3000 mm, I = 5.29x10^7 mm^4.

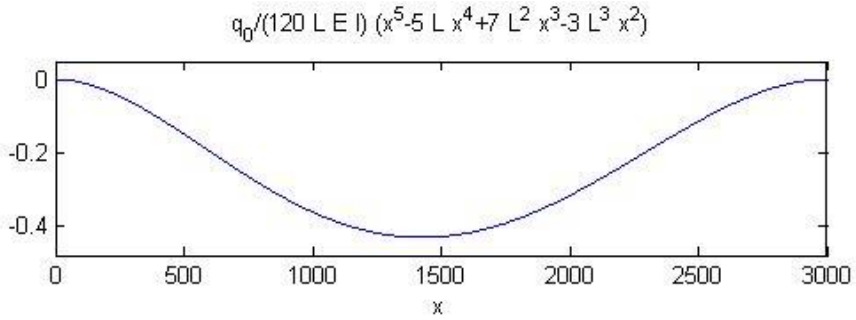
Mekkora lesz a lehajlás 1 és 2 m-nél? Keressük meg azt a helyet, ahol legnagyobb a lehajlás és határozzuk meg a lehajlás értékét!

Először ábrázoljuk a lehajlásokat!

```
> %% Lehajlás számítás
> % Változók megadása:
> E = 70000; I = 5.29e7; q0 = 15; L = 3000; EI = E*I;
```

Célszerű összevonni az E és I változókat egybe (EI), hogy ne keveredjenek az 'e' Euler-féle számmal (2.71…) és az 'i' képzetes egységgel ($\sqrt{-1}$). Ez a szimbolikus számításoknál problémát okozhatna.

```
> y = @(x) q0/(120*L*EI)*(x^5-5*L*x^4+7*L^2*x^3-3*L^3*x^2)
> % lehajlások ábrázolása
> figure(1); fplot(y,[0 3000])
> y(1000), y(2000)  % lehajlás 1 ill. 2 m-nél: -0.3601 és -0.3151 mm
```



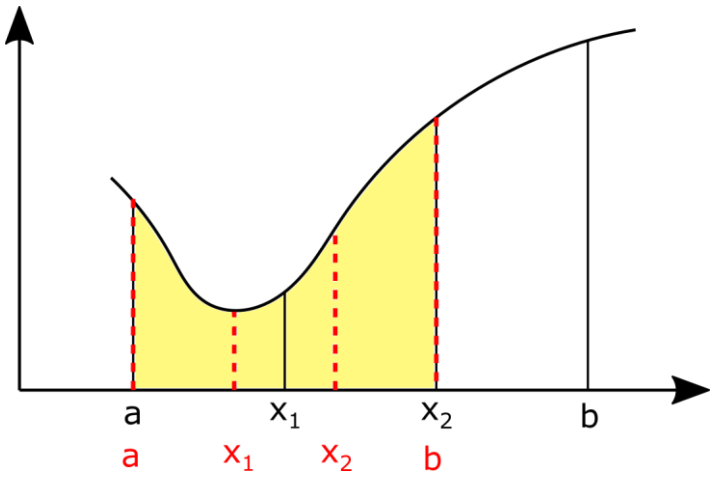
Nézzünk meg két módszert, amivel meg lehet keresni egy egyváltozós függvény minimumát!

INTERVALLUM MÓDSZER (TERNARY SEARCH)

Az intervallum módszer hasonlít a nemlineáris egyenleteknél tanult zárt intervallum módszerekhez. Kezdőértéknek itt is egy intervallumot kell felvenni [a,b], ahol most nem egy zérushelye, hanem egy minimuma lesz a függvénynek, vagyis unimodális lesz függvény (a minimumhelyig a függvény monoton csökken, utána monoton nő). A zárt intervallum módszerekhez hasonlóan itt is valamilyen módon szűkíteni kell ezután az

intervallumot, amíg megtaláljuk a megoldást. Ehhez most két belső pontot vegyünk fel (x_1, x_2) és vizsgáljuk meg a függvényértékeket ezekben a pontokban!

Mivel a függvény a minimumhelyig monoton csökken, utána pedig monoton nő, a minimumhely csak a legkisebb függvényértéket adó pont és a két szomszédja közötti intervallumban lehet. Tehát, ha $f(x_1) < f(x_2)$ akkor a minimumhely biztosan az $[a, x_2]$ intervallumban lesz, ha $f(x_1) > f(x_2)$, akkor pedig biztosan az $[x_1, b]$ intervallumban. Lásd az ábrát! Ezután az új intervallumban újra felveszünk két belső pontot és addig ismételjük az eljárást, míg az intervallum egy megadott küszöbérték alá nem csökken.



A módszer mindig konvergálni fog (amennyiben az intervallumon belül a függvény unimodális). A kérdés az, hogyan érdemes felvenni x_1, x_2 helyét, hogy minél kevesebb iterációval, számítással eljussunk a végeredményhez. Az egyik megközelítés, hogy egyenletesen vesszük fel a pontokat az intervallum 1/3-ában és 2/3-ában ('ternary search algorithm'). Nézzük meg hogyan oldhatjuk meg ezt Matlab-ban. Lásd az intervallum.m fájt!

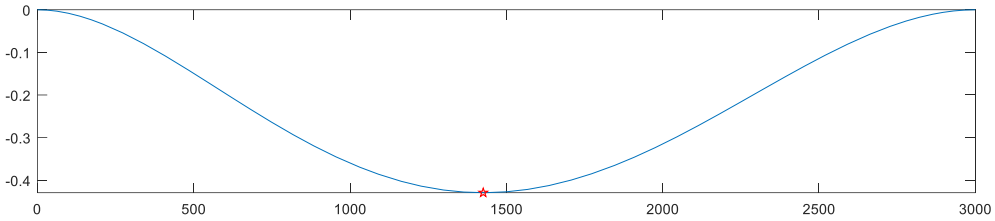
```
>  function [x, i] = intervallum(f, a, b, tol)
>  i = 1;
>   x1 = a + 1/3*(b-a);
>   x2 = b - 1/3*(b-a);
>
>   while abs(x2-x1) > tol
>     if f(x1) < f(x2)
>       b = x2;
>     else
>       a = x1;
>     end
>     i = i+1;
>     x1 = a + 1/3*(b-a);
>     x2 = a - 1/3*(b-a);
>   end
>   x = (x1+x2)/2;
>  end
```

Keressük meg ezzel a módszerrel a maximális lehajlás helyét! Hiába beszélünk maximális lehajlásról, itt is egy minimum hely meghatározásáról van szó, ha megfigyeljük az első ábrán a koordináta rendszert, akkor látni fogjuk, hogy a lehajlások negatív elmozdulás értékeket jelentenek, tehát a maximális lehajlás a legkisebb y

koordinátát jelenti. A kezdő unimodális intervallum legyen a [1000, 2000] az ábra alapján!

```
> % intervallum módszer - egyenlő felosztás
> [x1 i1] = intervallum(y,1000,2000,1e-6)
> % x1 =  1.4259e+03; i1 =  50
> y1 = y(x1) % -0.4293
> hold on; plot(x1,y1,'rp')
```

Tehát összesen 50 iteráció alatt találtuk meg a minimumhelyet (a maximális lehajlás helyét) 1425.9 mm-nél van, értéke pedig -0.4293 mm lett.



ARANYMETSZÉS MÓDSZERE (GOLDEN-SECTION SEARCH)

Van azonban hatékonyabb felvétele is a pontoknak, mint az egyenletes felvétel. Használjuk az aranymetszési arányt! Ez az arány a természetben is gyakran előfordul és a művészetekben is gyakran használják. Az aranymetszési aránnyal úgy oszthatunk fel egy L szakaszt két részre ($L = L1 + L2$), hogy a nagyobbik szakasz aránya a teljes hosszhoz ugyanakkora legyen, mint a kisebbik szakasz aránya a nagyobbikhoz.

$$R = \frac{L2}{L} = \frac{L1}{L2}$$

Fejezzük ki ebből L1-et és L2-t R és L függvényében: $L2 = R \cdot L;\ L1 = L2 \cdot R = L \cdot R^2$. Ezt helyettesítsük be az $L = L1 + L2$ egyenletbe:

$$L = L \cdot R^2 + R \cdot L$$

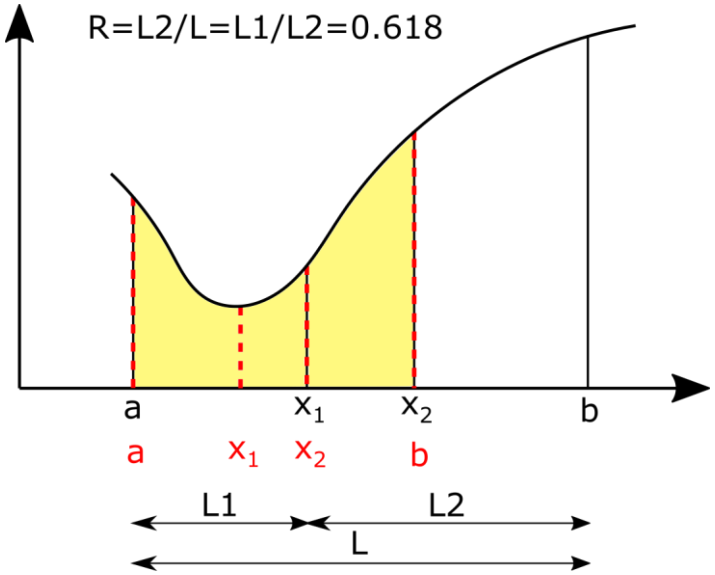
Ezt elosztva L-lel és 0-ra rendezve kapjuk, hogy:

$$R^2 + R - 1 = 0$$

A másodfokú egyenlet egyetlen pozitív gyöke lesz az a keresett arányszám, ahogy a nagyobbik szakasz aránylik az egészhez, vagy a kisebbik szakasz a nagyobbikhoz, ez lesz az aranymetszési arány:

$$R = \frac{\sqrt{5} - 1}{2} = 0.618$$

Nézzük meg, hogyan használhatjuk ezt a hatékonyabb szélsőérték meghatározáshoz, módosítva az intervallum módszernél a belső pontok felvételét!



Vegyük fel úgy a belső pontokat szimmetrikusan, hogy $0.618 \cdot L$ távolságra legyen a két pont a szakasz egyik és másik végétől. Ebben az esetben is szűkítsük az intervallumot a belső pontok függvényértékei alapján, tehát a minimumhely a legkisebb függvényértéket adó pont és a két szomszédja közötti intervallumban lehet most is. A különbség az előzőekhez képest az, hogy az aranymetszés tulajdonságai miatt most az új intervallum egyik belső pontja meg fog egyezni az előző intervallum egyik korábbi belső pontjával! Az ábrán az új intervallum x_2 pontja meg fog egyezni az előző intervallum x_1 pontjával! Ez azt jelenti, hogy ebben a pontban nem kell újra kiszámolni a függvényértéket, elég csak a másik belső pont ezt megtenni. Ez különösen bonyolult függvények esetében lehet jelentős időnyereség. Nézzük meg Matlabban hogyan tudnánk ezt megvalósítani (aranymetszes.m)!

```
> function [x, i] = aranymetszes(f, a, b, tol)
> i = 1;
> R = (sqrt(5)-1)/2;
> x1 = b - R*(b-a);
> x2 = a + R*(b-a);
> f1 = f(x1); f2 = f(x2);
>
> while abs(x2-x1)>tol
>   if f1 < f2
>      b = x2;
>      x2 = x1; f2 = f1; % ezt átvesszük az előző iterációból!
>      x1 = b - R*(b-a);
>      f1 = f(x1);  % ezt számoljuk
>   else
>      a = x1;
>      x1 = x2; f1 = f2; % ezt átvesszük az előző iterációból!
>      x2 = a + R*(b-a);
>      f2 = f(x2);  % ezt számoljuk
>   end
>   i = i+1;
> end
> x = (x1+x2)/2;
```

Az első iterációban még ki kell számítani x_1, x_2 pontban is a függvény értékeket, de utána már elég csak az egyiket számítani, a másikat átvehetjük a korábbi iterációból!

(Megj. : Az intervallum/aranymetszés módszere megoldható rekurzív algoritmussal is, lásd a golden.m fájlt.)

Keressük meg ezzel is maximális lehajlás helyét! Ábrázoljuk is az eredményt!

```
> % aranymetszes módszere
> [x2 i2] = aranymetszes(y,1000,2000,1e-6)
> % x2 = 1.4259e+03; i2 = 42
> y2 = y(x2) % -0.4293
> plot(x2, y2, 'mo')
```

Most egyrészt a korábbi 50 iteráció helyett 42 iterációs lépés is elég volt a megoldáshoz, viszont, ha a függvény kiértékelések számát nézzük, akkor még nagyobb a különbség. Az egyenlő felosztás esetén minden iterációban 2 függvényértéket kellett kiszámolni, vagyis 50*2=100 függvénykiértékelés történt, az aranymetszés esetében csak az első iterációban kellett 2 kiértékelést végezni, utána már csak iterációnként egyet, vagyis összesen 43-szor kellett kiszámolni a függvény értékét! Ez bonyolult függvények esetében nagy előnyt jelent az egyenletesen felvett pontokkal szemben.

NEWTON-MÓDSZER

Ha a függvény deriváltjának számítása nem okoz gondot akkor megoldhatjuk a szélsőérték keresést úgy is, hogy az első derivált gyökhelyeit megkeressük. Alkalmazzuk most a Newton módszert szélsőérték keresésre! A gyökhelykereséssel szemben nem az $f(x) = 0$ egyenletet, hanem az $f'(x) = 0$ egyenletet oldjuk meg, de ugyanaz az algoritmus használható most is (lásd a korábbi **newton.m** fájlt).

A Newton módszer iterációs képlete zérushely kereséshez: $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$

Ugyanez szélsőérték kereséséhez az $f(x)$-et, $f'(x)$-re cserélve:

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

A Newton módszerrel történő szélsőérték kereséshez szükség van mind az első, mind a második derivált számítására! Oldjuk meg az előző feladatot Newton módszerrel is! Kezdőértéknek válasszuk most az előző intervallum egyik végpontját, az összehasonlíthatóság végett, mondjuk az 2000-et! (Természetesen az ábra alapján pontosabb kezdőérték is választható lenne.).

Az eredeti egyenlet a következő:

$$y = \frac{q_0}{120LEI}(x^5 - 5Lx^4 + 7L^2x^3 - 3L^3x^2),$$

Ennek az első (vagy második) deriváltját nem túl nehéz számítógép nélkül sem meghatározni, mivel egy polinomról van szó. Az első derivált:

$$f(x) = y' = \frac{q_0}{120LEI}(5x^4 - 20Lx^3 + 21L^2x^2 - 6L^3x),$$

Természetesen Matlab-bal is meghatározhatjuk az *x* szerinti deriváltat, szimbolikusan. Ahhoz, hogy össze tudjuk hasonlítani a számítógép nélkül végzett deriválással,

először alakítsuk szimbolikussá az EI, L, q_0 és x változókat, ezekkel definiáljuk a szimbolikus ys függvényt, majd számítsuk ki a deriváltakat szimbolikusan:

```
> %% Newton módszer
> % Deriváltak meghatározása
> syms EI L q0 x
> ys = q0/(120*L*EI)*(x.^5-5*L*x.^4+7*L^2*x.^3-3*L^3*x.^2)
> dx=diff(ys,x)
> % -(q0*(6*L^3*x - 21*L^2*x^2 + 20*L*x^3 - 5*x^4))/(120*EI*L)
> ddx = diff(ys,x,2)
> % -(q0*(6*L^3 - 42*L^2*x + 60*L*x^2 - 20*x^3))/(120*EI*L)
```

Mielőtt függvénnyé alakíthatnánk a szimbolikus kifejezéseket, meg kell adnunk újra (be kell helyettesítenünk) az EI, L és q_0 változók értékeit. Most a kapott eredményeket egyszerűen másoljuk be a függvény definíció eleje után CTRL+C/CTRL+V használatával.

```
> % Alakítsuk át függvénnyé a szimbolikus kifejezéseket!
> E = 70000; I = 5.29e7; q0 = 15; L = 3000; EI = E*I;
> dxf = @(x) -(q0*(6*L^3*x - 21*L^2*x^2 + 20*L*x^3 - 5*x^4))/(120*EI*L)
> ddxf = @(x) -(q0*(6*L^3 - 42*L^2*x + 60*L*x^2 - 20*x^3))/(120*EI*L)
```

Más megoldás lehet a már korábban is alkalmazott **matlabFunction** parancs. Azonban ebben az esetben csak akkor használhatjuk, ha előtte behelyettesítjük az ismeretlen (EI,q_0,L) értékeket a szimbolikus kifejezésekbe, ezt a **subs** paranccsal tehetjük meg. A **subs** parancs segítségével be lehet helyettesíteni egyszerre az összes korábban számként megadott változót, vagy meg lehet adni egy bizonyos változó értékét is.

```
> % Más megoldás
> dx = subs(dx),ddx = subs(ddx)
> dxf = matlabFunction(dx)
> ddxf = matlabFunction(ddx)
```

Végül a megoldás Newton módszerrel:

```
> % megoldás Newton módszerrel
> [xn in] = newton(dxf, ddxf, 2000, 1e-6, 100)
> % xn = 1.4257e+03; in = 3
```

Most mindössze 3 iterációból eljutottunk a megoldáshoz, látszik, hogy ez a módszer sokkal gyorsabban konvergál, amennyiben konvergál.

MATLAB BEÉPÍTETT FÜGGVÉNY ALKALMAZÁSA (FMINSEARCH)

Természetesen a Matlab-nak van saját beépített függvénye is, amivel minimumot lehet keresni, pl. az **fminsearch** parancs. Ez Nelder-Mead szimplex módszert használ.

```
> % Matlab beépített függvény - fminsearch
> y = @(x) q0/(120*L*EI)*(x.^5-5*L*x.^4+7*L^2*x.^3-3*L^3*x.^2)
> xmin = fminsearch(y,2000) % 1.4259e+0
```

Részletekkel együtt:

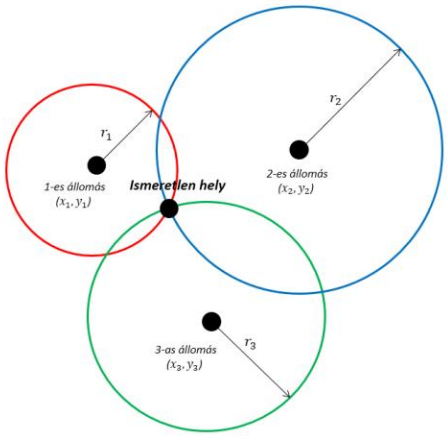
```
> [x,fval,exitflag,output] = fminsearch(y,2000)
> i = output.iterations % i = 25 - iterációk száma
> n = output.funcCount % n = 50 - függvény kiértékelések száma
```

TÖBBVÁLTOZÓS FÜGGVÉNY SZÉLSŐÉRTÉK KERESÉSE

Nem csak egy, hanem többváltozós feladatok esetében is gyakran van szükség szélsőérték meghatározásra, lehet ez egy felület legkisebb, legnagyobb értékű helyének meghatározása, egy térbeli tartó bizonyos pontjainak x,y irányú maximális elmozdulása, úthálózat csomópontjainak ideális megválasztása a távolságok minimalizálásával stb. A megkötés nélküli többváltozós esetben is többféle megoldási módszer közül választhatunk. Használhatunk például többváltozós Newton-módszert, gradiens módszert, Nelder-Mead szimplex módszert is.

POZÍCIÓ MEGHATÁROZÁS TÚLHATÁROZOTT ESETBEN

Nézzünk példát most egy kétváltozós szélsőérték feladat megoldására. A korábban már megismert mobiltelefonos pozíció meghatározásra vonatkozó ívmetszést használó példát folytassuk. A nemlineáris egyenletrendszereknél két mérési eredményünk volt két változóra, a gyakorlatban azonban fölös méréseink is szoktak lenni. 3 vagy több mérés esetén, ha nem teljesen hibátlanok a mérések, akkor maradék ellentmondások adódnak a pozíció meghatározásakor, kiegyenlítésre van szükség. Akárcsak a túlhatározott lineáris egyenletrendszereknél, itt is a legkisebb négyzetek módszerét használva minimalizálhatjuk a hibát, a maradék eltérések négyzetösszegét. A feladat megoldható linearizálással is, de használhatunk többváltozós szélsőérték kereső algoritmusokat is. Az ismeretlen pozíció (x,y) koordinátájának meghatározására most 4 mobiltoronyra végeztünk távolság méréseket, ebből kellene meghatározni a legvalószínűbb pozíciót!



Mobil torony sorszáma	X koordináta (x_i) [m]	Y koordináta (y_i) [m]	Mért bázis-terminál távolság (r_i) [m]
1	561	487	2130
2	5203	4625	5620
3	5067	-5728	6040
4	1012	5451	5820

A mért távolságok egy-egy kört határoznak meg, aminek az egyenlete implicit alakban a következő:

$$(x - x_i)^2 + (y - y_i)^2 - r_i^2 = 0,$$

ahol x_i, y_i a mobiltornyok koordinátái, x,y pedig a keresett álláspont.

Első lépésként ábrázoljuk a köröket és nézzük meg a metszéspont környékét Matlab-ban! Először adjuk meg vektorokban a mérések eredményeit és ábrázoljuk a mobiltornyok helyzetét!

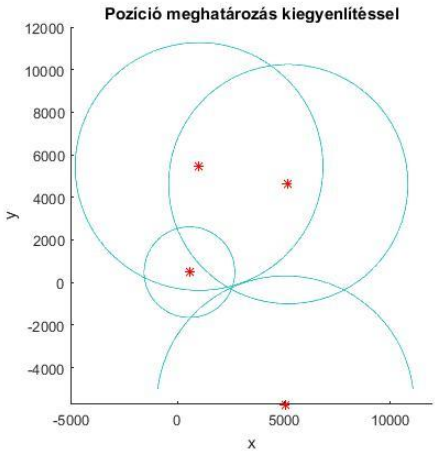
```
clear all; clc; close all;
xt = [561; 5203; 5067; 1012]
yt = [487; 4625; -5728; 5451]
rm = [2130; 5620; 6040; 5820]
% Körök középpontjai
figure(1); hold on;
plot(xt, yt, 'r*')
```

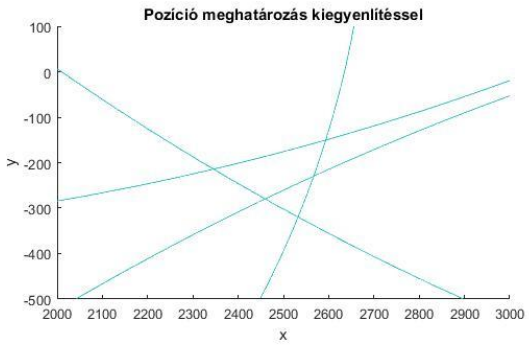
Ezután definiáljuk a kör egyenletét általánosan, és szimbolikus x,y változót használva ábrázoljuk az egyes köröket!

```
% Kör általános egyenlete
kor = @(x,y,xi,yi,ri) (x-xi).^2 + (y-yi).^2 - ri.^2
% körök ábrázolása
syms x y
E = kor(x,y,xt,yt,rm)
for i = 1:4
    fimplicit(E(i),[-5000 12000])
end
axis equal
title('Pozíció meghatározás kiegyenlítéssel')
```

Nagyítsunk rá a metszéspont körüli területre!

```
% A metszéspont körüli terület
figure(2); hold on;
for i = 1:4
    fimplicit(E(i),[2000 3000 -500 100])
end
axis equal
title('Pozíció meghatározás kiegyenlítéssel')
```



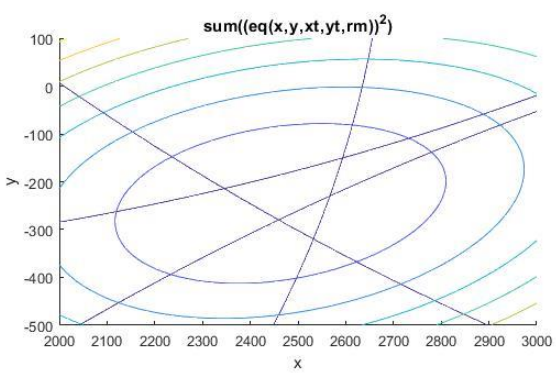


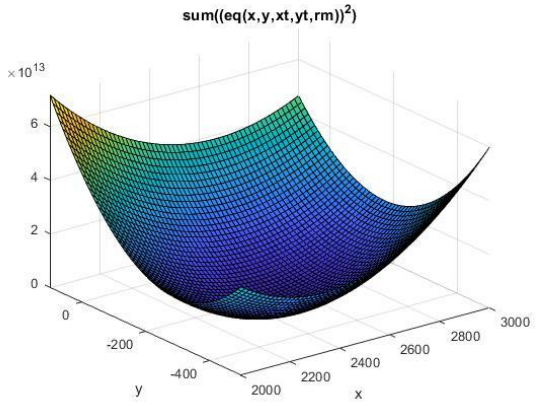
Látjuk, hogy a négy kör nem egy pontban metszi egymást, hanem közrezárnak egy területet, ahol a feltételezett pozíciónk van. Ezt a legvalószínűbb helyzetet határozhatjuk meg a maradék eltérések négyzetösszegének minimalizálásával. Írjuk fel a minimalizálandó függvényt (*f*), ami a maradék eltérések négyzetösszege lesz

$$f(x,y) = \sum_{i=1}^{n}((x-x_i)^2 + (y-y_i)^2 - {r_i}^2)^2$$

Definiáljuk a célfüggvényt és ábrázoljuk is Matlab-ban szintvonalakkal és 3D felülettel is!

```
> % minimalizálandó függvény
> f = sum((kor(x,y,xt,yt,rm)).^2)
> F = matlabFunction(f) % f szimbolikus kifejezés függvénnyé alakítása
> fcontour(f,[2000 3000 -500 100]);
> figure(3)
> fsurf(f, [2000 3000 -500 100])
```





A fenti függvény minimum helye lesz a keresett pozíció legvalószínűbb értéke. Hogyan tudjuk ezt megtalálni? Használhatjuk például a többváltozós Newton-módszert!

TÖBBVÁLTOZÓS NEWTON-MÓDSZER

Egyváltozós esetben a Newton-módszer szélsőérték keresésre alkalmazott iterációs képlete a következő volt:

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}$$

Ezt a képletet lehet általánosítani többváltozós esetre is, csak az első derivált helyett a többváltozós függvény gradiens vektorát kell használjuk (∇f), a második derivált helyett pedig a Hesse mátrixot (H). A több változót itt is egy vektorban kel megadni (x). Itt az

$$x_{i+1} = x_i - H^{-1}(x_i) \cdot \nabla f(x_i)$$

ahol a Hesse-mátrix, az f(x) függvény második parciális deriváltjainak a mátrixa, a gradiens vektor pedig az első parciális deriváltak vektora. Kétváltozós f(x,y) esetben a gradiens vektor és a Hesse-mátrix a következő lesz:

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}; \quad H(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x\, \partial y} \\ \frac{\partial^2 f}{\partial y\, \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

Korábban, a nemlineáris egyenletrendszerek megoldásakor, már használtuk a Jacobi mátrixot, amiben a vektorban tárolt egyenleteknek/függvényeknek az első parciális deriváltjai voltak benne. A Hesse mátrix előállítható egy függvény gradiens vektorára kiszámolt Jacobi mátrixszal is.

TÖBBVÁLTOZÓS NEWTON-MÓDSZER MATLAB-BAN

```
function [x i X] = gradmulti(grad, hesse, x0, eps, nmax)

x1 = x0 - pinv(hesse(x0))*grad(x0);
i=1;
X=[x0 x1];

while and(norm(x1 - x0) > eps, i < nmax)
    x0 = x1;
    x1 = x0 - pinv(hesse(x0))*grad(x0);
    i = i + 1;
    X = [X x1];
end
x = x1;
```

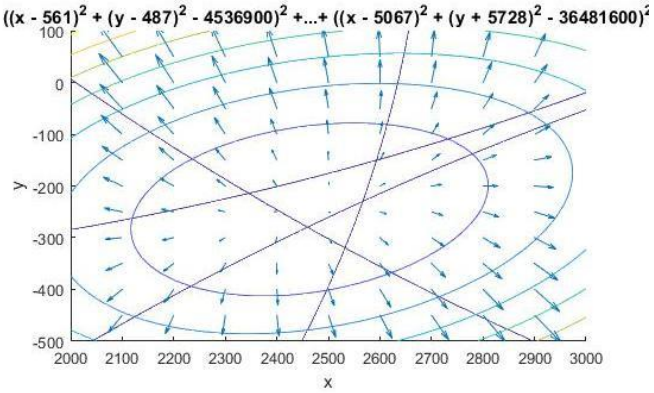
A fenti függvény (gradmulti.m) a többváltozós Newton-módszer megoldása Matlab-ban, ahol a bemenet a gradiens vektor, a Hesse mátrix, egy x0 kezdeti pozíció, eps tolerancia érték a leállási feltételhez és egy nmax maximális iteráció szám.

A kimenetben x1 a megoldás, i az iteráció szám, X pedig az egymást követő megoldások lépéseit tartalmazza.

POZÍCIÓ MEGHATÁROZÁS TÖBBVÁLTOZÓS NEWTON-MÓDSZERREL

Láttuk, hogy többváltozós esetben szükség lesz a gradiens vektorra és a Hesse mátrixra, az egyváltozós esetben alkalmazott első és második derivált helyett. Állítsuk elő ezeket Matlab-ban. A gradiens vektor előállítására használhatjuk a **gradient** parancsot a Matlab-ban, mind numerikusan, mind szimbolikusan. Hogy jobban el tudjuk képzelni ábrázoljuk először a numerikusan előállított gradiens vektorokat! Ehhez hozzunk létre egy rácsot **meshgrid** paranccsal és ebben a rácsban számoljuk ki a gradiens értékeket, amiket a **quiver** paranccsal ábrázolhatunk!

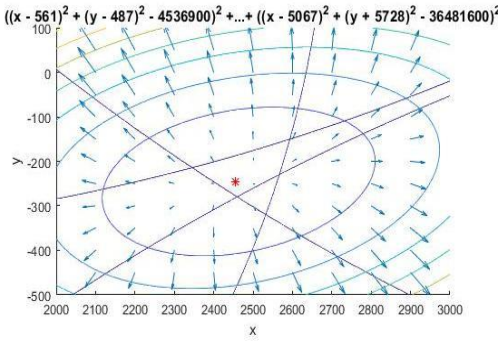
```
% Gradiens vektor ábrázolása numerikusan
[X,Y] = meshgrid(2000:100:3000, -500:50:100);
Z = F(X,Y);
[px,py] = gradient(Z); % gradiensek számítása numerikusan
figure(2)
quiver(X,Y,px,py)
```



A többváltozós Newton-módszer alkalmazásához nem numerikusan van szükségünk a gradiens vektorra és Hesse mátrixra, hanem vektorváltozós függvény formájában. Ezt meghatározhatjuk szimbolikus számításokkal a **gradient** és a **hessian** parancsok alkalmazásával a szimbolikus *f* függvényre!

```
> % Gradiens vektor szimbolikusan
> G = gradient(f)
> % Hesse mátrix szimbolikusan
> H = hessian(f)
> % Alakítsuk H-t és G-t vektorváltozós függvénnyé
> G = matlabFunction(G) % G szimbolikus kifejezés függvénnyé alakítása
> H = matlabFunction(H) % H szimbolikus kifejezés függvénnyé alakítása
> G = @(x) G(x(1),x(2)) % vektorváltozóssá alakítás
> H = @(x) H(x(1),x(2)) % vektorváltozóssá alakítás
```

A megoldáshoz válasszunk kezdőértéket az ábrából és hívjuk meg a gradmulti.m függvényt!




```
> x0 = [2400; -300]
> [p i pp] = gradmulti(G,H,x0,1e-6,100)
> % Ábrázoljuk a megoldást!
> plot(p(1),p(2),'r*')
```

Nagyon gyorsan konvergált a módszer, 4 iterációból eljutottunk a minimumhelyre a kívánt pontosságon belül.

MATLAB BEÉPÍTETT FÜGGVÉNY ALKALMAZÁSA (FMINSEARCH)

Vannak Matlabos beépített függvények is, amiket többváltozós szélsőérték kereséshez használhatunk, az egyik az **fminsearch**, ami a Nelder-Mead szimplex módszert használja, egy másik pedig az **fminunc**, ami kvázi-Newton minimalizálást alkalmaz. Ha a deriváltak nem számíthatóak könnyen, akkor célszerű a szimplex módszert használni. A módszer során felveszünk egy kezdő poliédert (szimplexet), 2 dimenziós esetben egy háromszöget, majd az eljárás során különböző műveleteket alkalmazva (nyújtás, zsugorítás, tükrözés) úgy változtatjuk a 3 pont helyzetét, hogy mindig igazodjon a függvényfelület alakjához, amíg a minimumhely környezetére zsugorodik. Az eljárás megértéséhez érdemes megnézni mintának a következő animációt: https://en.wikipedia.org/wiki/File:Nelder-Mead_Himmelblau.gif

Oldjuk meg a feladatot szimplex módszerrel! Ehhez az *F* függvényt vektor változóssá kell alakítanunk!

```
> x0 = [2400; -300]
> F = @(x) F(x(1),x(2)) % vektorváltozóssá alakítás
> sol = fminsearch(F,x0)
> plot(sol(1),sol(2),'ks')
```

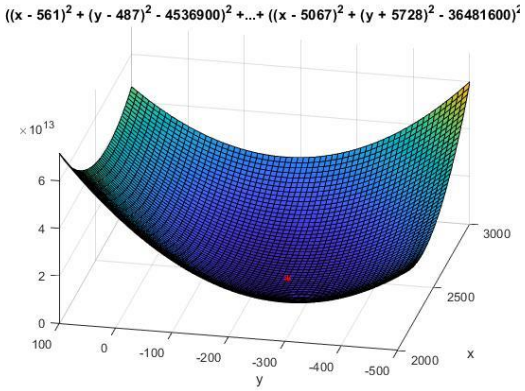
Nézzük meg az eltérést a mért távolságok és kiegyenlített álláspont-mobiltornyok távolságai között!

```
> % hibák
> ex = xt - sol(1); ey = yt - sol(2);
> er = rm - sqrt(ex.^2+ey.^2)
> %    99.0847
> %    26.3188
> %   -31.7595
> %   -57.7440
```

Ábrázoljuk a megoldást a 3D ábrán is!

```
> figure(3); hold on;
```

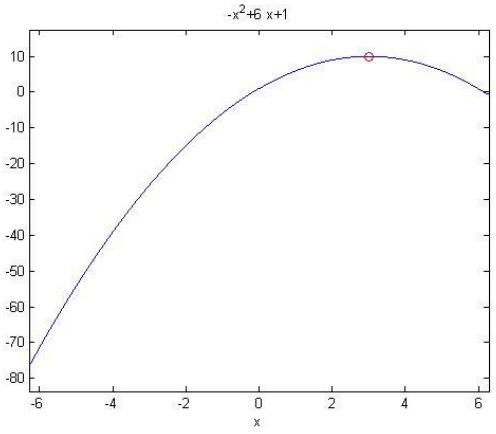
```
> plot3(sol(1),sol(2),F(sol),'r*')
```



Megjegyzés: Az ismertetett módszerek lokális szélsőérték kereső algoritmusok voltak, mindig egy adott kezdőérték közelében keresték a lokális minimumot. A globális minimum meghatározásához több lokális minimum esetében meg kell vizsgálni az összes lokális minimum értékét, és közülük kiválasztani a legkisebbet, az lesz a globális minimum. Léteznek olyan módszerek is, melyekkel rögtön a globális minimumot lehet meghatározni egy adott tartományon (például genetikus algoritmusok), de ezekkel most idő hiányában nem tudunk foglalkozni.

MAXIMUM KERESÉS[17]

Az eddig használt módszerek közül az intervallum és aranymetszés módszerével minimumhelyet tudunk csak megkeresni, a Newton módszerrel vízszintes érintőjű helyeket keresünk, tehát ez mind minimum, mind maximum meghatározásra alkalmazható, a beépített fminsearch pedig, ahogy a neve is mutatja szintén minimum hely meghatározására alkalmazható. Lehetne persze kis módosítással az intervallum/aranymetszés módszert is maximum keresésre használni, de egyszerűbb, ha maximum keresés helyett a függvény (-1) szeresének a minimumát keressük meg. Nézzünk erre egy példát! Keressük meg az $f(x) = -x^2 + 6x + 1$ függvény maximumát!



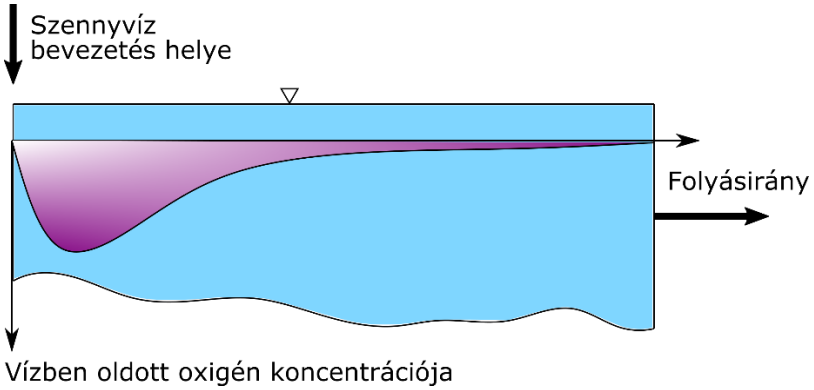

```
> clear all; clc; close all;
> f = @(x) -x.^2 + 6*x + 1
> fplot(f)
> fm = @(x) -f(x)
> xm = fminsearch(fm,4) %  3.0000
> fmax = f(xm) %  10.0000
> hold on; plot(xm, fmax, 'ro')
```

Ne felejtsük el, hogy a maximum értékének megállapításához az eredeti függvénybe kell visszahelyettesíteni!

[17] Otthoni átnézésre

GYAKORLÓ FELADAT[18]

Szennyvízbevezetések által terhelt élővizek esetében, a környezeti hatások megismerése és egyúttal lehetséges minimalizálása érdekében fontos a szennyező anyag terjedésének, koncentráció eloszlásának meghatározása. Folyókba torkoló tisztított szennyvízbevezetés esetében vizsgáljuk meg a folyó vízben oldott oxigén koncentrációjának minimális értékét! Ennek értéke a víz élővilágának védelme érdekében nem lehet kisebb egy kritikus minimum szintnél!



Az ábra a koncentráció változását mutatja a szennyező anyag tartózkodási idejének függvényében. Az oldott oxigén koncentráció (c) változását az idő függvényében a következő függvénnyel lehet leírni [mg/L]:

$$c(t) = c_s - \frac{k_d L_0}{k_d + k_s - k_a}\left(e^{-k_a t} - e^{-(k_d+k_s)t}\right) - \frac{S_b}{k_a}\left(1 - e^{-k_a t}\right)$$

ahol t a tartózkodási idő [nap], c_s a telítési koncentráció (most az értéke: 10 mg/L), L_0 a biokémiai oxigénigény (BOD) a betáplálásnál (50 mg/L), k_d a lebomlási sebesség [0.1 1/nap], k_s a kiülepedési sebesség [0.05 1/nap], k_a az átlevegőzési sebesség [0.6 1/nap], S_b pedig a kiülepedési oxigénigény [1 mg/L/nap].

Határozzuk meg a folyó minimális oxigén koncentrációját a szennyvíz bevezetés közelében! Először ábrázoljuk a függvényt!

```
> % szennyvíz bevezetés
> clear all; clc; close all;
> cs = 10; L0 = 50; kd = 0.1; ks = 0.05; ka = 0.6; Sb = 1;
>
> c = @(t) cs - kd*L0/(kd+ks-ka)*(exp(-ka*t)-exp(-(kd+ks)*t)) -
  Sb/ka*(1-exp(-ka*t))
```

A koncentráció függvénye el lett mentve a koncentracio.mat fájlba, így a begépelések elkerülése miatt betölthetjük a függvényt abból is:

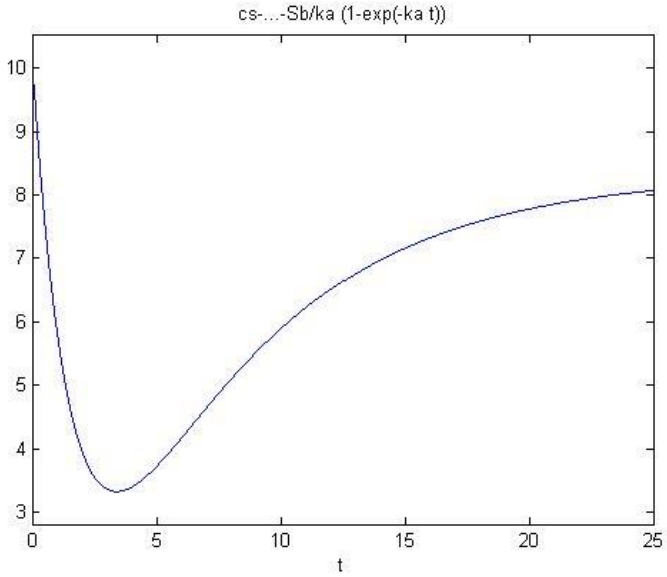
```
> load koncentracio;
> c
> % c = @(t)cs-kd*L0/(kd+ks-ka)*(exp(-ka*t)-exp(-(kd+ks)*t))-Sb/ka*(1-
  exp(-ka*t))
```

Ábrázoljuk 0-25 nap között a koncentráció változását!

```
> figure(1)
```

[18] Otthoni átnézésre

```
> fplot(c,[0 25])
```



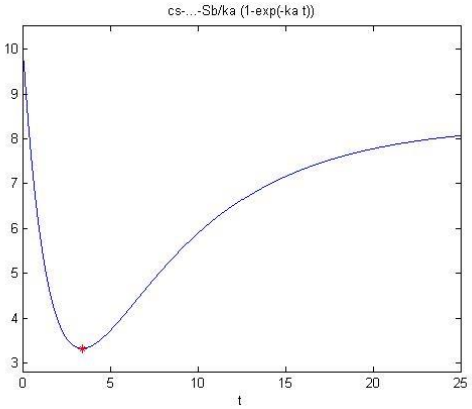
Keressük meg az intervallum módszerrel a folyó minimális oxigén koncentrációját! A kezdő unimodális intervallum legyen a [0, 5] az ábra alapján!

```
> % intervallum módszer - egyenlő felosztás
> [x1 i1] = intervallum(c,0,5,1e-6)
> % x1 =  3.3912; i1 =  37
> c1 = c(x1) % 3.3226
```

Tehát összesen 37 iteráció alatt találtuk meg a minimumhelyet, a minimális oxigén koncentráció pedig 3.32 mg/L lett.

Keressük meg az aranymetszes módszerével is a minimális oxigén koncentrációt! ábrázoljuk is az eredményt!

```
> % aranymetszés módszere
> [x2 i2] = aranymetszes(c,0,5,1e-6)
> % x2 = 3.3912; i2 = 31
> c2 = c(x2) % 3.3226
>
> hold on;
> plot(x2, c(x2), 'r*')
```



Most egyrészt a korábbi 37 iteráció helyett 31 iterációs lépés is elég volt a megoldáshoz, viszont, ha a függvény kiértékelések számát nézzük, akkor még nagyobb a különbség. Az egyenlő felosztás esetén minden iterációban 2 függvényértéket kellett kiszámolni, vagyis 37*2=74 függvénykiértékelés történt, az aranymetszés esetében csak az első iterációban kellett 2 kiértékelést végezni, utána már csak iterációnként egyet, vagyis összesen 32-szer kellett kiszámolni a függvény értékét! Ez bonyolult függvények esetében nagy előnyt jelent az egyenletesen felvett pontokkal szemben.

Alkalmazzuk most a Newton módszert szélsőérték keresésre! A Newton módszerrel történő szélsőérték kereséshez szükség van mind az első, mind a második derivált

számítására! Kezdőértéknek válasszuk most az előző intervallum egyik végpontját, az összehasonlíthatóság végett, mondjuk az 5-öt! (Természetesen az ábra alapján pontosabb kezdőérték is választható lenne, például a 4). A szimbolikus deriváltak függvénnyé alakításához használjuk a **matlabFunction** függvényt!

```
> %% Newton módszer
> syms t
> df = diff(c(t),t) % df = (5*exp(-(3*t)/20))/3 - (23*exp(-(3*t)/5))/3
> ddf = diff(c(t),t,2) % ddf = (23*exp(-(3*t)/5))/5 - exp(-(3*t)/20)/4
>
> % Alakítsuk át a szimbolikus kifejezéseket függvényekké!
> df = matlabFunction(df)
> ddf = matlabFunction(ddf)
>
> % megoldás Newton módszerrel
> [cn in] = newton(df, ddf, 5, 1e-6, 100) % cn = 3.3912; in = 6
```

Most mindössze 6 iterációból eljutottunk a megoldáshoz, látszik, hogy ez a módszer sokkal gyorsabban konvergál, amennyiben konvergál. Próbáljuk meg változtatni a kezdőértéket, adjuk meg az intervallum másik végpontját a 0-t is, majd egy jobb közelítésként a 4-et, és próbáljunk ki egy távolabbi értéket is, mondjuk a 10-et! Mit kapunk eredményül?

A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

subs	-	szimbolikus változóba konkrét értékek behelyettesítése
fminsearch	-	Egy/többváltozós függvény minimának megkeresése Nelder-Mead szimplex módszert alkalmazva
fminunc	-	Feltétel nélküli szélsőérték keresés kvázi-Newton minimalizálást alkalmazva.

14. NUMERIKUS INTEGRÁLÁS

A numerikus integrálás egy közelítő integrálási módszert jelent, melynek alkalmazásával a határozott integrálok közelítő értékét adjuk meg. Integrálásra nagyon sok terület van szükség, legyen szó ívhossz számításról ($L = \int_a^b \sqrt{1 + (f'(x))^2}dx$), terület, térfogat számításról vagy differenciál egyenletek megoldásáról.

NUMERIKUS INTEGRÁLÁS TRAPÉZ SZABÁLYT ALKALMAZVA

Az egyik legismertebb módszer a trapéz-szabály alkalmazása, ahol a két szomszédos pontot összekötő egyenes alatti trapéz területével közelítjük az adott szakaszra az integrált. Ezt használhatjuk diszkrét pontok esetében is és analitikusan megadott függvények esetében is, amikor nem tudjuk meghatározni szimbolikusan az integrált. Ez utóbbi esetben felveszünk n pontot az integrálandó függvény szakaszon, n-1 szakaszra osztva a tartományt, és ezekre a pontokra alkalmazzuk a trapéz szabályt.

Egyetlen intervallum esetén:

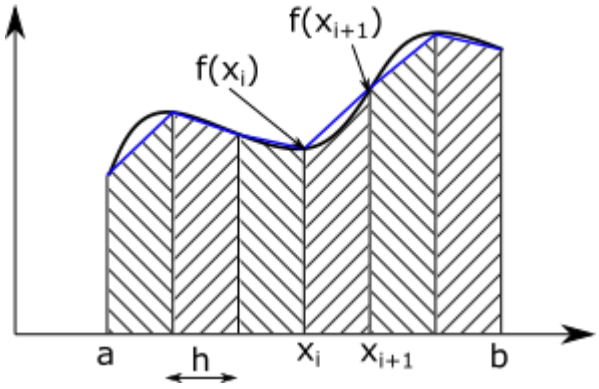
$$\int_a^b f(x)dx \approx \frac{f(a) + f(b)}{2}(b - a)$$

Több intervallumnál összegezni kell a trapézok területét. $N = n - 1$ részintervallum esetén:

$$\int_a^b f(x)dx \approx \frac{1}{2}\sum_{i=1}^{N}\big(f(x_i) + f(x_{i+1})\big)(x_{i+1} - x_i)$$

A fenti képletben nem szükséges, hogy az egyes részintervallumok azonos hosszúságúak legyenek, ha azonban az intervallumok hossza megegyezik $(x_2 - x_1) = (x_3 - x_2) = \cdots = (x_n - x_{n-1}) = h$, akkor egyszerűsíteni lehet a képletet:

$$\int_a^b f(x)dx \approx \frac{h}{2}\sum_{i=1}^{N}\big(f(x_i) + f(x_{i+1})\big)$$



Nézzük meg a fentieket egy példán keresztül!

A Föld sűrűsége (ρ) a sugarával (R) együtt változik, megközelítően az alábbiak szerint:

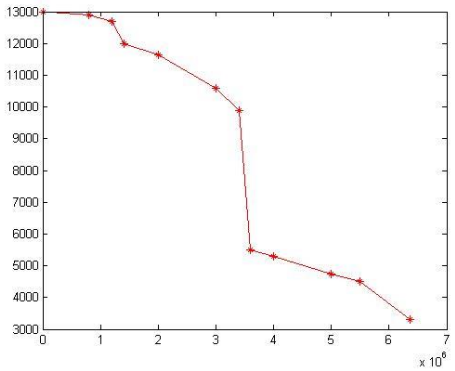
Sugár [km]	0	800	1200	1400	2000	3000	3400	3600	4000	5000	5500	6370
Sűrűség [kg/m^3]	13000	12900	12700	12000	11650	10600	9900	5500	5300	4750	4500	3300

Határozza meg a Föld tömegét az alábbi integrál alapján (a számításoknál figyeljünk a mértékegységekre)!

$$M_{Föld} = \int_0^{6370} \rho 4\pi R^2 dr$$

A sűrűségadatokat beolvashatjuk a **'fold_surusege.txt'** fájlból, vagy beírhatjuk kézzel is.

```
> % Föld tömege
> clc; close all; clear all;
> adat = load('fold_surusege.txt')
> R = adat(:,1)*1000
> ro = adat(:,2)
> figure(1); plot(R,ro,'m*-')
```



Számítsuk ki az integrálandó függvény értékeit a megadott sugár értékekhez!

```
> fx = 4*pi*ro.*R.^2
```

A megoldáshoz használjuk most a Matlab beépített **trapz** függvényét, ami diszkrét pontok alapján közelíti a határozott integrál értékét trapéz szabályt alkalmazva. A pontoknak nem kell egyenletesen elhelyezkedniük.

```
> M = trapz(R,fx) %  6.0261e+24 kg
```

Vagyis a Föld tömegére $6.0261 \cdot 10^{24}$ kilogrammot kaptunk. Ez nagyságrendileg stimmel, a jelenleg elfogadott becslés a Föld tömegére $(5.9722 \pm 0.0006) \cdot 10^{24}$ kg.

NUMERIKUS INTEGRÁLÁS SIMPSON-SZABÁLLYAL

A trapéz szabály egyenesekkel közelíti a szomszédos pontok között a függvényt. Pontosabb eredményt lehet elérni könnyen integrálható, magasabb rendű függvény közelítés alkalmazásával. A legismertebb ilyen módszerek a Simpson-formulák, amelyek másod vagy harmadfokú polinomokkal közelítik a függvény szakaszokat (Simpson's 1/3 method, Simpson's 3/8 method). A másodfokú Simpson-formula esetében 3 szomszédos pontra lehet illeszteni egy parabolát. Ezt megtehetjük a Newton-féle interpolációs polinomot felírva 3 pontra:

$$p(x) = a_1 + a_2(x - x_1) + a_3(x - x_1)(x - x_2)$$

Ahol az együtthatók a következők:

$$a_1 = y_1; a_2 = \frac{y_2 - y_1}{x_2 - x_1};\ a_3 = \frac{\frac{y_3 - y_2}{x_3 - x_2} - \frac{y_2 - y_1}{x_2 - x_1}}{x_3 - x_1}$$

Egyenlő (h) intervallumok esetén:

$$a_1 = y_1; a_2 = \frac{y_2 - y_1}{h};\ a_3 = \frac{y_3 - 2\,y_2 + y_1}{2\,h^2}$$

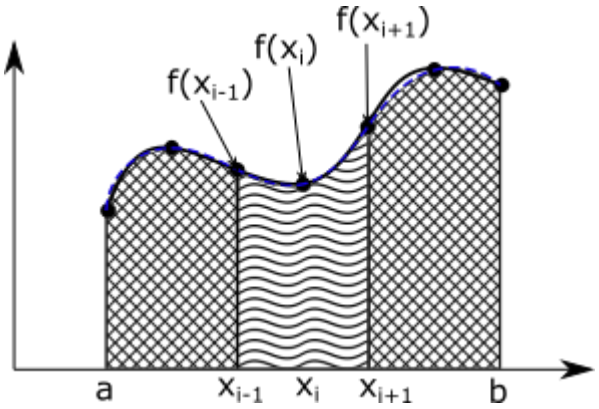
Visszahelyettesítve az együtthatókat a polinomba, levezethető a következő képlet 3 pontra:

$$\int_{x_1}^{x_3} f(x)dx \approx \int_{x_1}^{x_3} p(x)dx = \frac{h}{3}\left(f(x_1) + 4\,f(x_2) + f(x_3)\right)$$

Általánosítva:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx \approx \frac{h}{3}\left(f(x_{i-1}) + 4\,f(x_i) + f(x_{i+1})\right)$$

Legyen n pontunk egyenletesen felvéve, egymástól h távolságra az [a,b] intervallumon, ekkor $x_1 = a,\ x_n = b$. Az n pont $N = n - 1$ szakaszra osztja az intervallumot. A Simpson szabályhoz 3 pont szükséges a parabola illesztéshez, mindig két egymást követő szakaszra számolhatjuk csak az integrált, ezért mindig páros számú szakaszra kell felbontanunk a függvényt a módszer alkalmazásához:



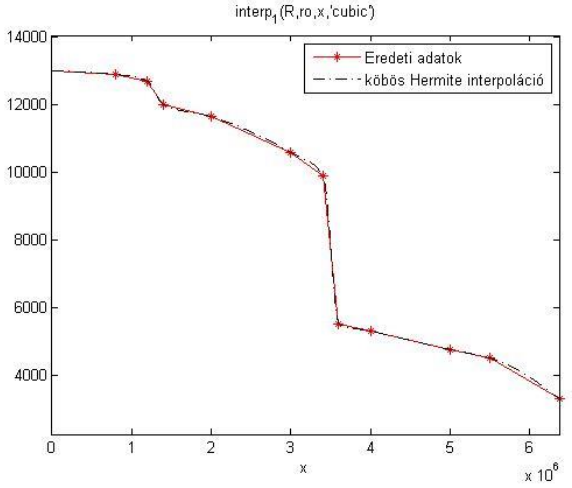
$$\int_a^b f(x)dx = \int_{x_1=a}^{x_3} f(x)dx + \int_{x_3}^{x_5} f(x)dx + \cdots \int_{x_{n-1}=x_N}^{x_n=b} f(x)dx = \sum_{i=2,4,6\ldots}^{N} \int_{x_{i-1}}^{x_{i+1}} f(x)dx$$

A három pontra kapott egyenletet felhasználva, n pontra a következő képletet kapjuk:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4\sum_{i=2,4,6\ldots}^{n-1} f(x_i) + 2\sum_{j=3,5,7\ldots}^{n-2} f(x_j) + f(b)\right]$$

Számoljuk ki a Föld tömegét a Simpson szabályt alkalmazva is! Matlab-ban ezt a **quad** paranccsal tehetjük meg. A **quad** parancshoz azonban nem diszkrét pontokat, hanem egy függvényt kell megadnunk. Ehhez illesszünk a sűrűség adatokra egy spline görbét! Ugyanezt a példát néztük az interpoláció témakörénél is, ott láttuk, hogy a görbében lévő jelentős törések miatt a köbös másodrendű spline illesztés (**spline** parancs) nagy oszcillációt eredményez, ezért most használjunk a köbös elsőrendű interpolációt (**interp1** parancs **'pchip'** módszere).

```
> % köbös Hermite interpoláció
> ro_cubic = @(x) interp1(R,ro,x,'pchip')
> figure(1); hold on;
> g = fplot(ro_cubic, [0 6370000]);
> set(g,'Color', 'k','Linestyle','-.','LineWidth',1);
> legend('Eredeti adatok','köbös Hermite interpoláció')
```



Végezzük el az integrálást a Matlab Simpson-szabályt alkalmazó **quad** parancsával is! Ehhez adjuk meg az integrálandó mennyiséget a sugár függvényeként, felhasználva az előbb illesztett görbét! Majd számítsuk ki az integrált!

```
> fx_cubic = @(R) 4*pi*ro_cubic(R).*R.^2
> M2 = quad(fx_cubic,0,6370000) % 5.9658e+24 kg
```

A Föld tömegére most $5.9658 \cdot 10^{24}$ kilogrammot kaptunk. Ez jóval közelebb áll az elfogadott becsléshez, mint a trapéz szabállyal kapott érték.

Megjegyzés: A **quad** parancs használatát már nem javasolják a Matlab-on belül, későbbi verziókban már nem is lesz benne, hanem helyette az **integral** nevű parancs használatát ajánlják, ami komplexebb esetekben is jól működik. Ez már nem a hagyományosnak tekinthető Simpson szabályt, hanem adaptív kvadratúrát alkalmaz az integrál kiszámítására. A meghívása megegyezik a **quad** parancséval. Most ebben az esetben mégis a Simpson-módszerrel kiszámolt integrál értéke áll közelebb a ténylegeshez.

```
> M3 = integral(fx_cubic,0,6370000) % 6.0541e+24
```

Kiegészítés: Mind a numerikus deriválás, mind a numerikus integrálás pontosítására használható a Richardson-féle extrapoláció. Ezzel a módszerrel a csonkítási hibát tudjuk csökkenteni, úgy, hogy két pontatlanabb közelítést kombinálva egy nagyságrenddel pontosabb közelítést állíthatunk elő. Ennek a részleteivel most idő hiányában nem foglalkozunk.

TÖBBDIMENZIÓS INTEGRÁLOK SZÁMÍTÁSA SZABÁLYOS TARTOMÁNYON

Két és három dimenziós integrálok számítása is egy gyakran felmerülő feladat, ilyen például a terület, térfogat számítás is. Egy két dimenziós határozott integrál az alábbi alakba írható:

$$I = \int_{y_1}^{y_2} \int_{x_1}^{x_2} f(x,y)\, dx\, dy$$

Az integrálás ilyenkor két lépésre bontható, egy belső és egy külső integrálásra. Felírhatjuk először a külső integrált valamelyik korábbi módszerrel (trapéz, Simpson szabály), ahol minden egyes tag egy belső integrált fog tartalmazni, amiket szintén numerikusan számolhatunk. Így az egyváltozós numerikus integrálást lehet általánosítani több dimenzióra, szabályos (téglalap, téglatest) tartomány esetén.

Matlab-ban szabályos tartomány esetén egy függvény kettős integráljának kiszámítására használhatjuk az **integral2** parancsot, 3 dimenziós esetben pedig az **integral3** parancsot.

```
> q = integral2(fun,xmin,xmax,ymin,ymax)
> q = integral3(fun,xmin,xmax,ymin,ymax,zmin,zmax)
```

Az előbbire néztünk is már egy példát a 2D interpoláció témakörében, egy szabályos rácshálóban megadott terep térfogatának számításánál. Egészen más megközelítést kell alkalmaznunk azonban, ha az integrálási tartomány szabálytalan alakú.

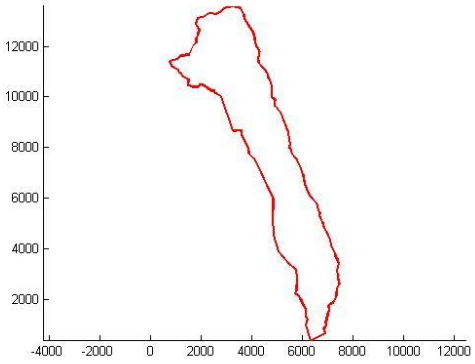
TÖBBDIMENZIÓS INTEGRÁLOK SZÁMÍTÁSA SZABÁLYTALAN TARTOMÁNYON

Szabálytalan tartományon az integráláshoz használhatjuk a **Monte-Carlo módszer**t. Ez egy sztochasztikus algoritmus, ami véletlen számokat használ. A hagyományos integrálási módszerek általában egy szabályos rácson értékelik ki az integrandust, míg ebben az esetben véletlen pontokban történik a függvény kiértékelés. Ezt a módszert a legkönnyebben terület, térfogat számításon lehet bemutatni, de a módszer általánosítható más esetekre is.

TERÜLET SZÁMÍTÁS MONTE-CARLO MÓDSZERREL

Meghatároztuk egy vízgyűjtő terület határának a pontjait, számítsuk ki a vízgyűjtő területét! Ehhez először töltsük be a **'vizgyujto.txt'** állományt, jelenítsük meg, a torzulások elkerülése végett azonos léptékkel az x és y tengelyen a határpontokat!

```
> clc; clear all; close all;
> adat = load('vizgyujto.txt');
> x = adat(:,1); y = adat(:,2);
> figure(1); hold on;
> plot(x,y,'r-','LineWidth',2)
> axis equal
```



Ha a területet Monte-Carlo módszerrel szeretnénk meghatározni, akkor az az alapelv, hogy meghatározzuk a terület befoglaló téglalapját és generálunk ezen a tartományon *N* véletlen pontot egyenletes eloszlásban. Ezután megszámoljuk, hogy hány pont esik az adott területre (*n*) és meghatározzuk azt az arányszámot (ρ), hogy a belül lévő pontok hogy viszonyulnak az összes ponthoz. Kellően sok pont felvétele esetén ez az arány közelítőleg a keresett terület és a befoglaló terület arányát adja:

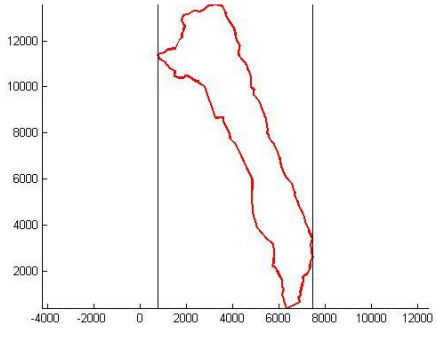
$$\rho = \frac{n}{N}$$

Ha ismerjük a befoglaló téglalap területét: $T = a \cdot b$, akkor a keresett szabálytalan tartomány (jelen esetben vízgyűjtő) területe T_v számítható:

$$T_v = \int_T f(x)\, dT \approx \rho \cdot T = \frac{n}{N} \cdot (a \cdot b)$$

Oldjuk meg ezt Matlab-ban! Először rajzoljuk meg a befoglaló téglalapot a területünk köré (**rectangle**)!

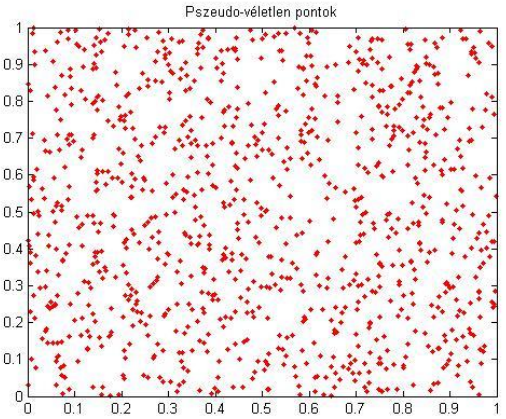
```
> a = max(x)-min(x) % 6.6698e+03
> b = max(y)-min(y) % 1.3169e+04
> rectangle('Position',[min(x),min(y),a,b])
```

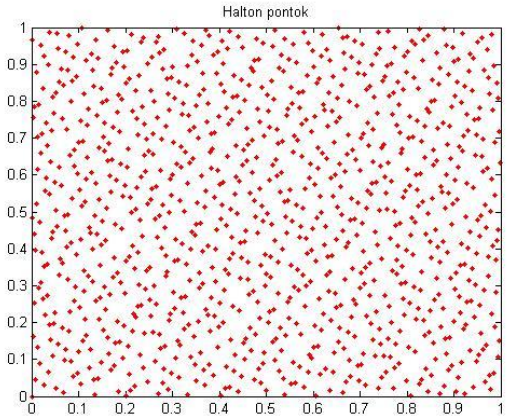


Generáljunk véletlen pontokat két dimenzióban! Ehhez több parancsot is használhatunk, az egyik a pszeudo véletlen számokat létrehozó **rand** parancs, a másik a Halton pontok (**haltonset**), amelyek a *van der Corput* sorozatokon alapulnak. Generáljunk velük 1000 pontot. Mindkettő a [0,1] tartományon dolgozik.

```
> % Pszeudo-véletlen pontok előállítása
> xyr = rand(1000,2);
```

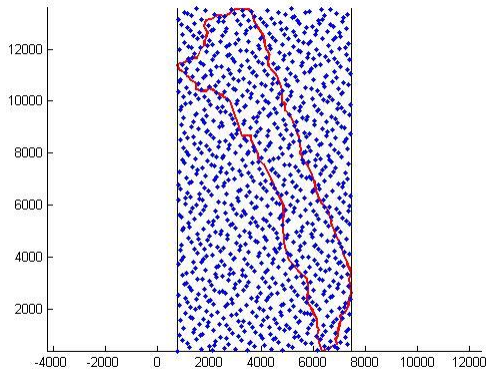
```
figure(2);
plot(xyr(:,1),xyr(:,2),'r.')
title('Pszeudo-véletlen pontok')

% Halton pontok előállítása
hs = haltonset(2); % 2 dimenziós Halton sorozat generálása
xyh = net(hs,1000);
figure(3);
plot(xyh(:,1),xyh(:,2),'r.')
title('Halton pontok')
```





A fenti ábrák közül az első a pszeudo véletlen pontokat, a jobb oldali a Halton pontokat mutatja. Látszódik, hogy ez utóbbi egyenletesebb eloszlású, így használjuk ezt a számításainkhoz!

Mivel a [0,1]x[0,1] tartományon vannak a pontjaink, transzformáljuk át őket a meghatározott befoglaló téglalap területére! Ehhez szorozzuk meg az oldalakat a megfelelő téglalap oldalhosszával és toljuk el a kezdőpontba!



```
% Halton pontok áttranszformálása
  a tartományba, ahol dolgozunk
xh = xyh(:,1)*a + min(x);
yh = xyh(:,2)*b + min(y);
figure(1);
plot(xh,yh,'b.')
```

A Monte-Carlo módszer alkalmazásához meg kell számoljuk, hogy hány pont van a tartományon belül. Ehhez a Matlab **inpolygon** parancsát használhatjuk, ami egy vektort ad vissza a belül lévő pontoknál egyesekkel, a többi pont indexénél nullákkal. A nem nulla elemeket megszámolhatjuk az **nnz** paranccsal (number of nonzero matrix elements).

```
% Terület számítás Monte-Carlo módszerrel
k = inpolygon(xh,yh,x,y);
plot(xh(k),yh(k),'r*')
n = nnz(k) % belül lévő pontok száma: 280
N = length(xh) % összes pont száma: 1000
T = a*b % teljes terület: 87824061 m^2
t = n/N*T % belül lévő terület: 2.4591e+07
format long
```

```
> t %  2.459073708000000e+07 = 24590737.08 m^2
```

Ellenőrzésképp kiszámíthatjuk a területet a geodéziában is használt, koordinátákra felírható trapézokra bontás módszerével is ($T = \sum \frac{(y_{i+1}+y_i)(x_{i+1}-x_i)}{2}$) !

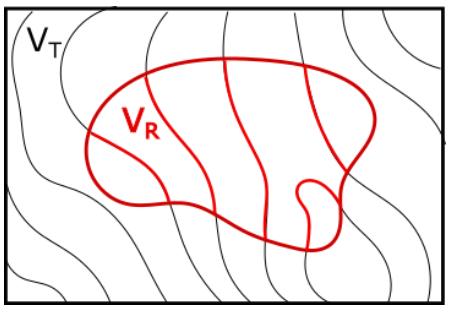
```
> % Ellenőrzés: területszámítás koordináták alapján (trapéz módszer)
> x1 = x([2:end,1]); % eggyel balra tolt koordináták
> y1 = y([2:end,1]); % eggyel balra tolt koordináták
> Tp = sum((y1+y).*(x1-x)/2) %  24591531 m^2
```

Vagy használható ugyanerre a Matlab beépített **trapz** parancsa is, ekkor az utolsó pont koordinátája után oda kell írjuk az első pont koordinátáit:

```
> x1 = [x; x(1)]; y1 = [y; y(1)];
> Tp = trapz(x1,y1) % 24591531
```

A két módszer hasonló eredményt adott, a Monte-Carlo módszer tovább pontosítható több pont felvételével. Egy szabálytalan idom területének kiszámolására sokféle módszer áll rendelkezésünkre, a Monte-Carlo módszer előnye abban nyilvánul meg, hogy általánosítható más esetekre is. Például ezzel a módszerrel kiszámolhatjuk, hogy mennyi csapadék hullott a vízgyűjtő területére, amennyiben ismerjük a csapadék eloszlását, például néhány helyen csapadékmérés történt!

MONTE-CARLO MÓDSZER ÁLTALÁNOSÍTÁSA



Fogalmazzuk meg a Monte-Carlo módszert általánosan! Legyen $f(x)$ értelmezve egy $x \in V_T$ tartományon és keressük a függvény határozott integrálját a tartomány egy V_R résztartományán $V_R \subset V_T$. a keresett integrál: $\int_{V_R} f(x)dV$.

A függvény átlagértékét a keresett tartományon kiszámíthatjuk az alábbi módon integrálással:

$$\bar{f_V} = \frac{1}{V_R}\int_{V_R} f(x)dV$$

Vagy felvehetünk a tartományon *n* pontot és kiszámolhatjuk ezekben a pontokban a függvény értékek átlagát, ami sok pont felvétele esetén közelítőleg megegyezik az integrálással kiszámolt értékkel:

$$\bar{f_V} \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i)$$

ahol $x_i \in V_R$ és n a tartományba eső pontok száma.

Az átlagra kapott kifejezéseket egyenlővé téve kifejezhetjük az integrált:

$$\int_{V_R} f(x)dV \approx \frac{V_R}{n}\sum_{i=1}^{n} f(x_i)$$

A V_R tartomány közelítését megkaphatjuk, a területszámításnál is használt módon. Amennyiben a véletlenszerűen felvett pontok egyenletes eloszlást követnek, akkor

kellően sok pont esetén a tartományon belül lévő pontok száma úgy aránylik az összes ponthoz, mint a V_R rész tartomány nagysága a teljes V_T tartományhoz:

$$\frac{V_R}{V_T} = \frac{n}{N} \rightarrow V_R = \frac{V_T \cdot n}{N}$$

ahol n a tartományba eső, N pedig az összes pont száma. Az integrál közelítése tehát:

$$\int_V f(x)dV \approx \frac{\frac{V_T \cdot n}{N}}{n} \sum_{i=1}^{n} f(x_i) = \frac{V_T}{N} \sum_{i=1}^{n} f(x_i)$$

Vagyis az integrál számítható a tartományon belül lévő pontokban kiszámolt függvényértékek összegének és a (teljes tartomány nagysága/az összes pont) hányadosának szorzatával. A V_T/N tulajdonképpen az egy pontra jutó terület/térfogat nagyságát adja meg.

LEHULLOTT CSAPADÉKMENNYISÉG SZÁMÍTÁS MONTE-CARLO MÓDSZERREL

A megadott vízgyűjtő területen csapadékmérő állomásokat helyeztek el és megmérték a lehullott csapadék mennyiségét egy nagy vihar alatt, az állomásokon 5-12 mm esőt mértek helytől függően. Kérdés, hogy összesen mennyi csapadék hullott a vízgyűjtő terület egészére?

A csapadékmérő állomásokon mért csapadékmennyiségeket közelítették a következő másodfokú polinommal:

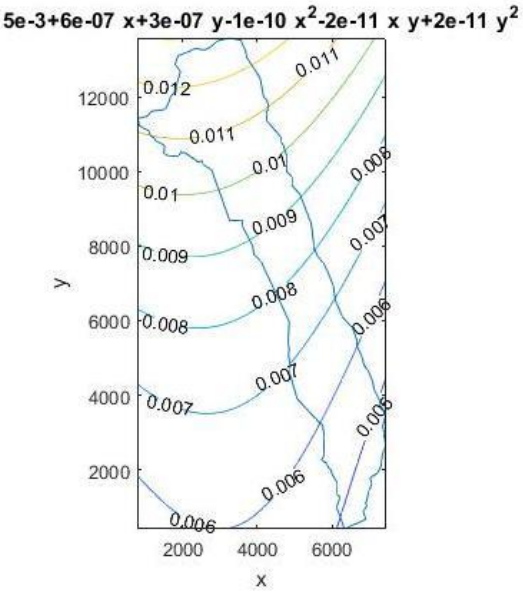
$$f(x,y) = 0.005 + 6 \cdot 10^{-7}\, x + 3 \cdot 10^{-7}\, y - 10^{-10}\, x^2 - 2 \cdot 10^{-11}\, xy + 2 \cdot 10^{-11}\, y^2$$

A fenti függvényt kellene integrálni a vízgyűjtő területére. Oldjuk meg a feladatot Monte-Carlo módszerrel!

Adjuk meg a fenti függvényt! Beírhatjuk kézzel is, vagy betölthetjük a csap.mat állományból!

```
> load csap.mat
> csap % @(x,y)5e-3+6e-07.*x+3e-07.*y-1e-10*x.^2-2e-11.*x.*y+2e-11*y.^2
> figure(4)
> h=ezcontour(csap,[min(x) max(x) min(y) max(y)])
```

```
> set(h,'Show','on'); hold on
> plot(x,y); axis equal;
```



Miután 1000 véletlen pontot már felvettünk a területen, használhatjuk itt is ezeket a pontokat. Az összes csapadékot megkapjuk, ha kiszámoljuk a területen belül lévő pontokban az átlagos csapadék mennyiséget, és ezt utána megszorozzuk a területtel (mivel ezt már az előbb meghatároztuk).

```
> xb = xh(k);
> yb = yh(k);
> n = length(xb) % 280
> N = length(xh) % 1000
> % Az átlagos csapadék a területen
> cs = 1/n*sum(csap(xb,yb)) %
  0.008612820224953 m
> % Az összes lehullott csapadék:
> CS = cs*t % 2.117955976691141e+05
```

Vagy használhatjuk az általánosított Monte-Carlo képletet, ez esetben nem kell kiszámolni a szabálytalan tartomány területét, elég csak a befoglaló téglalap területe és a felvett pontok száma, illetve a tartományon belül lévő pontokban a függvényértékek összege.

```
> % Az általános Monte-Carlo módszert használva,
> % ha a területet nem számoljuk ki külön
> CS2 = T/N*sum(csap(xb,yb)) % 2.117955976691141e+05
```

Amennyiben nem szabálytalan, hanem szabályos területen kell elvégezni az integrálást, ha például a befoglaló téglalapon lehullott csapadék mennyiségére vagyunk kíváncsiak, akkor használhatjuk az **integral2** parancsot. Az **integral2** parancshoz egy függvényt és a 2D tartomány határait kell megadjuk.

```
> % A befoglaló téglalapon leesett csapadék mennyisége
> CS_teglalap = integral2(csap,min(x),max(x),min(y),max(y))
> % 7.197677742886153e+05
```

GYAKORLÓ FELADAT NUMERIKUS DERIVÁLÁSHOZ, INTEGRÁLÁSHOZ[19]

Adott egy q [N/m] fajlagos súlyú szabadvezeték, amelyet két egymástól b távolságra lévő h magasságú oszlophoz rögzítünk úgy, hogy a feszítőerő vízszintes komponense H. A kábel alakját az alábbi koszinusz - hiperbolikus függvény írja le:

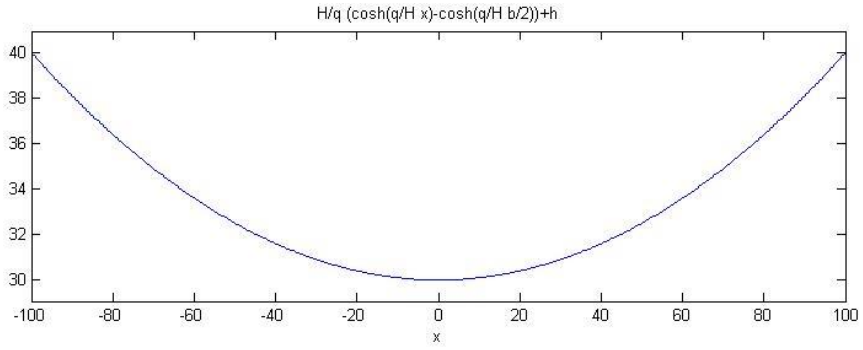
$$f(x,H,q,b,h) = \frac{H}{q}\cdot\left(cosh\left(\frac{q}{H}\cdot x\right) - cosh\left(\frac{q}{H}\cdot\frac{b}{2}\right)\right) + h$$

Határozzuk meg a kábel hosszát, ha

H = 1000 N, q = 2 N/m, b = 200m, h = 40m.

[19] Otthoni átnézésre!

Oldjuk meg a feladatot numerikusan és ellenőrizzük a hibáját szimbolikus számítással!



Egy f(x) függvény ívhosszát [a, b] intervallumon az alábbi módon határozhatjuk meg:

$$s = \int_a^b \sqrt{1 + (f'(x))^2}\,dx$$

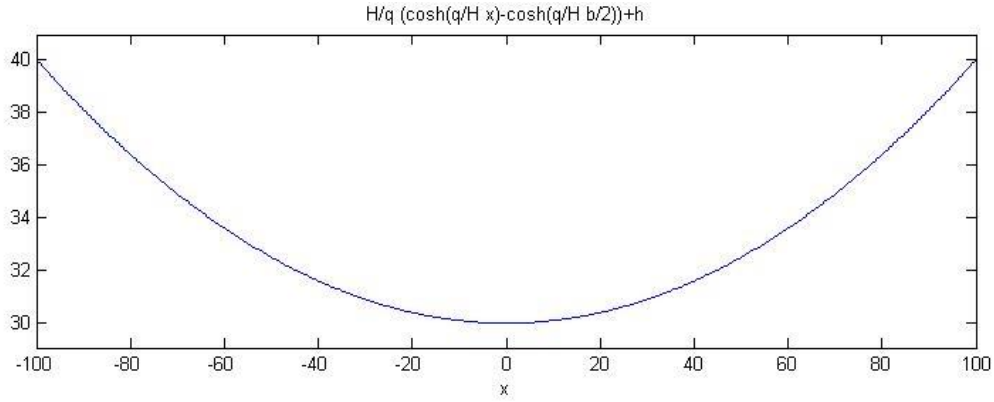
Oldjuk meg a következő feladatokat:

1. Állítsuk elő a pontos megoldást szimbolikusan a későbbi ellenőrzéshez!
2. A deriváltfüggvényt közelítsük a másodrendű hibájú, $O(h^2)$ differencia módszerrel!

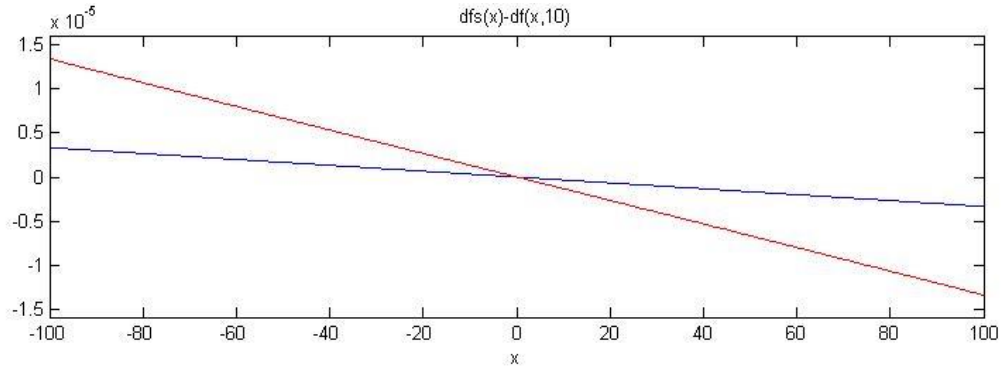
$$dF(x,\Delta) = \frac{F(x+\Delta) - F(x-\Delta)}{2 \cdot \Delta}$$

   Írjuk fel a derivált függvény közelítését Δ lépésköz függvényeként! Hasonlítsuk össze a pontos megoldással grafikusan $\Delta = 5$ és $\Delta = 10$ esetén!
3. Számítsuk ki az ívhosszat a trapézszabállyal $\Delta = 5$ és $\Delta = 10$ esetén!
4. Végezzük el az integrálást a negyedrendű hibájú Simpson szabály alapján is!
5. Ábrázoljuk a hibákat grafikusan oszlopdiagram segítségével!

```
> %% Ívhossz számítás
> % ívhossz: S=int(sqrt(1+df^2),a,b))
>
> clear all; format short; clc; close all;
>
> H=1000; % Feszítőerő vízszintes komponense [N]
> q=2;    % Szabadvezeték fajlagos súlya [N/m]
> b=200;  % Két oszlop közötti távolság [m]
> h=40;   % Oszlop magassága [m]
>
> f=@(x) H/q*(cosh(q/H*x)-cosh(q/H*b/2))+h;
> % A kábel alakját leíró függvény
>
> figure(1); clf;
> fplot(f, [-100,100])
```

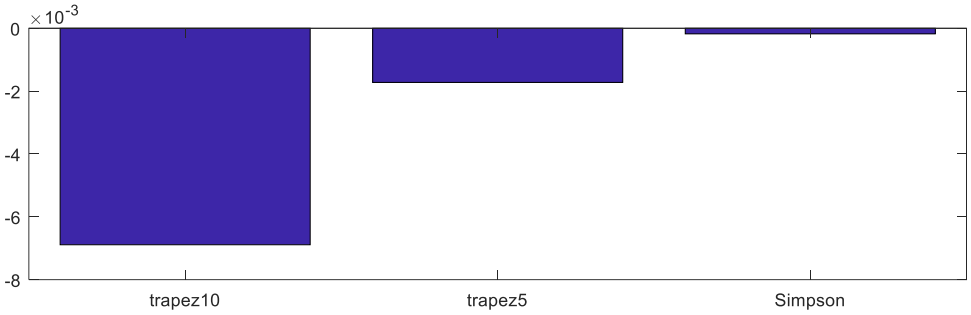



```
>  % A pontos megoldás - szimbolikusan
>  syms x;
>  fs =  H/q*(cosh(q/H*x)-cosh(q/H*b/2))+h
>  diff(fs,x)
>  % ans = sinh(x/500)
>  fsint = sqrt(1+sinh(x/500)^2)
>  ihsym = int(fsint,x,-100,100)
>  % ihsym = 500*exp(1/5) - 500/exp(1/5)
>  ihsym=double(ihsym)
>  % ihsym = 201.3360
>
>  % Numerikus megoldás
>  % A deriváltfüggvényt a másodrendű hibájú O(h^2) differencia
   módszerrel közelítjük
>  df=@(x,d) (f(x+d)-f(x-d))/(2*d); % numerikus közelítés
>  dfs=@(x) sinh(x/500); % a pontos megoldás szimbolikus számításból
>
>  % delta=5 és delta=10 esetén a deriváltak közelítésének hibája
>  d5=@(x) dfs(x)-df(x,5);
>  d10=@(x) dfs(x)-df(x,10);
>
>  % a hibafüggvények ábrázolása
>  figure(2);clf;
>  fplot(d5, [-100,100]);
>  hold on
>  fplot(d10, [-100,100]);
```




```
>  %% Integrálás - ívhossz: S=int(sqrt(1+df^2),a,b))
>  % A pontos érték: 201.3360
>
>  % Integrálás a trapéz szabály alapján (másodrendű hibával) -
   trapz(x,y)
```

```
>  % x,y vektorok (összetartozó értékek)
>  x=-100:10:100; % delta=10
>  y10=sqrt(1+df(x,10).^2);
>  ihtr10=trapz(x,y10)
>  % ihtr10 = 201.3429
>
>  x=-100:5:100; % delta=5
>  y5=sqrt(1+df(x,5).^2);
>  ihtr5=trapz(x,y5)
>  % ihtr5 = 201.3377
>
>  % Integrálás Simpson szabály alapján (negyedrendű hibával) -
   quad(fun,a,b)
>  % fun-függvény, a,b-integrálási határok
>  i10=@(x) sqrt(1+df(x,10).^2); % integrálandó függvény
>  ihSimp=quad(i10,-100,100)
>  % ihSimp = 201.3362
>
>  % Hibák
>  etr10=ihsym-ihtr10 % etr10 = -0.0069, Trapézszabály hibája, delta=10
>  etr5=ihsym-ihtr5   % etr5  = -0.0017, Trapézszabály hibája, delta=5
>  eSimp=ihsym-ihSimp % eSimp = -1.7709e-004, Simpson módszer hibája
>  figure(3)
>  bar([etr10 etr5 eSimp])
>  names = {'trapez10'; 'trapez5'; 'Simpson' };
>  set(gca,'XTickLabel',names)
```



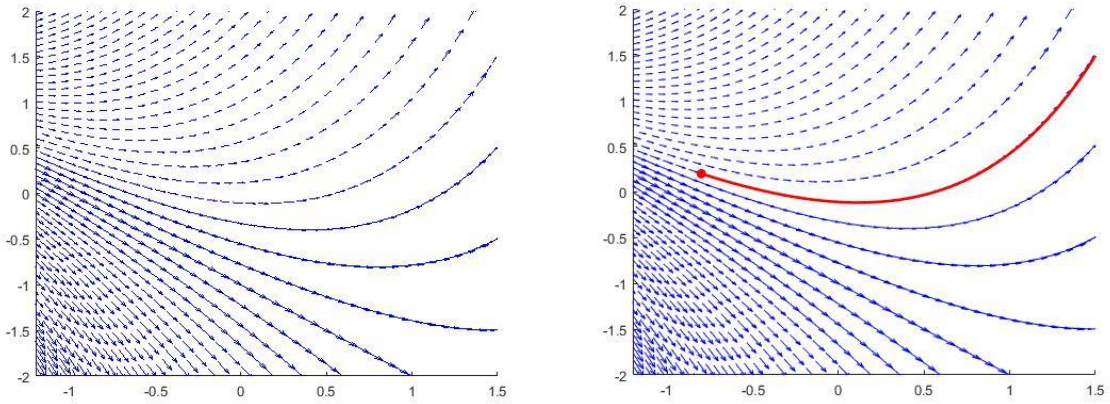
A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

trapz(x,y)	-	Numerikus integrálás diszkrét pontok alapján trapéz szabállyal
quad(fun,a,b)	-	Függvény numerikus integrálása Simpson-szabállyal
integral(fun,a,b)	-	Függvény numerikus integrálása adaptív kvadratúrával
integral2	-	Kettős integrál számítása numerikusan, szabályos téglalap tartományon
integral3	-	Hármas integrál számítása numerikusan, szabályos téglatest tartományon
rectangle	-	Téglalap rajzolás
haltonset(n)	-	n dimenziós Halton sorozat előállítása
net(hset,n)	-	n pont kiválasztása a Halton sorozatból
inpolygon	-	egy zárt poligonon belül lévő pontok meghatározása
nnz	-	nem nulla elemek száma

15. DIFFERENCIÁLEGYENLETEK – KEZDETI ÉRTÉK PROBLÉMA

A differenciálegyenlet egy olyan egyenlet, amely az ismeretlen függvény deriváltjait is tartalmazza. A megoldás itt nem egy konkrét érték lesz, hanem egy olyan függvény, amely kielégíti a differenciálegyenlet rendszert. Közönséges differenciálegyenlet esetén ez egy egyváltozós függvény.

Az egyértelmű megoldás érdekében meg kell adni egy (vagy több) pontot, amelyen a megoldásfüggvény áthalad. Nézzünk egy egyszerű példát, adott a következő differenciálegyenlet: $\frac{dy}{dx} = y + x$, keressük azt a függvényt, amelyik kielégíti ezt a feltételt. A lenti ábra bal oldalán ábrázolt összes függvény kielégíti ezt a feltételt. Ez a differenciálegyenlet trajektória vagy iránymezője (phase portrait). Ha azonban azt a megoldást keressük, ami áthalad az $x = -0.8, y = 0.2$ ponton, akkor már egyetlen függvény lesz a megoldásunk (jobb oldali ábrán pirossal jelölve).



Kezdeti érték probléma esetén ismerjük a függvény és deriváltja(i) értékét a kezdőpontban, ezek alapján próbáljuk meghatározni a függvényt. Peremérték feladatok esetében legalább az egyik érték (a függvény és deriváltjainak értékei közül) nem a kezdőpontban, hanem a végpontban adott. Ez azzal bonyolítja a feladatot, hogy meg kell határoznunk azt a kezdeti értéket is, ahonnan elindulva a végpontban megadott értéket kapjuk.

Lineáris differenciálegyenletben a keresett függvénynek vagy deriváltjának csak a lineáris kifejezése szerepel. Például:

$$e^x \frac{dy}{dx} + a \cdot x^2 + x^4 \cdot y = 0 - \text{lineáris differenciálegyenlet}$$

$$\frac{dy}{dx} + a \cdot x \cdot y + b \cdot y^2 = 0 - \text{nemlineáris differenciálegyenlet}$$

Az egyenlet n-ed rendű, ha abban az ismeretlen függvény legmagasabb deriváltja az n-edik derivált. A megoldásfüggvény meghatározása sokszor - különösen nemlineáris esetben - csak numerikusan lehetséges. Ebben az esetben a függvényt nem analitikusan kapjuk meg, hanem diszkrét pontokban a függvény értékeket, numerikus integrálással. A cél olyan numerikus eljárások alkalmazása, amelyek előírt lokális hiba mellett minél kevesebb lépéssel, pontosabb függvénykiértékeléssel képesek meghatározni a megoldásfüggvény pontjait.

ELSŐRENDŰ KÖZÖNSÉGES DIFFERENCIÁLEGYENLET-KEZDETIÉRTÉK PROBLÉMA

Elsőrendű differenciálegyenlet általános alakja (legyen t a független változó):

$$y' = \frac{dy}{dt} = f(t, y)$$

Egyváltozós esetben egy független változónk van, ez most t, és egy függő változó, ezt most y -nal jelöltük. $f(t, y)$ függvény írja le az első deriváltat. Amennyiben a differenciálegyenlet bal oldalán nem csak az első derivált szerepel, akkor a megoldás előtt át kell rendezni az egyenletet a fenti alakra. Kezdeti érték probléma esetén kezdeti feltételként ismert, hogy a megoldás áthalad a (t_0, y_0) ponton:

$$y(t_0) = y_0$$

EULER-MÓDSZER

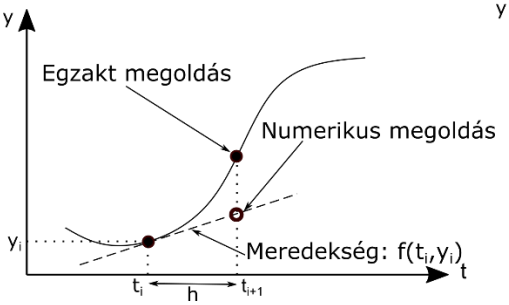
Szeretnénk meghatározni egy általunk felvett intervallumban, adott lépésközönként (h) az eredeti függvény értékeit. Tekintsük állandónak egy adott h szakaszon a függvény meredekségét (m). Ha ismerjük a függvény értékét a szakasz kezdőpontjában és a meredekség értékét, akkor a szakasz végén a függvény értékét közelíthetjük az ismert kezdőponton áthaladó m meredekségű egyenessel.

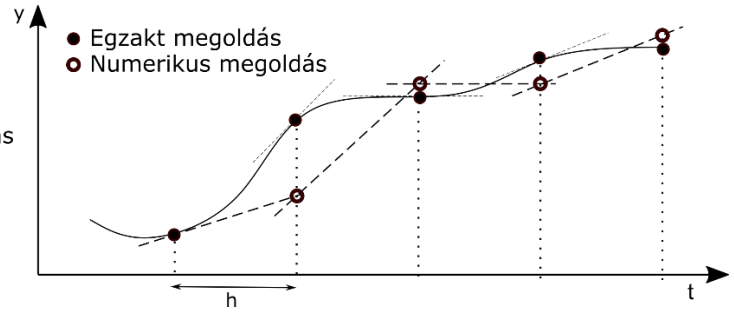
Az Euler-módszer esetén feltételezzük, hogy $m = f(t, y)$ értéke állandó az integrálási részintervallumokban $(h = t_{i+1} - t_i)$ és értéke az intervallum elején kiszámolható értékkel egyezik meg.

$$y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y) \cdot dt \approx y_i + f(t_i, y_i) \cdot h = y_i + m_i \cdot h$$

$$t_{i+1} = t_i + h$$

ahol m a szakasz kezdőpontjában kiszámolt, az adott szakaszon állandónak tekintett meredekség. A módszer lokális hibája $O(h^2)$, globális hibája pedig $O(h)$, azaz a módszer elsőrendű.





Nézzük meg, hogyan oldhatjuk meg az Euler-módszert Matlab-ban (**euler.m**)!

```
function [t,y] = euler (f, y0, a, b, h)
n = round((b - a)/h);
t(1) = a;
y(1) = y0;
for i = 1 : n
    y(i + 1) = y(i) + h*f(t(i), y(i));
    t(i + 1) = t(i) + h;
```

```
> end
```

ELSŐRENDŰ DIFFERENCIÁLEGYENLET MEGOLDÁSA EULER-MÓDSZERREL

Nézzünk egy példát rá! Egy víztorony R=10 m sugarú gömb alakú tartályán alul, h=0 magasságban elhelyezkedő r=5 cm sugarú nyíláson keresztül elkezdik leengedni a benne tárolt (kb. 4000 m^3) vizet. A leengedés kezdetekor ($t = 0$) a vízszint magassága a tartályban 17.44 m. A nyílás kifolyási tényezője $\mu = 0.85$.

a) Mekkora lesz a vízszint a tartályban 12 óra múlva?
b) Mennyi idő alatt ürül ki a tartály?

A víztorony pillanatnyi vízszintjét (a tartály aljától mérve) a következő elsőrendű közönséges differenciálegyenlettel írhatjuk le:

$$f(t,h) = \frac{dh}{dt} = -\frac{\mu\, r^2 \sqrt{2\, g\, h}}{2\, h\, R - h^2}$$

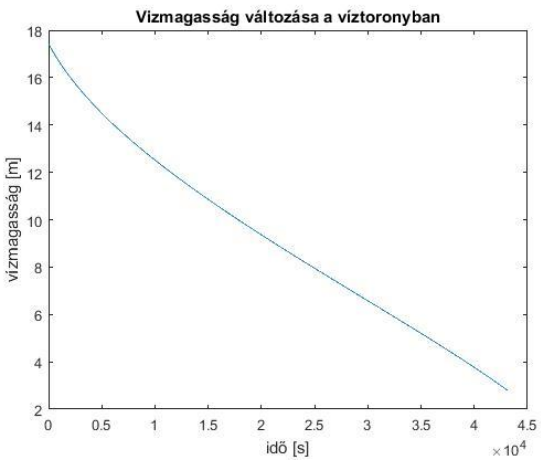
ahol $R = 10\, m, r = 0.05\, m, g = 9.81 \frac{m}{s^2}, \mu = 0.85.$

Oldjuk meg az a) feladatot, hogy mekkora lesz a vízszint 12 óra múlva!

Egy elsőrendű differenciálegyenletnél a megoldás első lépése mindig az, hogy kifejezzük az első deriváltat a többi változó függvényében, ha eredetileg nem így volt megadva. Ez lesz az *f* függvényünk.

Írjuk be a feladatot Matlab-ba és oldjuk meg Euler módszert használva, 60 másodperces lépésközzel! Előfordulhat, hogy az első derivált *f* függvényében nem szerepel a független változó (ami most *t*), de a megoldáshoz a Matlab-ban a differenciálegyenlet függvényének megadásakor az ismeretlenek között mindig meg kell adni a független változót is. Ez a helyzet most is, *t* csak a változók felsorolásánál szerepel, a függvényben nem.

```
> R = 10; r = 0.05; g = 9.81; mu = 0.85;
> % Derivált függvény megadása
> f = @(t,h) -mu*r^2*sqrt(2*g*h)/(2*h*R-h.^2)
> %dhdt = @(t,h) -sqrt(2*g*h)/(alfa*h*(2*R-h));
> % Kezdeti feltétel, tartomány, lépésköz megadása
> h0 = 17.44; t0 = 0; tv = 12*3600 %
  12 óra = 43200 s
> % lépésköz 60 s
> d = 60;
>
> % megoldás Euler-módszerrel
> [T, H] = euler(f, h0, t0, tv, d);
> figure(2)
> plot(T,H);
> xlabel('idő [s]')
> ylabel('vízmagasság [m]')
> title('Vizmagasság változása a
  víztoronyban')
```



A H vektor utolsó eleme megadja, hogy mennyi volt a vízszint 12 óra elteltével:

```
> H(end) % 2.7712 m
```

Az Euler módszer elsőrendű, azaz O(*h*) hibájú módszer. Nézzük meg, tudunk-e ennél pontosabb módszert használni!

EULER MÓDSZER JAVÍTÁSAI (HEUN-, KÖZÉPPONTI-, RUNGE-KUTTA-MÓDSZER)

Hasonlóképp becsüli a függvény értékeket az Euler, a Heun, a Középponti és a Runge-Kutta módszer is, a különbség csupán a meredekség kiszámításának módjában van. Euler módszernél a lépésköz elején számoljuk ki a derivált értékét és ezt használjuk meredekségnek (lásd a fenti ábrák).

A **Heun módszernél** a meredekség az intervallum elején (m_i) és végén (m_{i+1}) számolt meredekségek átlaga. Ahhoz azonban, hogy a meredekséget az intervallum végén ki tudjuk számolni, ismerni kell az ottani függvény értéket is, mivel $m_{i+1} = f(t_{i+1}, y_{i+1})$. Ezért először egy ún. prediktor lépésként Euler módszerrel számítják a végpontbeli közelítő függvény értéket és ezt használják a meredekség meghatározásához. A két meredekség átlagát használva számítható a tényleges függvényérték a végpontban.

1) Prediktor lépés (Euler módszer): $y_{i+1}^{(0)} = y_i + m_i \cdot h = y_i + f(t_i, y_i) \cdot h$,
2) Korrektor lépés: $t_{i+1} = t_i + h, \qquad m_{i+1} = f(t_{i+1}, y_{i+1}^{(0)})$

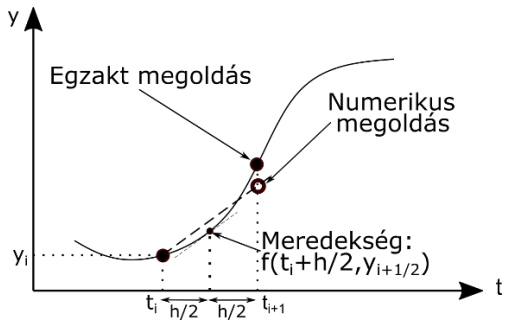
$$y_{i+1} = y_i + \frac{(m_i + m_{i+1})}{2} \cdot h = y_i + \frac{f(t_i, y_i) + f(t_{i+1}, y_{i+1}^{(0)}) \cdot)}{2} \cdot h$$

A módszer lokális hibája $O(h^3)$ és globális hibája $O(h^2)$ azaz a módszer másodrendű hibájú, egy nagyságrenddel pontosabb, mint az Euler-módszer.

A **középponti módszer** esetén a felezőpontban számoljuk ki a deriváltat, és ez lesz az állandónak tekintett meredekség az egész intervallumra. Ehhez először ki kell számolni az előzetes függvényértéket a felezőpontban Euler módszerrel és utána tudjuk számolni ebben a pontban a meredekséget.

1) Felezőpont függvényértéke (Euler módszer): $y_{i+\frac{1}{2}} = y_i + m_i \cdot \frac{h}{2} = y_i + f(t_i, y_i) \cdot \frac{h}{2}$,
2) Meredekség a felezőpontban: $t_{i+\frac{1}{2}} = t_i + \frac{h}{2}, \; m_{i+\frac{1}{2}} = f\left(t_{i+\frac{1}{2}}, y_{i+\frac{1}{2}}\right)$
   Függvényérték a végpontban: $y_{i+1} = y_i + m_{i+\frac{1}{2}} \cdot h$

A módszer lokális hibája $O(h^3)$ és globális hibája $O(h^2)$, azaz hasonlóan a Heun módszerhez, ez is egy nagyságrenddel pontosabb, mint az Euler-módszer.



Tovább lehet pontosítani az Euler-módszert, ha több pontban számoljuk ki a deriváltat, és ezek súlyozott átlaga lesz az állandónak tekintett meredekség. Ez a legelterjedtebb,

negyedrendű hibájú **Runge-Kutta módszer**, amelynek globális csonkítási hibája: $O(h^4)$. Matlab-ban ezt valósítja meg a beépített **ode45** függvény.

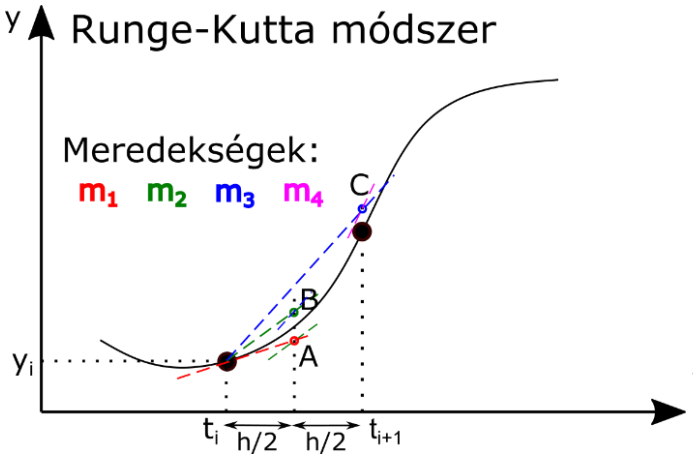
$$y_{i+1} = y_i + \frac{1}{6} \cdot (m_1 + 2m_2 + 2m_3 + m_4) \cdot h$$

$m_1 = f(t_i, y_i)$ - meredekség a kezdőpontban → A pont számítása ezzel

$m_2 = f\left(t_i + \frac{h}{2}, y_i + m_1 \cdot \frac{h}{2}\right)$ - meredekség az A pontban → B pont számítása ezzel

$m_3 = f\left(t_i + \frac{h}{2}, y_i + m_2 \cdot \frac{h}{2}\right)$ - meredekség a B pontban → C pont számítása ezzel

$m_4 = f(t_i + h, y_i + m_3 \cdot h)$ - meredekség a C pontban



$$m_1 = f(t_i, y_i)$$

$$y_A = y_i + m_1 \cdot \frac{h}{2} \rightarrow m_2 = f(t_i + \frac{h}{2}, y_A)$$

$$y_B = y_i + m_2 \cdot \frac{h}{2} \rightarrow m_3 = f(t_i + \frac{h}{2}, y_B)$$

$$y_C = y_i + m_3 \cdot h \rightarrow m_4 = f(t_i + h, y_C)$$

$$y_{i+1} = y_i + \frac{1}{6} \cdot (m_1 + 2m_2 + 2m_3 + m_4) \cdot h$$

ELSŐRENDŰ DIFFERENCIÁLEGYENLET MEGOLDÁSA RUNGE-KUTTA-MÓDSZERREL

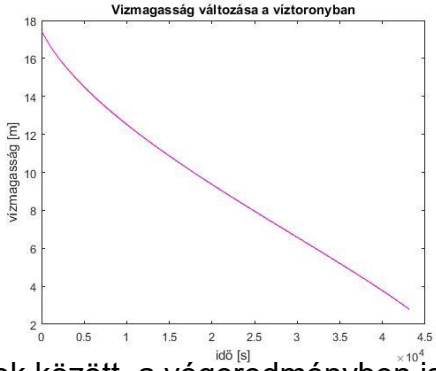
Oldjuk meg az előbbi víztornyos feladatot Runge-Kutta módszerrel is! Ehhez használjuk a Matlab beépített **ode45** parancsát!

Ennek legegyszerűbb hívása a következő:

```
>       [TOUT,YOUT] = ode45(ODEFUN,TSPAN,Y0)
```

ahol ODEFUN egy függvényhivatkozás $y' = f(t, y)$ függvényre, TSPAN lehet a [T0 TV] intervallum megadása, vagy megadott lépésközökkel a kezdő és végpont közötti vektor, Y0 pedig az y függvény kezdeti értéke.

```
> % Megoldás Runge-Kutta-módszerrel
> [T1, H1] = ode45(f, [0,43200], h0);
> H1(end) % 2.7779 m
> % vagy lépésköz megadásával
> [T2, H2] = ode45(f, 0:60:43200, h0);
> H2(end) % 2.7713 m
> hold on;
> plot(T1,H1,'r')
> plot(T2,H2,'m')
```



Ebben az esetben nincs látható különbség a módszerek között, a végeredményben is csak pár mm az eltérés a vízszintben. Érdekes megnézni, hogyan vette fel az algoritmus a lépésközöket, abban az esetben, amikor csak kezdő és végső időpontot adtunk meg:

```
> diff(T1)
```

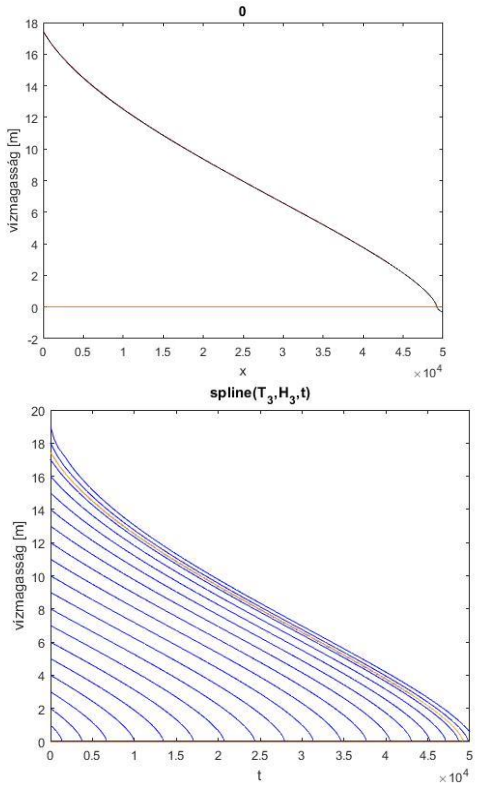
```
> min(diff(T1)) % 995.1333
> max(diff(T1)) % 1.1649e+03
```

Tehát a lépésköz 995 és 1165 másodperc között változott. Sűrűbb lépésköz választása általában pontosítja az eredményt, viszont megnöveli a számítás időszükségletét.

b) Számítsuk ki, hogy a tartály teljes kiürüléséhez hány órára van szükség!

Ehhez először számítsuk ki hosszabb időre a lefolyás alakulását, mondjuk az előbbi 43200 másodperc helyett 50000 másodpercre. Vigyázzunk, mivel negatív *h* esetén komplex számokat kapunk megoldásként! A megoldáshoz illesszünk spline görbét a pontjainkra és zérushely kereséssel határozzuk meg a kiürülés időpontját!

```
> % Lépésköz megadásával
> [T3, H3] = ode45(f, 0:60:50000, h0)
> plot(T3,H3,'k')
> plot([0,50000],[0,0])
> % képzetes rész elhagyása
> H3 = real(H3);
> % spline illesztés
> sp =@(t) spline(T3,H3,t)
> fplot(sp,[0,50000])
> % Hány óra múlva lesz 0 a
  vízmagasság?
> x0 = fzero(sp,49000) % 4.9192e+04 s
> x0 = x0/3600 % 13.6644 h
```



Hogyan alakul a kiürülés különböző kezdeti magasságok esetén? Rajzoljuk fel a trajektória vagy iránymezőt, a maximális 20 méteres kiinduló vízszinttől egészen 1 méteres vízszintig!

```
> % Trajektória vagy iránymező
> for i=20:-1:1
>     [T, H] = ode45(f, [0,50000], i);
>     plot(T,H,'b')
> end
> axis([0 50000 0 20])
```

ELSŐRENDŰ DIFFERENCIÁLEGYENLET RENDSZER MEGOLDÁSA

Sok esetben egy adott folyamat több változótól is függ, amelyek egymást is befolyásolhatják. Ilyen esetekben nem egy differenciálegyenletet, hanem egy differenciálegyenlet rendszert kell megoldanunk. Legyenek a függő változóink $y_1, y_2, \dots, y_n$, a független változónk pedig t . Általános esetben egy elsőrendű differenciálegyenlet rendszer felírása:

$$\frac{dy_1}{dt} = f_1(t, y_1, y_2, y_3 \dots, y_n)$$

$$\frac{dy_2}{dt} = f_2(t, y_1, y_2, y_3 \dots, y_n)$$

...

$$\frac{dy_n}{dt} = f_n(t, y_1, y_2, y_3 \dots, y_n)$$

A kezdeti értékek pedig az [a,b] tartományon:

$$y_1(a) = Y_1,\ y_2(a) = Y_2, \cdots, y_n(a) = Y_n,$$

Ezen egyenletrendszerek egy része megoldható a korábban ismertetett explicit módszerek általánosításával: $t_{i+1} = t_i + h$, Euler módszer esetében például:

$$y_{1,i+1} = y_i + f_1(t, y_1, y_2, y_3 \dots, y_n) \cdot h$$

$$\dots$$

$$y_{n,i+1} = y_i + f_n(t, y_1, y_2, y_3 \dots, y_n) \cdot h$$

Hasonlóképp általánosíthatóak az Euler módszer javításai és a Runge-Kutta módszer is. Nézzünk egy egyszerű kétváltozós példát erre. A megoldást a [0, 1.2] tartományon keressük, *h*=0.4 lépésközönként.

$$\frac{dx}{dt} - x\,t + y = 0; \qquad x(0) = 1$$

$$\frac{dy}{dt} - y\,t - x = 0; \qquad y(0) = 0.5$$

Először rendezzük át az egyenleteket, hogy a baloldalon csak az első deriváltak szerepeljenek:

$$f_1 = \frac{dx}{dt} = x\,t - y;$$

$$f_2 = \frac{dy}{dt} = y\,t + x;$$

Itt két egyenletünk van, f_1 az egyik változó t szerinti első deriváltja, f_2 pedig a másik változó első deriváltja. Oldjuk meg a feladatot a Matlab beépített Runge-Kutta módszerével! A megadott x,y változók helyett vektorváltozót szükséges használni a Matlab beépített függvényeinek a hívásakor, legyen pl.

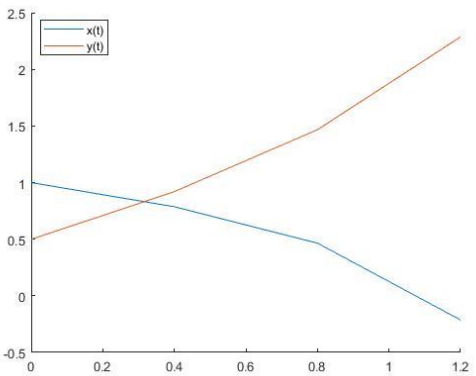
$$v = [x; y], \text{ tehát } v_1 = x, v_2 = y$$

Amennyiben nem túl bonyolult az egyenletrendszerünk, akkor megadhatjuk az egyenletrendszert egysoros függvényként a következőképp:

```
> f1 = @(t,v) v(1)*t-v(2)
> f2 = @(t,v) v(2)*t+v(1)
> F = @(t,v) [f1(t,v); f2(t,v)]
```

A megoldáshoz meg kell adni még a kezdőértékeket, értelmezési tartományt, lépésközt is.

```
> t = 0:0.4:1.2
> x0 = 1; y0 = 0.5; % kezdeti értékek
> [T,V] = ode45(F,t,[x0;y0])
> X = V(:,1); Y = V(:,2);
> figure(1); hold on;
> plot(T,X,T,Y)
> legend('x(t)','y(t)','Location','best')
```



Több változó vagy bonyolultabb összefüggések esetében már célszerű lehet külön fájlban megírni a differenciálegyenlet rendszert. Nézzük meg így is a megoldást. Írjuk meg egy külön **diffrsz.m** fájlba az elsőrendű differenciálegyenlet rendszert!

```
> function F = diffrsz(t,v)
>     f1 = v(1)*t - v(2);
>     f2 = v(2)*t + v(1);
>     F = [f1; f2];
> end
```

Figyeljünk oda, ha külön *.m fájlban adtuk meg a differenciálegyenlet rendszert, akkor a meghívásakor a függvény neve elé kell írni egy @ jelet!

```
> [T, V] = ode45(@diffrsz, t, [x0; y0])
```

MÁSODRENDŰ DIFFERENCIÁLEGYENLETEK

Egy másodrendű közönséges differenciálegyenlet *t* független és *y* függő változóval a következő alakba írható:

$$\frac{d^2y}{dt^2} = f\left(t, y, \frac{dy}{dt}\right)$$

Az egyenlet megoldható [a,b] intervallumon, ha van két ismert feltételünk. Amennyiben a két megadott érték a tartomány elején van, akkor kezdeti érték feladatról beszélünk. A két kezdeti feltétel az y és $\frac{dy}{dt}$ értéke a kezdőpontban. Jelölje ezeket az értékeket A és B.

$$y(a) = A; \qquad \left.\frac{dy}{dt}\right|_{t=a} = B$$

Ez a fajta másodrendű differenciálegyenlet átalakítható két elsőrendű differenciálegyenletből álló egyenletrendszerré, ami az előzőekhez hasonlóan megoldható. A feladat megoldásához az első lépés, hogy kifejezzük a második deriváltat, amennyiben nem ilyen formában van megadva az egyenlet. A második deriváltat $f\left(t, y, \frac{dy}{dt}\right)$ függvényeként írjuk fel. Természetesen nem biztos, hogy ezek mindegyikétől függ. Itt t a független változó, ezt Matlab-ban akkor is meg kell adni, ha esetleg nem függ tőle közvetlenül a derivált függvény. A függő változó és deriváltjai helyett vezessünk be egy új vektorváltozót (w)!

$$w = \begin{pmatrix} y \\ \frac{dy}{dt} \end{pmatrix}$$

Használjuk y és $\frac{dy}{dt}$ helyett w elemeit új változókként: $w_1 = y$ és $w_2 = \frac{dy}{dt}$. Ekkor két egyenletet kell felírnunk a két változó első deriváltjaira, és ezekhez kell megadni a kezdőértékeket:

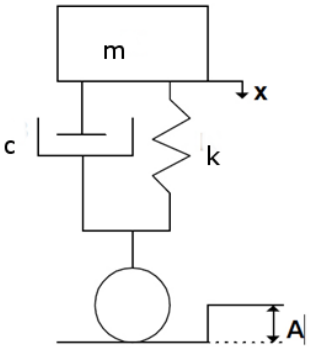
$$f_1 = \frac{dw_1}{dt} = \frac{dy}{dt} = w_2; \qquad w_1(a) = A$$

$$f_2 = \frac{dw_2}{dt} = \frac{d^2y}{dt^2} = f(t, w_1, w_2); \qquad w_2(a) = B$$

Ezekkel a definíciókkal a másodrendű differenciálegyenlet felírható két elsőrendű differenciálegyenletből álló egyenletrendszerként! Oldjunk meg egy ilyen másodrendű differenciálegyenletet!

MÁSODRENDŰ DIFFERENCIÁLEGYENLET MEGOLDÁSA MATLAB-BAN

Egy autó rugózásának szimulációját végezzük az alábbi egyszerű modell alapján, ahol az autó éppen áthalad egy *A* magasságú akadályon. A modellben *m* az autó tömege, *k* a rugómerevség (a rugóban fellépő erő arányos az elmozdulással), *c* a csillapítási tényező (a csillapító erő arányos a tömeg sebességével). Az adatok: $m = 1000\ kg; k = 1000\frac{kg}{s^2}; c = 500\frac{kg}{s};\ A = 0.1\ m.$ A kiinduló időpontban mind az autó függőleges helyzete, mind a függőlege sebessége 0. A vizsgált időintervallum 15 másodperc.



Csillapított szabad rezgésnél a tömegre ható erőket összegezve az alábbi közönséges differenciálegyenletet kapjuk az autó függőleges mozgására:

$$m\,\ddot{x} + c\,\dot{x} + k\,x = 0$$

Ahol x az autó magassági helyzete, $\dot{x}$ az idő szerinti első derivált, tehát az autó függőleges sebessége, $\ddot{x}$ pedig az idő szerinti második derivált, vagyis az autó függőleges gyorsulása. Áttérve az autó koordináta rendszerére a függőleges irányú mozgás mozgásegyenlete:

$$m\,\frac{d^2x}{dt^2} + c\,\frac{dx}{dt} + k\,(x - A) = 0$$

A kezdeti feltételek, hogy a kezdeti függőleges helyzet és a kezdeti függőleges sebesség is nulla, mielőtt az akadályhoz érne az autó:

$$x(0) = 0; \qquad \left.\frac{dx}{dt}\right|_{x=0} = 0$$

Első lépésként fejezzük ki a második deriváltat $\left(\frac{d^2x}{dt^2}\right)$-t az egyenletből!

$$\frac{d^2x}{dt^2} = \frac{1}{m}\left(k\,A - k\,x - c\,\frac{dx}{dt}\right) = f\left(t, x, \frac{dx}{dt}\right)$$

Alakítsuk át a másodrendű differenciálegyenletet elsőrendű differenciálegyenlet rendszerré! Vezessünk be egy új vektor változót a függő változó és deriváltjai helyett:

$$w = \begin{pmatrix} x & \frac{dx}{dt} \end{pmatrix}^T$$

Használjuk a $w_1 = x$ és $w_2 = \frac{dx}{dt}$ új változókat az egyenletünkben! Két egyenletet kell felírnunk, a két változó első deriváltjaira, és ezekhez kell megadni a kezdőértékeket:

$$f_1 = \frac{dw_1}{dt} = \frac{dx}{dt} = w_2; \qquad w_1(0) = 0$$

$$f_2 = \frac{dw_2}{dt} = \frac{d^2x}{dt^2} = \frac{1}{m}(k\,A - k\,w_1 - c\,w_2); \qquad w_2(0) = 0$$

Írjuk meg a differenciálegyenlet rendszert egy külön **autodiff.m** fájlban Matlab-ban! Legyen w egy vektorváltozó: $w = [w_1, w_2]$, tehát $w(1) = x$ a függőleges pozíció és $w(2) = \frac{dx}{dt}$ pedig a függőleges sebesség.

```
> function f = autodiff(t,w)
>    %  A mozgásegyenlet konstansai
>    m=1000; k=1000; A=0.1; c=500;
>    f1 = w(2);
>    f2 = 1/m*(k*A - k*w(1) - c*w(2));
>    f = [f1; f2];
> end
```

Figyeljük meg, hogy a bemenő változók között szerepel a *t* változó is, még akkor is, ha f_1, f_2 kifejezésben közvetlenül nem! Oldjuk meg a feladatot a Matlab beépített, Runge-Kutta módszert használó, **ode45** parancsával, 10^{-4} abszolút és relatív pontossággal, 0-15 másodpercre!

Az **ode45** opcionális paramétereit eddig még nem alkalmaztuk, de lehetőségünk van több érték beállítására az **odeset()** függvényt használva. A fontosabbak:

- RelTol = skalár relatív hibakorlát, amelyik az y minden komponensére érvényes
- AbsTol= skalár vagy vektor abszolút hibakorlát, amelyik a megoldásfüggvényekre egységesen vagy külön-külön érvényes
- MaxStep = maximális megengedett lépésköz
- InitialStep = javasolt kezdő *t* lépésköz

A megoldást készítsük el a **rezgomozgas.m** fájlba (fontos, hogy a megoldást tartalmazó fájl és a differenciálegyenlet rendszert tartalmazó fájl ugyanabban a könyvtárban legyen!):

```
> % Csillapított rezgés
> clc; clear all; close all;
> %  Megoldás Runge-Kutta módszerrel (ode45, odeset)
> options = odeset('RelTol', 1e-4, 'AbsTol',[1e-4 1e-4]);
> % legyen az időintervallum [0, 15] másodperc
> x0=0; % kezdeti pozíció
> v0=0; % kezdeti függőleges sebesség
> [T,W]=ode45(@autodiff,[0,15],[x0; v0],options);
```

A megoldásként kapott W mátrix első oszlopában vannak az elmozdulás értékek ($w(1) = x$) és a második oszlopában az első deriváltak ($w(2) = \frac{dx}{dt}$), vagyis a sebesség értékek.

Mivel nem túl bonyolult egyenletrendszerről van szó a feladat megoldható lett volna egysoros függvény használatával is a következőképp:

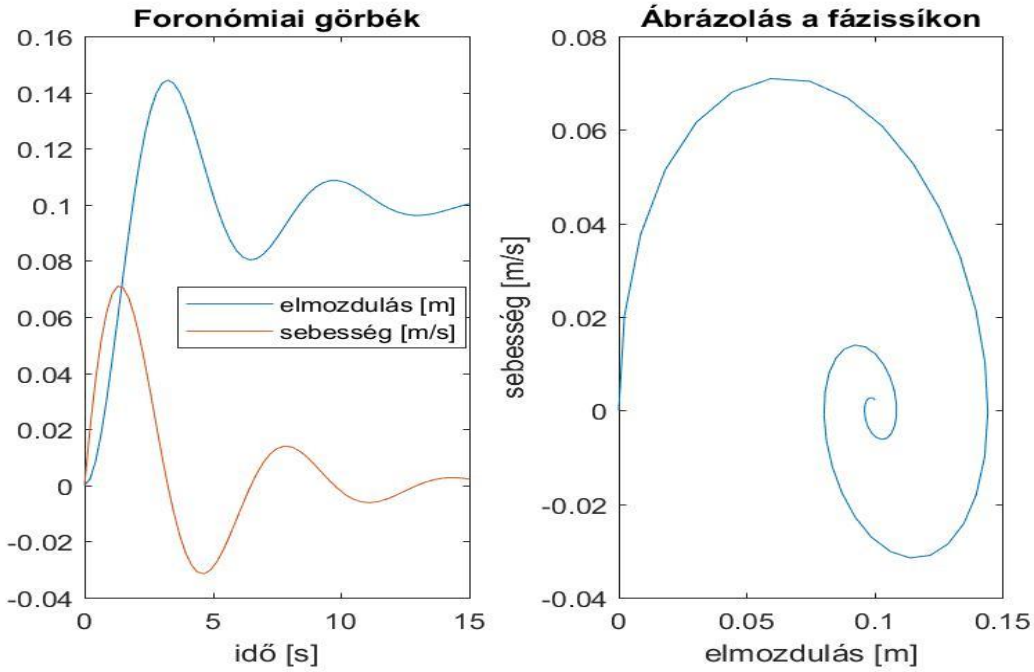
```
> % Más megoldás egysoros függvény használatával
> m=1000; k=1000; A=0.1; c=500;
> dwdt = @(t,w) [w(2); 1/m*(k*A - k*w(1) - c*w(2))]
> options = odeset('RelTol', 1e-4, 'AbsTol',[1e-4 1e-4]);
> x0=0; v0=0;
> [T1,W1]=ode45(dwdt,[0,15],[x0; v0],options);
```

Megjegyzések: Mindkét megoldás egyenértékű, külön fájlban megírva a differenciálegyenlet rendszert szemléletesebb. Figyeljünk arra, hogy a differenciálegyenlet rendszerben nem szerepel külön a *t* paraméter, mégis meg kell

adni a bemenő változóknál a differenciálegyenlet megoldásához! Ugyancsak fontos, ha a differenciálegyenlet rendszert külön fájlban adtuk meg, kell a neve elé írnunk egy @ jelet, ha egysoros függvényként, akkor nem.

Az elmozdulás, sebesség, gyorsulás értékek idő függvényében történő ábrázolását foronómiai görbéknek nevezik, a sebességek ábrázolását az elmozdulás függvényében (ahol az idő a görbe paramétere lesz) pedig fázis síkon történő ábrázolásnak. Rajzoljuk fel két egymás melletti ábrába a foronómiai görbéket és a fázissíkon a sebességeket az elmozdulás függvényében!

```
> % A kocsiszekrény elmozdulása és sebessége az idő függvényében
> subplot(1,2,1)
> x = W(:,1); % függőleges elmozdulás
> v = W(:,2); % függőleges sebesség
> plot(T,x,T,v)
> legend('elmozdulás [m]', 'sebesség [m/s]','Location','Best')
> xlabel('idő [s]')
> title('Foronómiai görbék')
>
> %  Ábrázolás a fázissíkon - az idő most paraméter
> % sebesség az elmozdulás függvényében
> subplot(1,2,2)
> plot(x,v)
> xlabel('elmozdulás [m]')
> ylabel('sebesség [m/s]')
> title('Ábrázolás a fázissíkon')
```



MAGASABB RENDŰ DIFFERENCIÁLEGYENLETEK

Harmad-, negyed- vagy magasabb rendű differenciálegyenleteket szintén vissza lehet vezetni elsőrendű differenciálegyenlet rendszerre, hasonlóan a másodrendű esethez, új változók bevezetésével. Természetesen annyi kezdőértékre lesz szükségünk, ahány egyenletet felírunk, harmadrendű esetben 3, negyedrendű esetben 4 stb.

Egy *n*-ed rendű differenciálegyenlet általánosan:

$$\frac{d^n y}{dt^n} = f\left(t, y, \frac{dy}{dt}, \frac{d^2y}{dt^2}, \dots, \frac{d^{(n-1)}y}{dt^{(n-1)}}\right), \qquad a \le t \le b$$

Kezdeti feltételek:

$$y(a) = A_1;\ \left.\frac{dy}{dt}\right|_{t=a} = A_2;\ \left.\frac{d^2y}{dt^2}\right|_{t=a} = A_3; \dots; \left.\frac{d^{(n-1)}y}{dt^{(n-1)}}\right|_{t=a} = A_n;$$

Vezessünk be egy n elemű új vektorváltozót:

$$w = \left(y \quad \frac{dy}{dt} \quad \frac{d^2y}{dt^2} \quad \dots \quad \frac{d^{n-1}y}{dt^{n-1}}\right)^T$$

Írjuk fel a következő elsőrendű differenciálegyenlet rendszert *w* elemeire a hozzájuk tartozó kezdeti feltételekkel együtt ($y = w_1$):

$$f_1 = \frac{dw_1}{dt} = \frac{dy}{dt} = w_2; \qquad w_1(a) = A_1$$

$$f_2 = \frac{dw_2}{dt} = \frac{d^2y}{dt^2} = w_3; \qquad w_2(a) = A_2$$

$$\dots \qquad \dots$$

$$f_{n-1} = \frac{dw_{n-1}}{dt} = \frac{d^{n-1}y}{dt^{n-1}} = w_n; \qquad w_{n-1}(a) = A_{n-1}$$

$$f_n = \frac{dw_n}{dt} = \frac{d^n y}{dt^n} = f\left(t, y, \frac{dy}{dt}, \frac{d^2y}{dt^2}, \dots, \frac{d^{(n-1)}y}{dt^{(n-1)}}\right); \qquad w_n(a) = A_n$$

Példaként oldjuk meg a következő harmadrendű differenciálegyenletet a [0,1] intervallumon!

$$2x - 3y + 4\,\frac{dy}{dx} + x\,\frac{d^2y}{dx^2} - \frac{d^3y}{dx^3} = 0$$

Ahol a következő kezdeti feltételek adottak:

$$y(0) = 3;\ \left.\frac{dy}{dx}\right|_{x=0} = 2;\ \left.\frac{d^2y}{dx^2}\right|_{x=0} = 7;$$

Első lépés, hogy fejezzük ki a legmagasabb deriváltat!

$$\frac{d^3y}{dx^3} = 2x - 3y + 4\,\frac{dy}{dx} + x\,\frac{d^2y}{dx^2}$$

Alakítsuk át a harmadrendű differenciálegyenletet egy elsőrendű differenciálegyenlet rendszerré, ami 3 egyenletet tartalmaz! A harmadik deriváltat felírhatjuk $f\left(x, y, \frac{dy}{dx}, \frac{d^2y}{dx^2}\right)$ függvényeként. A függő változó és deriváltjai helyett vezessünk be egy új vektorváltozót!

$$w = \begin{pmatrix} y & \frac{dy}{dt} & \frac{d^2y}{dt^2} \end{pmatrix}^T$$

Tehát: $w_1 = y, w_2 = \frac{dy}{dx}, w_3 = \frac{d^2y}{dx^2}$. Az elsőrendű differenciálegyenlet rendszerben az újonnan bevezetett változók első deriváltjait kell megadjuk! 3 változónk van, tehát 3 egyenletet kell felírnunk.

$$f_1 = \frac{dw_1}{dx} = \frac{dy}{dx} = w_2; \qquad w_1(0) = 3$$

$$f_2 = \frac{dw_2}{dx} = \frac{d^2y}{dx^2} = w_3; \qquad w_2(0) = 2$$

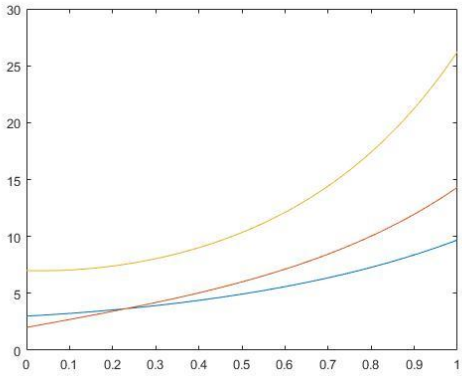
$$f_3 = \frac{dw_3}{dx} = \frac{d^2y}{dx^2} = 2x - 3w_1 + 4\,w_2 + x\,w_3; \quad w_3(0) = 7$$

Matlab-ban ennek a felírása a **diff3.m** fájlban, *w=[w1,w2,w3]:*

```
> function dwdx = diff3(x,w)
>    f1 = w(2);
>    f2 = w(3);
>    f3 = 2*x - 3*w(1) +4*w(2) + x*w(3);
>    dwdx = [f1; f2; f3];
> end
```

Megoldása a [0,1] intervallumon:

```
> w10=3; w20=2; w30=7;
> [X,W]=ode45(@diff3,[0,1],[w10; w20; w30])
> figure(1);
> plot(X,W(:,1),X,W(:,2),X,W(:,3))
```



MAGASABB RENDŰ DIFFERENCIÁLEGYENLET RENDSZEREK

Egy magasabb rendű differenciálegyenlet rendszer hasonlóképp felírható új változók bevezetésével elsőrendű differenciálegyenlet rendszerré. Nézzünk például egy másodrendű differenciálegyenlet rendszert!

$$\frac{d^2x}{dt^2} = F_1\left(t, x, y, \frac{dx}{dt}, \frac{dy}{dt}\right)$$

$$\frac{d^2y}{dt^2} = F_2\left(t, x, y, \frac{dx}{dt}, \frac{dy}{dt}\right)$$

Definiáljunk egy új vektorváltozót a függő változók és deriváltjaik helyett!

$$w = \begin{pmatrix} x & y & \frac{dx}{dt} & \frac{dy}{dt} \end{pmatrix}^T$$

Tehát: $w_1 = x;\ w_2 = y;\ w_3 = \frac{dx}{dt};\ w_4 = \frac{dy}{dt};$ A négy új változónak megfelelő 4 egyenletből álló lineáris egyenletrendszer a következő lesz:

$$f_1 = \frac{dw_1}{dt} = \frac{dx}{dt} = w_3$$

$$f_2 = \frac{dw_2}{dt} = \frac{dy}{dt} = w_4$$

$$f_3 = \frac{dw_3}{dt} = \frac{d^2x}{dt^2} = F_1\left(t, x, y, \frac{dx}{dt}, \frac{dy}{dt}\right)$$

$$f_4 = \frac{dw_4}{dt} = \frac{d^2y}{dt^2} = F_2\left(t, x, y, \frac{dx}{dt}, \frac{dy}{dt}\right)$$

A megoldása az előzőek szerint történhet!

A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

ode45	-	Közönséges differenciálegyenlet rendszer kezdeti érték problémájának megoldása Runge-Kutta módszerrel
odeset	-	Közönséges differenciálegyenlet kezdeti érték feladatát megoldó függvények opcionális paramétereinek megadása (pl. RelTol, AbsTol, MaxStep, InitialStep)

16. DIFFERENCIÁLEGYENLETEK - PEREMÉRTÉK FELADATOK

Eddig a közönséges differenciálegyenletek kezdeti érték feladatával foglalkoztunk, amikor a függvény és deriváltjainak értéke a kérdéses intervallum kezdőpontjában volt megadva. Elsőrendű esetben a függvény értéke a kezdőpontban, másodrendű esetben a függvény és első deriváltjának az értéke a kezdőpontban, harmadrendű esetben a függvény, az első és a második derivált értéke a kezdőpontban stb. Peremérték feladat esetében a függvény és deriváltjainak értékei közül legalább az egyik nem a kezdőpontban, hanem a végpontban adott, és így kell megkeressük azt a függvényt, ami kielégíti a feltételeket.

Nézzünk egy másodrendű közönséges differenciálegyenletet az [a,b] intervallumon:

$$\frac{d^2y}{dt^2} = f\left(t, y, \frac{dy}{dt}\right)$$

A másodrendű differenciálegyenlet megoldásához két feltétel szükséges. Kezdeti érték feladat esetében ez a függvény (y) és az első deriváltjának ($\frac{dy}{dt}$) az értéke a kezdőpontban. Peremérték feladat esetében több változat lehetséges. Lehet adott a függvényérték a kezdő és a végpontban, ezt Dirichlet-peremfeltételnek nevezzük:

$$y(a) = Y_a, \qquad y(b) = Y_b$$

Lehet adott az első derivált értéke a két végpontban, ezt Neumann-peremfeltételnek nevezzük:

$$\left.\frac{dy}{dt}\right|_{t=a} = D_a; \qquad \left.\frac{dy}{dt}\right|_{t=b} = D_b$$

Vagy lehetnek vegyesen pl. a függvény értéke a kezdőpontban és az első derivált értéke a végpontban.

Vizsgáljuk azokat az eseteket általánosan, amikor a probléma visszavezethető elsőrendű explicit differenciálegyenlet rendszerre. Ekkor a megoldandó egyenletrendszer:

$$\frac{d\boldsymbol{y}}{dt} = \boldsymbol{f}(t, \boldsymbol{y})$$

A két peremen ($t = a$ és $t = b$) ismertek a következő értékek:

$$\boldsymbol{y}_i(a) = \boldsymbol{A}_i; \qquad i = 1, \cdots, k$$

$$\boldsymbol{y}_j(b) = \boldsymbol{B}_j; \qquad j = k+1, \cdots, n$$

A megoldás egyik lehetséges módja, hogy visszavezetjük a feladatot egy kezdeti érték probléma ismételt megoldására, ezt nevezzük **tüzérségi módszernek**.

TÜZÉRSÉGI MÓDSZER (SHOOTING METHOD)

A megoldás elve az lesz, hogy megoldjuk a kezdeti érték problémát egy felvett $\boldsymbol{y}_j(a) = \boldsymbol{u}_j$ értékre, majd ellenőrizzük az $\boldsymbol{y}_j(b) = \boldsymbol{B}_j$ feltétel fennállását (a differenciálegyenlet rendszer numerikus megoldásával). Ha eltérés van, akkor módosítjuk az *u* értékét. Tegyük fel, hogy az ismeretlen $\boldsymbol{u}_j$ és az $\boldsymbol{y}_j(b)$ értékek közötti kapcsolatot egy $\boldsymbol{g}(u)$ függvény írja le, vagyis:

$$\boldsymbol{y}_j(b) = \boldsymbol{g}_j(u)$$

Ennek alapján az ismeretlen kezdeti értékeket az alábbi (*n-k*) változós függvények zérus helyeinek megkeresése adja:

$$\boldsymbol{h}_j(\boldsymbol{u}) = \boldsymbol{g}_j(\boldsymbol{u}) - \boldsymbol{y}_j(b) = \boldsymbol{0}$$

TÜZÉRSÉGI MÓDSZER ALKALMAZÁSA

Fellövünk egy petárdát függőlegesen felfelé, a petárda 5 másodperc elteltével robban fel. A petárda függőleges mozgását, ha eltekintünk a légellenállástól, akkor a következő másodrendű differenciálegyenlettel írhatjuk le:

$$\frac{d^2y}{dt^2} = -g$$

Mekkora sebességgel kell fellőnünk a petárdát, ha azt szeretnénk elérni, hogy pontosan 40 méter magasan robbanjon fel?

A kezdeti érték feladatoknál tanult módon alakítsuk át másodrendű differenciálegyenletet $\left(\frac{d^2y}{dt^2} = f\left(t, y, \frac{dy}{dt}\right)\right)$ két elsőrendű differenciálegyenletből álló rendszerré! Ehhez vezessünk be egy kételemű új vektorváltozót, aminek az első eleme a függő változó, vagyis a petárda függőleges helyzete (*y*), a második elem pedig az első derivált, a petárda függőleges sebessége ($\frac{dy}{dt}$):

$$w = \begin{pmatrix} y & \frac{dy}{dt} \end{pmatrix}^T$$

Ekkor $w_1 = y;\ \ w_2 = \frac{dy}{dt}$, az elsőrendű differenciálegyenlet rendszert pedig a vektorváltozó elemeinek deriváltjai adják:

$$f_1 = \frac{dw_1}{dt} = \frac{dy}{dt} = w_2$$

$$f_2 = \frac{dw_2}{dt} = \frac{d^2y}{dt^2} = -g$$

A peremfeltételek:

$$y(0) = 0; \qquad y(5) = 40;$$

Vagyis a függőleges elmozdulás (y) a fellövéskor 0 m, 5 másodperc elteltével pedig 40 m. Kérdés, hogy az első derivált, a sebesség, mekkora legyen kilövéskor, hogy $y(5) = 40$ teljesüljön?

Először oldjuk meg Runge-Kutta módszerrel úgy az egyenletrendszert, hogy felveszünk különböző kezdeti értékeket a sebességre. Legyen $w_2(0) = 20,30,40,50\ ^m/_s$! $\boldsymbol{w}$ egy vektor: $\boldsymbol{w} = [w_1, w_2]$, ahol w_1 a függőleges elmozdulás érték, w_2 pedig az első derivált, vagyis a sebesség értéke.

A differenciálegyenlet rendszert megadhatjuk külön fájlban, pl. a diff_petarda.m fájlban:

```
> function F = diff_petarda(t,w)
>   g = 9.81;
>   f1 = w(2);
>   f2 = -g;
>   F = [f1; f2];
> end
```

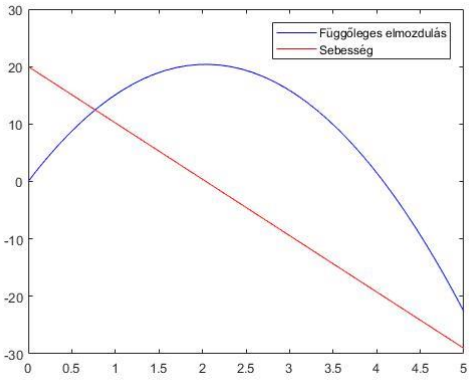
Vagy megadhatjuk egysoros függvényként is abban a fájlban ahová dolgozunk (pl. petarda_felloves.m):

```
> g = 9.81;
> dwdt = @(t,w) [w(2); -g]
```

A Runge-Kutta módszerhez használjuk a Matlab beépített **ode45** parancsát! A differenciál egyenlet rendszert külön fájlban definiálva a függvény neve elé kell tenni egy @ szimbólumot. Először adjunk meg 20 m/s kezdeti sebességet, és $0 \leq t \leq 5\ s$ intervallumot!

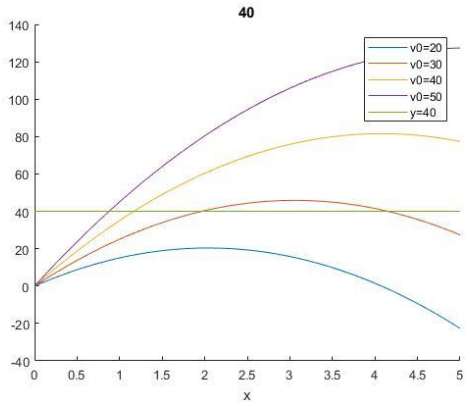
```
> ta = 0; tb = 5; y0 = 0; v0 = 20;
> [T W] = ode45(@diff_petarda,[ta; tb],[y0; v0]); % külön fájlban
> % vagy
> [T W] = ode45(dwdt,[ta; tb],[y0; v0]); % egysoros függvényként
```

Az eredményeknél T vektorban a lépésközök vannak a [0, 5] intervallumon. W egy mátrix két oszloppal és annyi sorral, amennyi elem a T vektorban van. W első oszlopában a függőleges elmozdulás értékek vannak (y), a második oszlopban pedig az első deriváltak $\left(\frac{dy}{dt}\right)$, vagyis a sebességek. Ábrázoljuk az elmozdulásokat és a sebességeket az idő függvényében!




```
> figure(1);
> Y = W(:,1); V = W(:,2);
> plot(T,Y,'b',T,V,'r')
> legend('Függőleges
  elmozdulás','Sebesség')
```

Az ábrából láthatjuk, hogy 20 m/s kezdősebességgel nem is közelítjük meg a 40 m-es magasságot. Vegyünk fel több kezdeti értéket (20,30,40,50 m/s) és most csak a függőleges helyzetet jelenítsük meg! Írassuk ki a végpontbeli magasság értékeket is (W első oszlopának az utolsó eleme)!




```
> figure(2); hold on;
> for vi=20:10:50
```

```
>      [T W] = ode45(dwdt,[ta; tb],[y0; vi]);
>      fprintf('v0=%d m/s; y(5)=%f m\r\n',vi,W(end,1))
>      plot(T,W(:,1));
>  end
>  plot([0,5],[40,40]);
>  legend('v0=20','v0=30','v0=40','v0=50','y=40','40 m')
```

```
v0=20 m/s; y(5)=-22.625000 m
v0=30 m/s; y(5)=27.375000 m
v0=40 m/s; y(5)=77.375000 m
v0=50 m/s; y(5)=127.375000 m
```

Az ábrából és az eredményekből is láthatjuk, hogy az 5 másodperces intervallum végén a 30 m/s kezdősebességgel kevesebb, a 40 m/s kezdősebességgel több mint 40 m-es magasságba jutottunk. Valahol a kettő között lesz a megoldás.

Az ismeretlen kezdeti feltétel, vagyis most a kezdeti sebesség, legyen u, és írjuk fel ennek egy g függvényében a végpontbeli függvény értéket, és vizsgáljuk meg, hogy ez mikor lesz 40!

$$w_1 = g(u) = 40$$

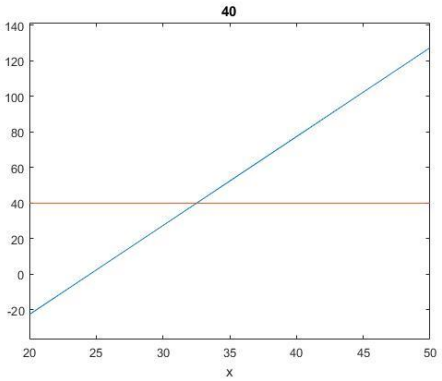
Vagyis keressük az $h = g(u) - 40$ zérushelyét!

Írjuk meg először egy külön fájlban a g függvényt, ami a kezdőérték függvényében visszaadja a végpontbeli magasság értéket! Ehhez a 'diff_petarda.m' külön fájlban megírt differenciálegyenlet rendszert használhatjuk. Legyen ez a fájl a **petarda_magassag.m** fájl!

```
>  function y = petarda_magassag(u)
>   ta = 0; tb = 5; y0 = 0;
>   [T W] = ode45(@diff_petarda,[ta; tb],[y0; u]);
>   y = W(end,1); % az első oszlop utolsó eleme
>  end
```

Ábrázoljuk a kezdősebesség függvényében az 5 másodpercnél elért magasságokat 20 és 50 m/s kezdősebességek között!

```
>  figure(3)
>  fplot(@petarda_magassag,[20, 50]);
>  hold on; plot([20,50],[40,40]);
```

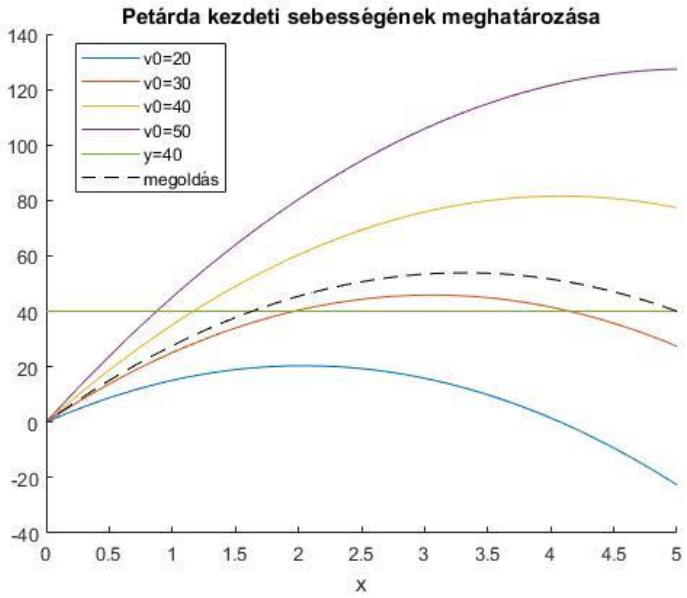


Az ábra alapján valahol 30 és 35 m/s között kell felvennünk a kezdőértéket a sebességhez. Legyen most $u_0 = 32$!

```
>  % a hiányzó kezdőérték meghatározás
>  h = @(u) petarda_magassag(u)-40
>  v0 = fzero(h,32) % 32.5250
```

Rajzoljuk fel a megoldást szaggatott vonallal a többi kezdőérték mellé az ábrába!

```
>  [T W] = ode45(@diff_petarda,[ta; tb],[y0; v0]);
>  figure(2);
>  plot(T,W(:,1),'k--');
>  legend('v0=20','v0=30','v0=40','v0=50','y=40','40 m',
   'megoldás','Location','best')
>  title('Petárda kezdeti sebességének meghatározása')
```

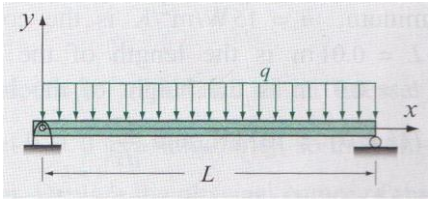


Sok más módszer is létezik peremérték feladatok megoldására, például véges differenciák módszere, próbafüggvények módszere (globális maradék minimalizálása, lokális maradékok eliminálása). Ezek ismertetésére most idő hiányában nem térünk ki.

PEREMÉRTÉK FELADAT MEGOLDÁSA A MATLAB BEÉPÍTETT FÜGGVÉNYÉVEL

Természetesen a Matlab-nak van saját beépített parancsa is a peremérték feladatok megoldására, a bvp4c parancs (bvp = boundary value problem). Nézzünk most egy példát tartószerkezet témaköréből!

Az alábbi L=4 m hosszú kéttámaszú tartóra egyenletesen megoszló terhelés hat (q). Nagy lehajlásokra, a tartó lehajlása (y) a következő közönséges differenciál egyenlettel határozható meg:



$$EI\frac{d^2y}{dx^2} = \left[1+\left(\frac{dy}{dx}\right)^2\right]^{\frac{3}{2}} \cdot \frac{1}{2} \cdot q \cdot (L \cdot x - x^2),$$

$y(0) = 0; y(L) = 0;$ - peremfeltételek

$EI = 1.4 \times 10^7 Nm^2$; - hajlítási merevség

$q = 10 \times 10^3 N/m$; - terhelés

Oldjuk meg a feladatot a Matlab beépített függvényével! Határozzuk meg a lehajlás értékeket *x* függvényében és jelenítsük meg őket! Jelenítsük meg az első deriváltakat is! Mennyi lesz a lehajlás értéke 1.35 m-nél? Mennyi a maximális lehajlás értéke? Milyen x értékeknél lesz a lehajlás pontosan 1 mm?

A Matlab beépített **bvp4c** peremérték feladat megoldó függvényét a következő formában lehet megadni:

```
> SOL = bvp4c(ODEFUN,BCFUN,SOLINIT)
```

A **bvp4c**-nek egy kimenete van (most ez a solution rövidítéseként a sol, de bármilyen változónév lehet), ami egy struktúra típusú változóként tartalmazza a független változót (sol.x) és a kiszámolt függő változó és derivált értékeket is (sol.y).

Három bemenete van:

- *odefun*: elsőrendű differenciálegyenlet rendszer (függvény)
- *bcfun*: a peremértékek függvényként megadva
- *solinit*: kiértékelendő tartomány, illetve a függvény és deriváltjai átlagos értékének becslése (a **bvpinit** paranccsal előállítva)

ELSŐRENDŰ DIFFERENCIÁLEGYENLET RENDSZER MEGADÁSA (ODEFUN)

Ehhez is, mint a kezdeti értékeknél használt **ode45** parancsnál át kell alakítani a magasabb rendű differenciálegyenletet egy elsőrendű differenciálegyenlet rendszerré (*odefun*). Ehhez először fejezzük ki a második deriváltat a másodrendű differenciálegyenletből!

$$\frac{d^2y}{dx^2} = \left(\left[1 + \left(\frac{dy}{dx} \right)^2 \right]^{\frac{3}{2}} \cdot \frac{1}{2} \cdot q \cdot (L \cdot x - x^2) \right) \frac{1}{EI}$$

Alakítsuk át egy új kételemű vektorváltozó bevezetésével elsőrendű differenciálegyenlet rendszerré! Legyen most: $w_1 = y; w_2 = \frac{dy}{dx}$.

$$w = \begin{pmatrix} y & \frac{dy}{dx} \end{pmatrix}^T$$

Ekkor az elsőrendű differenciálegyenlet rendszer:

$$f_1 = \frac{dw_1}{dx} = \frac{dy}{dx} = w_2$$

$$f_2 = \frac{dw_2}{dx} = \frac{d^2y}{dx^2} = \left([1 + w_2^2]^{\frac{3}{2}} \cdot \frac{1}{2} \cdot q \cdot (L \cdot x - x^2) \right) \frac{1}{EI}$$

Definiáljuk a differenciálegyenlet rendszert egy külön függvényben. Ezt megtehetjük egy külön fájlban, ahogy eddig is tettük, vagy a script fájl végén is (a Matlab R2016b verziójától). Legyen a függvény neve **diff_lehajlas**!

```
> function F = diff_lehajlas(x,w)
>   EI = 1.4e7; q = 10e3; L = 4;
>   f1 = w(2);
>   f2 = ((1+(w(2))^2)^(3/2) * (1/2) * q * (L*x-x^2) ) / EI;
>   F = [f1; f2];
> end
```

PEREMÉRTÉKEK FÜGGVÉNYKÉNT MEGADVA (BCFUN)

A peremfeltételek (a lehajlás az alátámasztásoknál):

$$y(0) = 0; \qquad y(L) = 0;$$

A **bvp4c** számára a peremfeltételeket is egy függvényben kell megadni (*bcfun*). Az [a,b] tartományon a feltételeket a wa és wb vektorokban adhatjuk meg, wa a tartomány elején megadott feltételeket jelenti, wb pedig a tartomány végén megadott feltételeket. A vektorok első eleme $(wa(1), wb(1))$ az egyenletrendszer függő

változójának értéke $(w_1 = y)$ a kezdőpontban/végpontban, vagyis a keresett függvény értéke a kezdő/végpontban, a második eleme $(wa(2), wb(2))$ az első derivált értéke $\left(w_2 = \frac{dy}{dx}\right)$ a kezdő/végpontban és így tovább a 3., 4. stb. elem a második, harmadik stb. derivált értéke lenne. Jelen esetben mivel a keresett lehajlás (elmozdulás) függvény értékei adottak a kezdő és végpontban:

$$wa(1) = 0, wb(1) = 0$$

A peremértékek vektorfüggvénye legyen mondjuk g. Megadása úgy történik, mint zérushely kereséskor, amikor a 0-ra rendezett alakot kell megadni, tehát

$$g_1 = wa(1) - 0 = 0$$

$$g_2 = wb(1) - 0 = 0$$

Ez tulajdonképpen a maradék ellentmondásokat jelenti, ha ezek 0-k, akkor teljesíti a függvény a megadott feltételeket. Jelen esetben mivel pont 0 a peremérték mindkét helyen ezért egyszerűen $g_1 = wa(1),\ \ g_2 = wb(1)$. Ha a mondjuk a kezdőpontban az első derivált értéke lenne megadva, például 2, a végpontban pedig a keresett függvény értéke 3, akkor a megadás a következő lenne: $g_1 = wa(2) - 2,\ \ g_2 = wb(1) - 3.$

Legyenek a peremfeltételek a **perem_lehajlas** függvényben (ez lehet külön fájlban vagy a script végén). Mivel most a lehajlás értékekre van feltételünk mindkét végpontban, ezek az *wa* és *wb* első elemei adottak.

```
> function G=perem_lehajlas(wa,wb)
>     g1=wa(1);
>     g2=wb(1);
>     G = [g1; g2];
> end
```

KIÉRTÉKELENDŐ TARTOMÁNY, ISMERETLENEK ÁTLAGOS ÉRTÉKE (SOLINIT)

A harmadik szükséges bemenet a **bvp4c** függvény használatához a kiértékelendő tartomány (*solinit*), illetve a függvény és deriváltjai átlagos értékének becslése (konstansként), amiket a **bvpinit** paranccsal kell megadni. A tartományunk most $0 \leq x \leq 4$. Vegyünk fel ezen 10 cm-ként pontokat. Meg kell még becsülni a függvény és az első deriváltjának is az átlagos értékét. Mivel most csak a függvény értékére van adatunk a kezdő és végpontban, és ezek mindegyike 0, legyen a becsült értékünk is 0 mind a függvényre, mind az első deriváltra. (Ha a kezdő és végpontban eltérő lenne az érték, akkor célszerű lenne a kettő átlagát választani itt a konstans becslésére.) A megoldást a **lehajlas_megoldas.m** fájlban készítsük el!

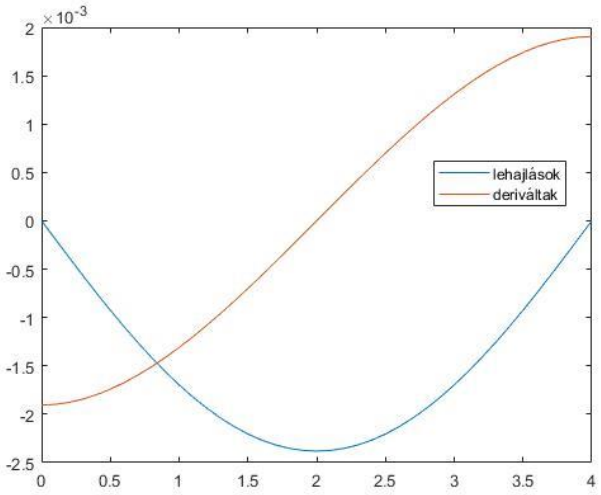
```
> clc; clear all; close all;
> % Becsült kezdeti értékek -  bvpinit
> x0 = 0:0.1:4; % kiértékelendő tartomány
> w10 = 0; w20 = 0; % becsült konstans értékek a függvényre és az első
  deriváltra
> kezdofelt = bvpinit(x0,[w10; w20])
```

MEGOLDÁS BVP4C PARANCCSAL

A megoldáshoz adjunk meg 10^{-4} relatív toleranciát. Ezt nem az **odeset**, hanem a **bvpset** paranccsal tehetjük meg. Utána oldjuk meg a feladatot a **bvp4c** parancsot használva és ábrázoljuk is a megoldást!

```
> % megoldás a beépített bvp4c függvénnyel
> opciok = bvpset('RelTol',1e-4);
> sol = bvp4c(@diff_lehajlas,@perem_lehajlas,kezdofelt,opciok)
> X = sol.x;W = sol.y;
> plot(X,W(1,:)); hold on
> plot(X,W(2,:));
> legend('lehajlások','deriváltak', 'Location','best')
```

Mivel a megoldás struktúra típusú, így sol.x és sol.y hivatkozással tudjuk szétválasztani a független változót (X) tartalmazó vektort és a keresett függvényt és első deriváltjait tartalmazó (W) mátrixot.



Az eddigi megoldás egyben a következőképp néz ki:

```
> % Kétámaszú tartó lehajlása
> clear all; clc; close all;
> % Becsült kezdeti értékek -  bvpinit
> x0 = 0:0.1:4; % kiértékelendő tartomány
> w10 = 0; w20 = 0; % becsült konstans értékek a függvényre és az első
  deriváltra
> kezdofelt = bvpinit(x0,[w10; w20])
> % megoldás a beépített bvp4c függvénnyel
> opciok = bvpset('RelTol',1e-4);
> sol = bvp4c(@diff_lehajlas,@perem_lehajlas,kezdofelt,opciok)
> X = sol.x;
> Y = sol.y;
> plot(X,Y(1,:)); hold on
> plot(X,Y(2,:));
> legend('lehajlások','deriváltak', 'Location','best')
> %-------------------------------------
> function F = diff_lehajlas(x,w)
>   EI = 1.4e7; q = 10e3; L = 4;
>   f1 = y(2);
>   f2 = ((1+(w(2))^2)^(3/2) * (1/2) * q * (L*x-x^2) ) / EI;
>   F = [f1; f2];
```

```
> end
> %-------------------------------------
> function G=perem_lehajlas(wa,wb)
>     g1=wa(1);
>     g2=wb(1);
>     G = [g1; g2];
> end
> %-------------------------------------
```

Kérdés volt még, hogy mennyi lesz a lehajlás értéke 1.35 m-nél, mennyi a maximális lehajlás értéke és milyen x értékeknél lesz a lehajlás pontosan 1 mm?

Ezeket megoldhatnánk úgy is, hogy egy spline függvényt illesztünk a kapott pontokra, de használhatjuk a **deval** parancsot is, ami tetszőleges pontban kiértékeli a bvp4c megoldását (**ode45** esetén is használható, ha azt nem két kimenettel hívjuk meg, hanem eggyel, akkor az is egy struktúra típusban adja vissza az értékeket). A **deval** csak interpolációra használható, extrapolációra nem!

```
> % Mennyi lesz a lehajlás értéke 1.35 m-nél?
> e1 = deval(sol,1.35) %-0.0021, -0.0009
> e1 = deval(sol,1.35,1) %-0.0021
> % Tehát 2.1 mm a lehajlás
> hold on; plot(1.35,e1,'ro')
```

Csak a megoldás és a kiértékelendő hely megadásakor a **deval** függvény használata során megkapjuk az adott pontban a függvény értékét, és az első derivált értékét is, ha csak az első (vagy a második) érdekel minket ezek közül, akkor megadhatjuk még a minket érdeklő változó indexét is. Most a lehajlás 2.1 mm lett a kérdéses pontban.

Mekkora lesz a maximális lehajlás? Ehhez definiáljuk függvényként a kapott megoldást! Mivel a koordináta rendszer úgy van felvéve, hogy a lehajlásokat negatív számként kapjuk, igazából egy minimumot keresünk, így egyszerűen használjuk most az **fminsearch** parancsot erre! Ehhez kezdőértéknek adjuk meg a 2-t!

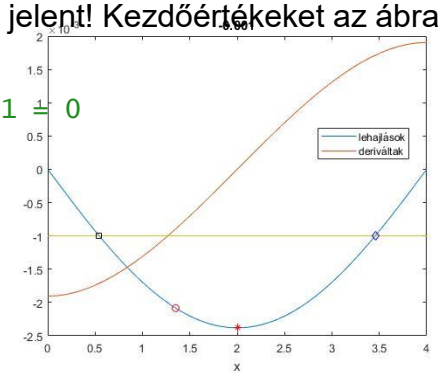
```
> % Mekkora lesz a maximális lehajlás?
> lehajlas = @(x) deval(sol,x,1)
> xmax = fminsearch(lehajlas,2) % 2
> lmax = lehajlas(xmax) % -0.0024
```

Mivel itt a terhelés teljesen szimmetrikus volt, egyértelmű volt, hogy a felezőpontban (2 méternél) lesz a maximális lehajlás, így akár le is kérdezhettük volna ott az értéket a **deval** használatával:

```
> lmax = deval(sol,2,1) %  -0.0024
> plot(xmax,lmax,'r*')
```

Milyen x értékeknél lesz a lehajlás pontosan 1 mm? Ez két pontot is jelenthet, melyeket az **fzero** használatával kereshetünk meg. Figyeljünk ismét a koordináta rendszerre és a mértékegységekre, hogy 1 mm lehajlás itt -0.001 m-t jelent! Kezdőértékeket az ábra alapján válasszunk 0.5 illetve 3.5 m-t!

```
> % lehajlas = -0.001 -> h = lehajlas + 0.001 = 0
> h = @(x) lehajlas(x) + 0.001
> x01 = 0.5; x02 = 3.5;
> x1 = fzero(h,x01) % 0.5437
> x2 = fzero(h,x02) % 3.4563
> plot(x1,lehajlas(x1),'ks'); hold on;
> plot(x2,lehajlas(x2),'bd')
```

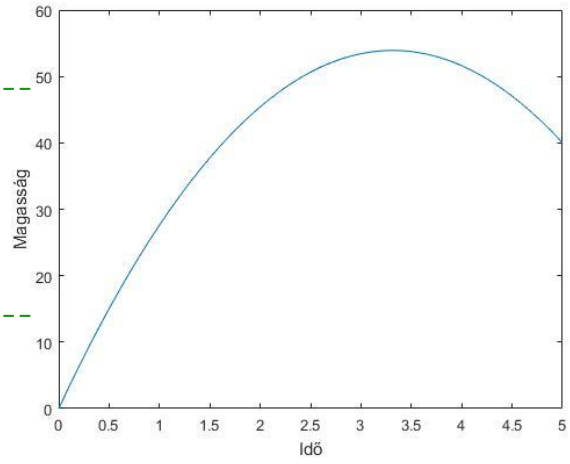


FÜGGŐLEGES MOZGÁS MODELLEZÉSE BVP4C FÜGGVÉNYT HASZNÁLVA

Nézzük meg hogyan nézne ki az elsőnek megoldott petárda kilövéses példánk a Matlab beépített függvényével! A másodrendű differenciálegyenlet: $\frac{d^2y}{dt^2} = -g$, a peremfeltételek: $y(0) = 0;\ y(5) = 40.$

A megoldás **bvp4c** használatával:

```
% petárda - bvp4c
clc; clear all; close all;
% Becsült kezdeti értékek -  bvpinit
t0 = 0:0.1:5; % kiértékelendő tartomány
w10 = 20; % becsült átlagos magasság (0+40)/2
w20 = 0; % becsült érték az átlagos első deriváltra
kezdofelt = bvpinit(t0,[w10; w20])
% megoldás a beépített bvp4c függvénnyel
opciok = bvpset('RelTol',1e-4);
sol = bvp4c(@diff_petarda,@perem_petarda,kezdofelt,opciok)
X = sol.x;
W = sol.y;
plot(X,W(1,:));
xlabel('Idő'); ylabel('Magasság')
%-------------------------------------
function F = diff_petarda(t,w)
  g = 9.81;
  f1 = w(2);
  f2 = -g;
  F = [f1; f2];
end
%-------------------------------------
function G=perem_petarda(wa,wb)
    g1=wa(1);
    g2=wb(1)-40;
    G = [g1; g2];
end
%-------------------------------------
```



GYAKORLÓ PÉLDA PEREMÉRTÉK FELADATOKHOZ

Oldjuk meg most a következő peremérték feladatot a [0,1] tartományon:

$$\frac{d^2y}{dx^2} + y = 0$$

Átrendezve:

$$\frac{d^2y}{dx^2} = -y$$

A megadott peremfeltételek: $y(0) = 1; \quad \frac{dy}{dx}(1) = 3$

Az intervallum elején adott a függvényérték, az intervallum végén pedig az első derivált! Az elsőrendű differenciálegyenlet rendszer, ha $w_1 = y; w_2 = \frac{dy}{dx}$:

$$f_1 = \frac{dw_1}{dx} = \frac{dy}{dx} = w_2$$

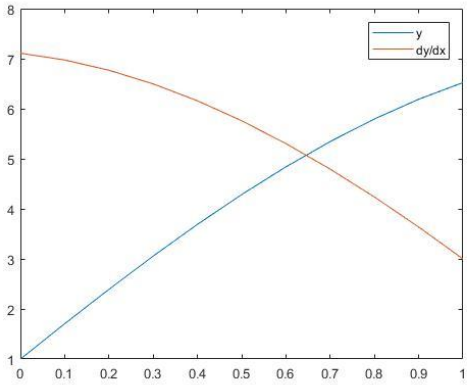
$$f_2 = \frac{dw_2}{dx} = \frac{d^2y}{dx^2} = -w_1$$

A peremfeltételek: $wa(1) = 1,\ wb(2) = 3$

Nullára rendezve: $g_1 = wa(1) - 1;\ g_2 = wb(2) - 3$

Megoldás Matlab-ban:

```
> % gyakorló peremérték feladat
> clc; clear all; close all;
> clc; clear all; close all;
> % Becsült kezdeti értékek -  bvpinit
> x0 = 0:0.1:1; % kiértékelendő tartomány
> w10 = 1; % becsült átlagos függvényérték
> w20 = 2; % becsült érték az első deriváltra
> kezdofelt = bvpinit(x0,[w10; w20])
> % megoldás a beépített bvp4c függvénnyel
> sol = bvp4c(@gyakorlo_diff,@gyakorlo_peremfelt,kezdofelt)
> X = sol.x; W = sol.y;
> plot(X,W(1,:),X,W(2,:));
> legend('y','dy/dx')
> %-----------------------------------
> function F = gyakorlo_diff(x,w)
>     f1 = w(2);
>     f2 = -w(1);
>     F = [f1; f2];
> end
> %-----------------------------------
> function G=gyakorlo_peremfelt(wa,wb)
>     g1=wa(1)-1;
>     g2=wb(2)-3;
>     G = [g1; g2];
> end
> %-----------------------------------
```



A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

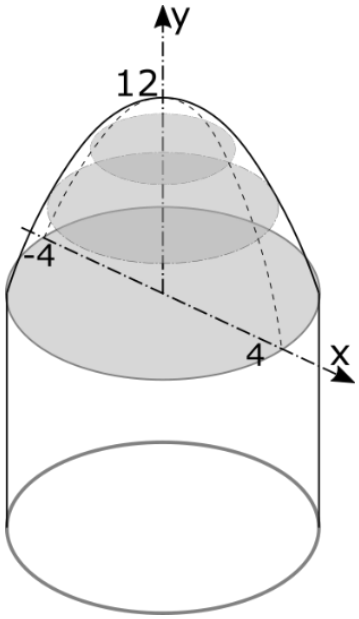
bvp4c	-	Közönséges differenciál egyenletek peremérték feladatának megoldása kollokációval
bvpinit	-	Kezdeti értékek becslése közönséges differenciálegyenletek peremérték feladatának megoldásához
bvpset	-	Közönséges differenciálegyenlet peremérték feladatát megoldó függvények opcionális paramétereinek megadása (pl. RelTol, AbsTol)
deval	-	Közönséges differenciálegyenlet megoldásának kiértékelése adott pontban

17. GYAKORLÓ FELADATOK 2.

A feladatok megoldásait Matlab fájlokban kell elkészíteni, amelyben a végrehajtható utasításokon túl legyenek megjegyzések is a megoldás lépéseiről.

1. MINTA ZÁRTHELYI DOLGOZAT

A lenti képen látható gabonasiló teteje a következő függvénnyel megadott görbe y tengely körüli körbeforgatásával állítható elő *x*=0-tól *x*=4-ig:



$$y = 12\cos^2\left(\frac{\pi}{8}\,x\right)$$

Egy $y = f(x)$ görbe $[a, b]$ közötti szakaszának körbeforgatásából kapott felület nagysága a következő képlettel számítható:

$$A = 2\,\pi \int_a^b x\sqrt{1 + (f'(x))^2}$$

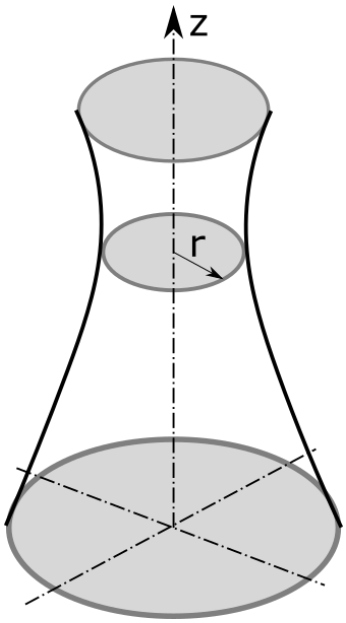
a) Definiálja a görbét és ábrázolja a [0,4] intervallumon! (3 pont)
b) Szimbolikus számítással állítsa elő a görbe derivált függvényét! (4 pont)
c) Számítsa ki a tető felületének nagyságát! (6 pont)
d) Mekkora részét tudjuk lefesteni a siló tetejének 12 liter festékből, ha 10 m^2 lefestéséhez 1.2 liter festékre van szükség és fentről kezdjük a festést? A megoldáshoz definiáljon egy függvényt, ami adott x értékhez kiszámolja a tető felületét és keresse meg, hogy ez milyen x értéknél felel meg a lefesthető felület nagyságának. A kezdeti érték megválasztásához ábrázolja a függvényt egy új ábrán! Mekkora y nagyságú sáv marad ki a festésből alul? (8 pont)
e) Határozza meg, hogy mekkora a tető térfogata! A siló tető részének térfogatát kiszámolhatja, egy forgástest térfogatának képletéből:

$$V = \pi \int_a^b r^2 dy$$

Most a forgástest sugarát az x értékek adják, ezért a térfogat számításához definiálja az *y* görbe inverz függvényét, az $x = g(y)$ függvényt. (Matlab-ban az $arccos$ függvénynek az *acos* parancs felel meg.). Rajzolja fel az inverz függvényt is egy új ábrába, és számítsa ki a térfogatot! Figyeljen az intervallum határainak helyes megadására! (7 pont)
f) Számolja ki a minimális és maximális y értékek között fél méteres osztásban a hozzájuk tartozó sugár értékeket (x)! Ezeket a pontokat rajzolja be az első ábrába! (2 pont)
g) Az előbb kiszámolt összetartozó y,x értékeket írja ki az fprintf használatával egy szöveges fájlba két oszlopba, egy tizedesre az y értékeket és két tizedesre az x értékeket! (5 pont)

2. MINTA ZÁRTHELYI DOLGOZAT

A képen látható hűtőtorony sugarát különböző magasságokban megmérték. Számítsuk ki a hűtőtorony térfogatát és felszínét!



.

a) Olvassa be az **adat01.txt** fájlban található adatokat. Az első oszlopban a magasságok, a második oszlopban a hozzájuk tartozó sugarak vannak méterben. Válassza szét a változókat és jelenítse meg a sugarakat a magasság függvényében! (2 pont)
b) Egy forgástest felszínét (A) és térfogatát (V) a következőképp számíthatjuk ki:

$$A = 2\,\pi \int_0^L r\,dz\,; \qquad V = \pi \int_0^L r^2\,dz$$

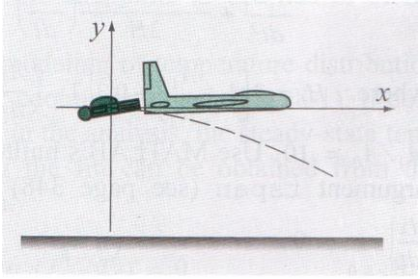
A beolvasott adatokat használva számítsa ki trapéz szabállyal a torony felszínét és térfogatát (6 pont)!
c) Illesszen köbös másodrendű spline görbét a pontokra és ezt felhasználva számítsa ki a felszínt és a térfogatot Simpson szabály alapján is a Matlab beépített függvényével! Az illesztett függvényt rajzolja be az ábrába is! (8 pont)
d) Az illesztett spline alapján keresse meg, hogy mekkora magasságban van a torony legkeskenyebb része? Mekkora itt az átmérő nagysága? (4 pont)
e) Szeretnék a tornyot lefesteni egy 10 m széles sávban. Hol legyen a sáv, hogy a lehető legkisebb felületet kelljen lefesteni? Ehhez definiáljon egy függvényt, ami adott kezdőponttól kiszámolja a felületet egy 10 m-es sávban, és keresse meg ennek a minimumát! A jó kezdőérték megválasztásához ábrázolja a függvényt (vigyázzon a határok megadásakor, hogy a maximális torony magasságot ne lépje túl)! Adja meg a lefestendő terület nagyságát, a sáv aljának és tetejének magasságát! Mennyibe fog kerülni a festék, ha 1.2 liter festékkel 10 m^2-t lehet lefesteni és 1 liter festék 540 Ft-ba kerül? (8 pont)
f) Számolja ki a torony sugarának értékeit két méterenként a magasság függvényében! (2 pont)
g) Írja ki a kiszámolt pontok adatait egy szöveges fájlba két oszlopba az fprintf használatával, egész számként a magasságokat és 3 tizedesre a hozzájuk tartozó sugár értékeket! (5 pont)

3. MINTA ZÁRTHELYI DOLGOZAT

Egy ejtőernyős kiugrik egy egyenesen, vízszintesen haladó repülőből. Az ejtőernyős mozgása közelítőleg az alábbi egyenletrendszerrel írható le:

$$\frac{d^2x}{dt^2} = -\frac{\gamma}{m}\left(\frac{dx}{dt}\right)\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}$$

$$\frac{d^2y}{dt^2} = -g - \frac{\gamma}{m}\left(\frac{dy}{dt}\right)\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}$$



ahol x és y az ejtőernyős pozíciója, az ábrán látható koordináta rendszernek megfelelően.

$m = 80\ kg; g = 9.81 m/s^2;\ \gamma\ = 5.38\ Ns^2/m^2$

A kezdeti feltételek:

$$x(0) = 0; y(0) = 0;\ \left.\frac{dx}{dt}\right|_{t=0} = 134\ m/s;\ \left.\frac{dy}{dt}\right|_{t=0} = 0;$$

Határozza meg az ejtőernyős mozgásának pályáját az első 5 másodpercben. A két másodfokú közönséges differenciál egyenletet vezesse vissza 4 elsőfokú közönséges differenciál egyenletre és oldja meg a feladatot tetszőleges módszerrel.

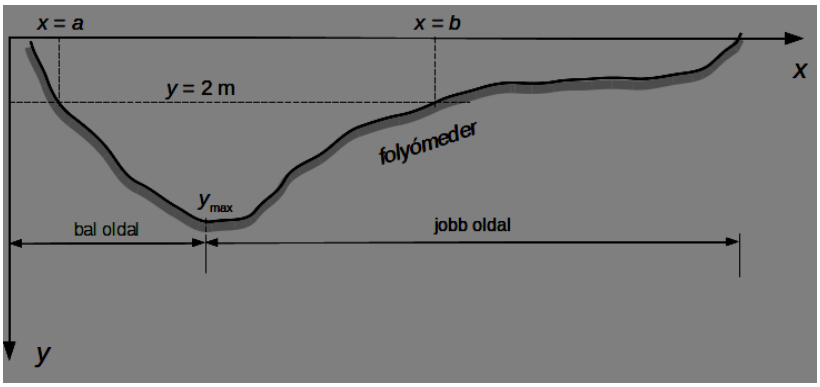
a) Vezesse vissza a feladatot új változók bevezetésével 4 elsőrendű differenciál egyenletre, és írja meg a rendszert reprezentáló Matlab függvényt! (Ez lehet egysoros függvény is, vagy külön függvényben is megadva)
b) Oldja meg az egyenletrendszert tetszőleges módszerrel az első 5 másodpercre, felhasználva a 4 kezdeti értéket!
c) Rajzolja fel az ejtőernyős pályáját az xy koordináta rendszerben! Két másik ábrába rajzolja fel a következő foronómiai görbéket: x,y elmozdulás és v_x, v_y sebességek az idő függvényében!
d) Mennyi 5 másodperc után az elmozdulás és a sebesség a két koordináta tengely irányában?
e) Mennyi a függőleges elmozdulás 2.25 másodperc után?
f) Az ejtőernyős 1500 m-es magasságban nyitja ki az ejtőernyőjét. Mikor éri el ezt a magasságot, ha a repülőgép az ugráskor 2000 m-en haladt? (Ehhez modellezze hosszabb intervallumra, pl. egy percre az ejtőernyős pályáját!)

4. MINTA ZÁRTHELYI DOLGOZAT

Adottak a Tisza Mártély és Mindszent közötti szakasza egy szelvényében a felmért folyómeder pontjainak (x, y) koordinátái a `szelveny.txt` állományban. Az x koordináták az első, az y meder mélység adatok a második oszlopban találhatók. A meder bal- illetve jobb oldali végpontja egyaránt az $y = 0$ m-es szinten található.

Oldja meg az alábbi feladatokat

1) Olvassa be az adatokat egy mátrixba és válassza külön egy-egy vektorba az x és y koordinátákat.
2) Határozza meg spline interpolációval a meder alakját. Írja meg az interpoláció egysoros $y_S(x)$ függvényét és rajzolja fel az $y_S(x)$ függvény segítségével a szelvényt a minimális x_{min} és maximális x_{max} koordináták között.
3) Határozza meg numerikus integrálással, Simpson-szabállyal a meder teljes F keresztmetszeti területét. Az integrálást végezze a meder spline interpoláció függvényével az x_{min} és x_{max} határok között.
4) Spline interpolációval vegyen fel a meder teljes szélességében az x koordináta szerint egyenletesen elhelyezkedő 200 pontot. Az így kapott poligon területét trapézokra bontással határozza meg, és hasonlítsa össze a numerikus integrálással kapott eredménnyel.
5) Határozza meg a folyómeder *nedvesített* kerületét (az előző pontban kapott poligon hosszát) $y = 0$ m-es vízállás esetében.
6) Határozza meg a keresztmetszet $F(y = 2)$ területét az ábrán látható $y = 2$ m-es vízállás esetében is numerikus integrálással, Simpson-szabállyal. Ehhez meg kell határoznia az ábrán látható $x = a$ és $x = b$ integrálási határokat, valamint el kell tolnia a folyómeder görbéjét 2 m-el az y tengellyel ellentétes irányban. Az integrálási határokat az $y_S(x) - 2 = 0$ nemlineáris egyenlet gyökkeresésével kaphatja meg. A megfelelő kezdőértékeket az ábráról veheti.
7) Határozza meg a meder bal- és jobb szakaszának (a meder legmélyebb pontjától balra, illetve jobbra eső szakaszoknak, lásd az ábrát) az $x_{bal}(y)$ és $x_{jobb}(y)$ inverzfüggvényeit spline interpolációval. Rajzolja fel egy-egy ábrán ezeket a függvényeket. (4 pont)
8) Határozza meg a keresztmetszet y mélység szerint változó $F(y)$ függvényét numerikus integrálással, Simpson-szabállyal. Az integrálási határok megállapításához használja az előző pontban megírt inverzfüggvényeket. Ábrázolja egy új ábrán az $y = (0, 9)$ tartományban a kapott mélység-keresztmetszet függvényt.



5. MINTA ZÁRTHELYI DOLGOZAT

Adottak egy ismeretlen sugarú és helyzetű gömb felületének lézerszkenneres mérési adatai a `gomb.txt` állományban. Ebben az állományban az első, második és harmadik oszlopban a mért pontok (x_i, y_i, z_i) koordinátái találhatók. Határozzuk meg a mért pontokra legjobban illeszkedő gömb R sugarát és a középpontjának (x_0, y_0, z_0) koordinátáit. A regressziós felület egyenlete $(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - t_0)^2 - R^2 = 0$. Ha bevezetjük az R sugár helyett a $d = x_0^2 + y_0^2 + z_0^2 - R^2$új paramétert, akkor a feladatot *lineáris* regresszióként írhatjuk fel az alábbi alakban

$$x_i^2 + y_i^2 + z_i^2 = 2x_i x_0 + 2y_i y_0 + 2z_i z_0 - d,$$

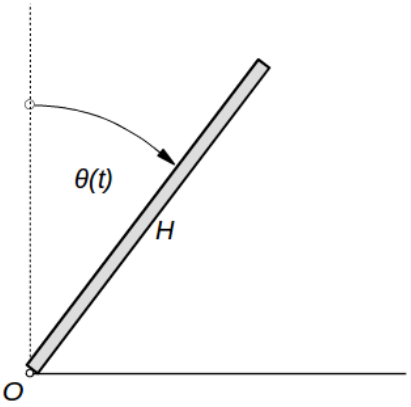
ahol a bal oldalon a mért mennyiségek, jobb oldalon pedig a $p = (x_0, y_0, z_0, d)$paraméterekre nézve lineáris függvény található.

Oldja meg az alábbi feladatokat:

1) Olvassa be az adatokat egy mátrixba és válassza külön egy-egy vektorba az x, y és z koordinátákat.
2) Rajzolja fel a mért pontokat egy térbeli ábrán.
3) Határozza meg lineáris regresszióval, a Matlab beépített függvényével az ismeretlen paraméterek értékét. Figyeljen arra, hogy a mérési vektor most a regressziós egyenlet bal oldalán található $x_i^2 + y_i^2 + z_i^2$ mennyiségeket tartalmazza.
4) Számítsa ki a legjobban illeszkedő gömb sugarát és írassa ki ezt az eredményt, valamint a gömb középpontjának a koordinátáit.
5) Rajzolja fel a mért pontokat tartalmazó ábrába a legjobban illeszkedő gömböt paraméteres felületként. A gömb paraméteres egyenletei $u \in (0, \pi), v \in (0, 2\pi)$:
$$x(u, v) = R\sin(u)\cos(v)$$
$$y(u, v) = R\sin(u)\sin(v).$$
$$z(u, v) = R\cos(u)$$
6) Számítsa ki mért pontoknak az illeszkedő gömbtől vett *sugár irányú* eltérései *előjeles* összegét (ha a mért pont a gömb felszíne fölött van, akkor az eltérés pozitív, egyébként negatív). Egy külön ábrán rajzolja fel az eltéréseket.

6. MINTA ZÁRTHELYI DOLGOZAT

Az ábrán látható $H = 2$ m hosszú, az O pontban csuklósan befogott rúd kezdetben a függőlegessel $\theta_0 = 1°$-os szöget zár be. Ebben a helyzetben elengedjük a rudat és az eldől. A rúd helyzetét a t időpontban a függőlegessel bezárt $\theta(t)$ szög jellemzi. A rúd mozgását az alábbi másodrendű differenciálegyenlettel írhatjuk le ($g = 9.81$ m/s^2):



$$\frac{d^2\theta}{dt^2} = \frac{3g}{2H} sin\theta.$$

Oldja meg az alábbi feladatokat:

1) Alakítsa át a másodrendű egyenletet két elsőrendű egyenletből álló rendszerré és írja meg a jobb oldalak egysoros függvényét
2) Oldja meg a differenciálegyenlet-rendszert a t =[0, 2] tartományon, a Matlab beépített 4. rendű ode45 Runge-Kutta módszerével. A relatív hiba legyen 10^{-4} , az abszolút hiba pedig 10^{-5}. A $t = 0$ időpontban a rúd helyzete: $\theta = \pi/180$, szögsebessége: $d\theta/dt = 0$. Ábrázolja a $\theta(t)$ szögre kapott megoldást
3) Írja meg a $\theta(t)$ spline interpolációját végző egysoros függvényt
4) Határozza meg az előző interpolációs függvény segítségével azt a t_v időpontot amikor a rúd éppen vízszintes helyzetben van, vagyis $\theta(t_v) = \pi/2$. Rajzolja be ezt a helyzetet az ábrába
5) Valamely θ szöggel jellemzett helyzetben a rúd O pontjától $H \cdot x$ távolságban található keresztmetszetben a hajlításból származó legnagyobb σ feszültség értéke

   $$\sigma(x, \theta) = \frac{dmgH}{8I} sin\theta(x^3 - 2x^2 + x) = 4.057 sin\theta(x^3 - 2x^2 + x)\ [MPa]$$

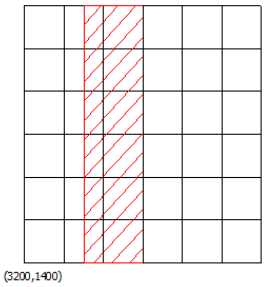
   ahol $d = 2$ cm a rúd vastagsága, $m = 2.2$ kg a tömege, $I = 1.33$ cm^4 a négyzet alakú rúd keresztmetszet súlyponti tengelyére vonatkozó inerciája. Írja meg a $\sigma(x, \theta)$ egysoros függvényét (vektorosan)
6) Rajzolja fel a rúd teljes H hossza mentén a hajlításból származó feszültségek eloszlását a $\theta = 0.5$ értékére a $\sigma(x, \theta)$-nak az előző pontban megírt egysoros függvényét felhasználva
7) A rúd O pontjától mérve hány m-re található az a keresztmetszet, ahol maximális a σ feszültség értéke? (Használja a Matlab beépített függvényét a $-\sigma(x)$ minimuma megkeresésére)

7. MINTA ZÁRTHELYI DOLGOZAT

A ***felszin.dat*** és ***agyag.dat*** állományokban egy felszín alatti agyagmező feltárásához végzett területszintezés és ehhez kapcsolódó próbafúrások adatait rögzítették. A szintezést és a fúrásokat egy 7x7-es szabályos rácsháló sarokpontjaiban végezték el. A terület mérete 600x600 m, a bal alsó sarok koordinátája [3200,1400]. Az adatállományokban a mérési eredmények északról délre és nyugatról keletre változnak.

Ezen adatok felhasználásával oldja meg az alábbi feladatokat.

1) Olvassa be a megadott állományokat és állítsa elő a rács x,y koordinátáit is!
2) A pontokra illesszen egy-egy harmadfokú spline felületet, és jelenítse meg a két szintfelületet egy közös ábrán!
3) Határozza meg a szintfelületek közé eső tartomány térfogatát és ezzel becsülje meg az agyagmező eléréséhez szükséges tereprendezés térfogatát!
4) A kitermelést egy 150x600 m-es, a koordináta-tengelyekkel párhuzamos sávban kívánják elvégezni (lásd ábra) úgy, hogy az agyagot 120 m-es szintig bányásszák ki. Hol válasszák meg ezt a sávot, hogy a kitermelt agyag mennyisége a lehető legnagyobb legyen? Ennek a kérdésnek a megválaszolásához az alábbi eljárást alkalmazza:

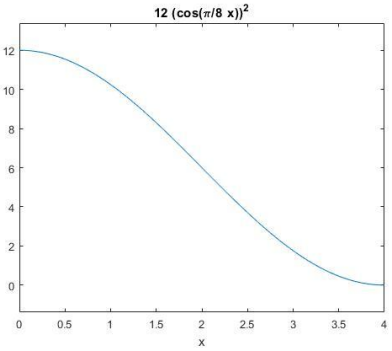


   a) Definiáljon egy egysoros függvényt, ami egy tetszőleges x koordináta esetén megadja az innen induló 150x600 m-es tartományba eső agyag mennyiségét.
   b) Ezen függvény felhasználásával számítsa ki 50 m-es lépésközzel a területet lefedő tartományokba eső agyag mennyiségét. Ábrázolja is az eredményt! (A feladat megoldásához használjon ciklust, mivel a Matlab kettős integrált használó algoritmusai nem fogadnak el vektorváltozókat. Figyeljen, hogy a tartomány ne lógjon ki a teljes 600x600-as területről!)
   c) Illesszen spline görbét az előbb meghatározott adatokra, rajzolja be az előbbi ábrába és keresse meg a maximumát! Mi lesz az ehhez az elemhez tartozó területnek a bal alsó koordinátája? Ez adja meg a kereset terület helyzetét.
5) Hány százaléka lesz az így kitermelhető agyag mennyisége a megadott 120 m-es szintig kitermelhető teljes (becsült) agyagmennyiségnek?

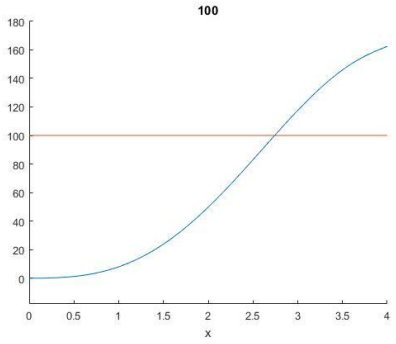
MEGOLDÁSOK

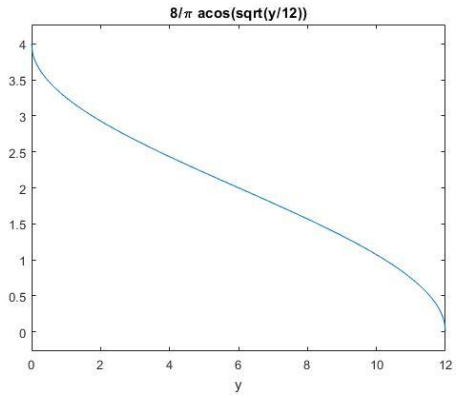
1. minta zárthelyi dolgozat megoldása

```
%% Gabonasiló, definiálás, ábrázolás (3 pont)
clc; clear all; close all;
y = @(x) 12*(cos(pi/8*x)).^2
figure(1)
ezplot(y,[0,4]);
```

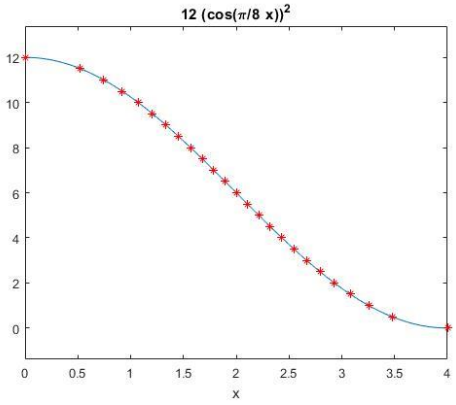



```
%% derivált fv. (4 pont)
syms x
dy = diff(y(x)) % -3*pi*cos((pi*x)/8)*sin((pi*x)/8)
dy = matlabFunction(dy)
% @(x)pi.*cos(x.*pi.*(1.0./8.0)).*sin(x.*pi.*(1.0./8.0)).*-3.0

%% Felszín: A = 2*pi*int(x*sqrt(1+(f')^2))dx (6 pont)
fx = @(x) x.*sqrt(1+(dy(x)).^2)
A = 2*pi*integral(fx,0,4) % 162.4232

%% Mekkora részét tudjuk lefesteni a siló tetejének (8 pont)
% 12 liter festékből (1.2 liter 10m^2)? Mekkora sáv marad ki?
m2 = 12/1.2*10 % 100 m^2
F = @(x) 2*pi*integral(fx,0,x)
figure(2); hold on;
ezplot(F,[0,4])
ezplot('100',[0,4])
F2 = @(x) F(x)-m2
xfestes = fzero(F2,2.5) % 2.7416
yfestes = y(xfestes) % 2.6994 sáv marad ki

%% Térfogat V = pi*int(x^2*dy) (7 pont)
x1 = @(y) 8/pi*acos(sqrt(y/12))
figure(3)
ezplot(x1,[0,12])
x2 = @(x) x1(x).^2
V = pi*integral(x2,0,12) % 179.3619

%% fél méteres szintekhez tartozó sugár értékek (2 pont)
yi = 0:0.5:12;
```



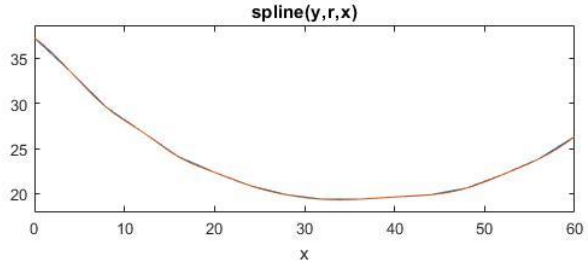


```
xi = x1(yi)
figure(1); hold on;
plot(xi,yi,'r*')


%% kiírás fájlba (5 pont)
fid = fopen('silopontok.txt','w');
for i = 1:length(xi)
    fprintf(fid,'%4.1f %.2f\r\n',yi(i),xi(i));
end
fclose(fid)
```

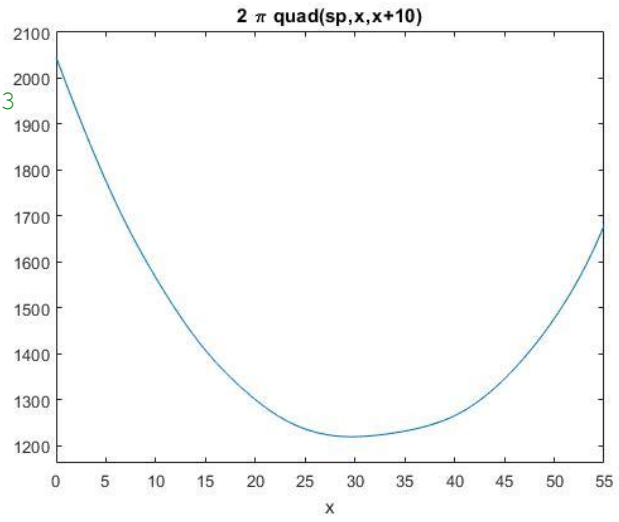


2. minta zárthelyi dolgozat megoldása

```
% hűtőtorony, % A = 2*pi*int(r*dy), V = pi*int(r^2*dy)
%% a) beolvasás, megjelenítés - (2 pont)
clc; clear all; close all;
adat = load('adat01.txt')
y = adat(:,1); r = adat(:,2);
figure(1); plot(y,r); axis equal;
```

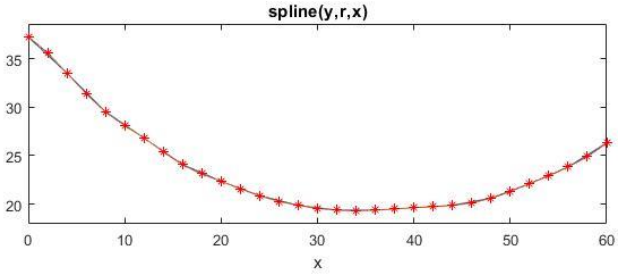



```
%% Felület, térfogat - trapéz módszerrel (6 pont)
A = 2*pi*trapz(y,r) % 8.8876e+03
V = pi*trapz(y,r.^2) % 1.0905e+05


%% %% Felület, térfogat - Simpson módszerrel, spline illesztés (8 pont)
sp = @(x) spline(y,r,x)
hold on; ezplot(sp,[0 y(end)])
A1 = 2*pi*quad(sp,0,y(end)) % 8.8752e+03
R2 = @(x) sp(x).^2;
V1 = pi*quad(R2,0,y(end)) % 1.0866e+05


%% legkeskenyebb pont (4 pont)
y0 = 35;
ymin = fminsearch(sp,y0) % 33.8887
rmin = sp(ymin) % 19.3167
dmin = rmin*2 % 36.5636
```




```
%% 10 méter széles sáv lefestése, minimális felület (8 pont)
A10 = @(x) 2*pi*quad(sp,x,x+10)
figure(2)
```

```
ezplot(A10,[0, 50])
saveleje = fminsearch(A10,30) % 29.6742
savvege = saveleje + 5 % 34.6742
felulet = A10(saveleje) % 1.2202e+03
% festék ára (10m^2 - 1.2 liter, 1 liter 540 Ft)
liter = felulet/10*1.2 % 146.4203
ar = liter*540 % 7.9067e+04
```




```
%% Sugár értékek méterenként (2 pont)
ym = 0:2:60;
rm = sp(ym);
figure(1); plot(ym,rm,'r*')

%% kiírás fájlba (5 pont)
fid = fopen('toronypontok.txt','w');
for i = 1:length(ym)
    fprintf(fid,'%2d %6.3f\r\n',ym(i),rm(i));
end
fclose(fid)
```

3. minta zárthelyi dolgozat megoldása

Vezessünk be két új változót az első deriváltakra, vagyis a koord. tengelyek szerinti sebességekre: $\frac{dx}{dt} = u$ és $\frac{dy}{dt} = w$, alakítsuk át négy elsőrendű differenciálegyenletté a két másodrendű egyenletet:

$$f_1 = \frac{dx}{dt} = u$$

$$f_2 = \frac{du}{dt} = \frac{d^2x}{dt^2} = -\frac{\gamma}{m}\left(\frac{dx}{dt}\right)\sqrt{\left(\frac{\mathrm{d}x}{\mathrm{d}t}\right)^2 + \left(\frac{dy}{dt}\right)^2} = -\frac{\gamma}{m}u\sqrt{u^2 + w^2}$$

$$f_3 = \frac{dy}{dt} = w$$

$$f_4 = \frac{dw}{dt} = \frac{d^2y}{dt^2} = -g - \frac{\gamma}{m}\left(\frac{dy}{dt}\right)\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2} = g - \frac{\gamma}{m}w\sqrt{u^2 + w^2}$$

A változóink legyenek az *xy* vektorban: $xy = [x; u; y; w]$. Írjuk fel Matlab-ban az egyenletrendszert, most egysoros függvényként:

```
g = 9.81; gamma = 5.38; m = 80;
xydiff = @(t,xy) [xy(2);
```

```
-(gamma/m)*xy(2)*sqrt(xy(2)^2+xy(4)^2);
xy(4);
-g-(gamma/m)*xy(4)*sqrt(xy(2)^2+xy(4)^2)]
```

Oldjuk meg a feladatot!

```
% Egyenletrendszer megoldása beépített függvénnyel az első 5 másodpercben
[T,XY]=ode45(xydiff,[0,5],[0; 134; 0; 0])
X = XY(:,1); U = XY(:,2); Y = XY(:,3); W = XY(:,4);
```

Rajzoljuk fel a megoldásokat!

```
%% Az ejtőernyős pályája xy koordináta rendszerben
figure(1); plot(X,Y)
title('Az ejtőernyős pályája az első 5 másodpercben')
%% x,y koordináta az első 5 másodpercben
figure(2); plot(T,X,T,Y);
legend('x koordináta','y koordináta','Location','best')
title('Az x,y irányú elmozdulás az idő függvényében')
%% x,y irányú sebesség az első 5 másodpercben (foronómiai görbék)
figure(3); plot(T,U,T,W);
legend('x irányú sebesség','y irányú sebesség','Location','best')
title('Az x,y irányú sebesség az idő függvényében')
```

Elmozdulás és sebesség 5 másodperc után

```
%% Elmozdulás és sebsesség 5 másodperc után a koordináta tengelyek irányában
disp('vízszintes elmozdulás 5s után: '); disp(X(end)) % 50.1310
disp('függőleges elmozdulás 5s után: '); disp(Y(end)) % -44.2489
disp('vízszintes sebesség 5s után: '); disp(U(end)) % 0.5648
disp('függőleges sebesség 5s után: '); disp(W(end)) % -12.0214
```

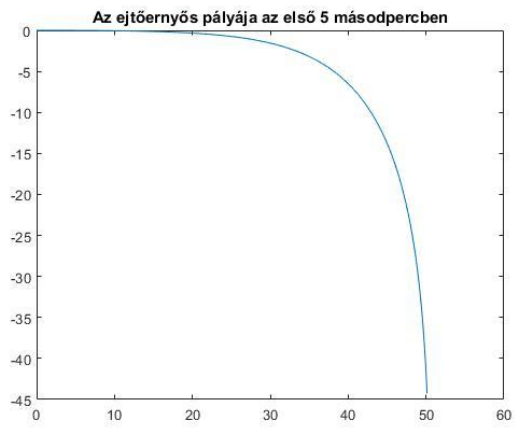
Mennyi a függőleges elmozdulás 2.25 másodperc után?

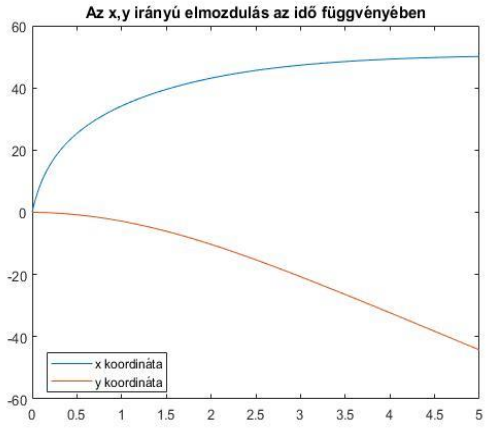
```
%% Mennyi függőleges elmozdulás 2.25 másodperc után?
ysp = @(x) spline(T,Y,x);
ysp(2.25) % -12.7098 y irányú elmozdulás 2.25 másodperc után
```

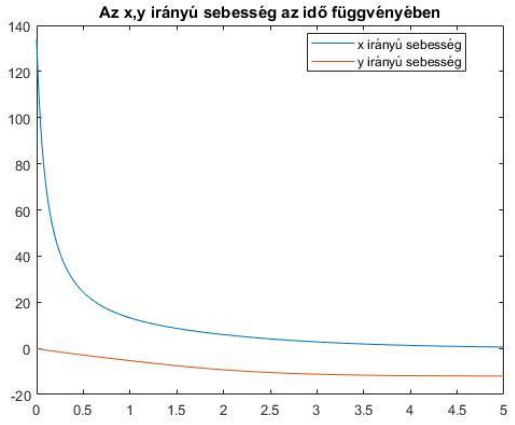
A függőleges elmozdulás: (2000 m-ről 1500 m-re): y=-500 m. Ehhez számoljuk ki hosszabb intervallumon, mondjuk 1 perc esetén, az elmozdulás értékeket!

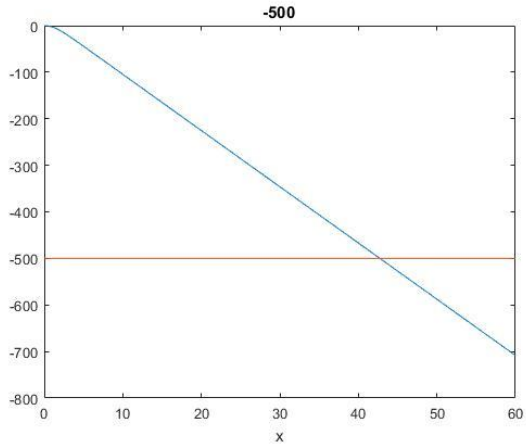
```
%% f) A függőleges elmozdulás mikor lesz -500 m?
[T,XY]=ode45(xydiff,[0,60],[0; 134; 0; 0]);
X = XY(:,1);U = XY(:,2); Y = XY(:,3); W = XY(:,4);
Y(end) %  -708.4927 m
```

```
figure(4); plot(T,Y); hold on; ezplot('-500',[0 60]);
ysp = @(x) spline(T,Y,x);
h = @(x) ysp(x)+ 500;
nyitas = fzero(h,40) %  42.7377 s
```









4. minta zárthelyi dolgozat megoldása

```
clc; clear all; close all;
% 1.) adatok beolvasása - folyómeder
xy = load('szelveny.txt');
% adatok szétválasztása
x = xy(:,1); y = xy(:,2);

% 2.)spline interpoláció
ys = @(u) spline(x,y,u);
% ábra készítése
xmin = min(x);xmax = max(x);
ezplot(ys,[xmin,xmax])

% 3.) keresztmetszeti terület meghatározása Simpson-szabállyal
Aq = quad(ys,xmin,xmax) % Aq = 758.7124

% 4.) spline interpolációval a szelvény 200 pontjának meghatározása
xv = linspace(xmin,xmax,200); yv = ys(xv);
% poligon területének számítása trapézokra bontással
xv1 = [xv,xv(end)]; yv1 = [yv,yv(end)];
Tp = trapz(xv1,yv1) % Tp =  758.6506
```

```
% 5.) nedvesített kerület meghatározása poligon hosszaként
dx = diff(xv); dy = diff(yv);
% a nedvesített kerület
s = sum(sqrt(dx.^2+dy.^2)) % s =  142.3642

% 6.) terület meghatározása 2 m-es vízállás esetén
% ábra a vízállásról
hold on; ezplot('2',[xmin,xmax])
% metszéspontok számítása
y2 = @(x) ys(x)-2;
a = fzero(y2,5) % a = 7.9048
b = fzero(y2,140) % b =  139.714
% integrálás Simpson-szabállyal
A2 = quad(y2,a,b) % A2 = 486.2315

% 7.) maximumhely keresése (meder legmélyebb pontja)
ysm = @(u) spline(x,-y,u);
fminsearch(ysm,45) % 45.2252
% meghatározzuk az inverz függvényeket
% bal oldal
xa = linspace(xmin,45,50); ya = ys(xa);
ysa = @(u) spline(ya,xa,u);
% jobb oldal
xb = linspace(46,xmax,150); yb = ys(xb);
ysb = @(u) spline(yb,xb,u);
% felrajzoljuk
figure(2); ezplot(ysa,[0,9])
figure(3); ezplot(ysb,[0,9])

% 8.) változó h mélységre felírt folyómeder alak függvénye
yh = @(x,h) ys(x)-h;
% numerikus integrálás az inverzfüggvényekből kiszámított határok között
Ah = @(h) quad(@(x) yh(x,h),ysa(h),ysb(h));
figure(4); ezplot(Ah,[0,9])
```

5. minta zárthelyi dolgozat megoldása

```
% gömb illesztése
clc; clear all; close all;
% 1.) adatok beolvasása, változók szétválasztása
xyz = load('gomb.txt');
x = xyz(:,1);y = xyz(:,2);z = xyz(:,3);
% adatok száma
n = length(x);

% 2.) felrajzoljuk az adatokat
scatter3(x,y,z,'filled')

% 3.) a gömb illesztése
% yr mérési vektor
b = x.^2 + y.^2 + z.^2;
% A alakmátrix
A = [2*x, 2*y, 2*z, -ones(n,1)];
% regresszió számítása
[p pint h] = regress(b,A)
% vagy
p = A\b

% 4.) illeszkedő gömb sugara
R = sqrt((p(1)^2+p(2)^2+p(3)^2)-p(4)) % R = 5.0496
% gömb középpontja
```

```
p(1:3) % 1.0510, 2.1116, 1.5695

% 5.) ábrázoljuk a regressziós gömböt paraméteres felületként
% x->x0+R*sin(t)*cos(s)
% y->y0+R*sin(t)*sin(s)
% z->z0+R*cos(t)
figure(1); hold on
ezsurf('1.05+5.05*sin(t)*cos(s)','2.11+5.05*sin(t)*sin(s)','1.57+5.05*cos(t
)',[0,pi,-pi,pi])

% 6.) az illeszkedő gömbtől vett sugár irányú eltérések előjeles összege
% pontok távolsága a gömb középpontjától
Rp = sqrt((x-p(1)).^2+(y-p(2)).^2+(z-p(3)).^2);
% eltérések
dR = Rp-R;
% előjeles összeg
sum(dR) %-0.2117
% ábrázoljuk az eltéréseket
figure(2);
bar(dR)
```

6. minta zárthelyi dolgozat megoldása

```
% csuklósan befogott rúd ledőlése
clear all; clc; close all
% 1) a megadott differenciálegyenlet jobb oldalának egysoros függvénye
%    elsőrendű rendszerré átalakítva
H = 2;
dthdt = @(t,th) [th(2);3*9.81/(2*H)*sin(th(1))]

% 2) az egyenlet megoldása a t=(0,2) tartományon és ábra
options = odeset('RelTol', 1e-4, 'AbsTol',[1e-5 1e-5]);
th0 = pi/180;
[T,X]=ode45(dthdt,[0,2],[th0;0], options);
plot(T,X(:,1))

% 3) a megoldás spline interpolációs függvénye
ths = @(t) spline(T,X(:,1),t)

% 4) a vízszintes helyzet elérésének időpontja
f = @(t) ths(t)-pi/2
% t kezdőértéke az ábráról 2.0
tv = fsolve(f,2.0) %tv =  1.9342
% berajzoljuk az ábrába
hold on; plot(tv,ths(tv),'ro')

% 5) hajlítófeszültség értéke - egysoros függvény vektorosan
sigma = @(x,theta) 4.057*sin(theta).*(x.^3-2*x.^2+x)

% 6) hajlítófeszültség eloszlása a theta=0.5 értékére
% a rúd 2 m-es hosszára átskálázott sigma
sigs = @(x) sigma(x/2,0.5)
figure(2); ezplot(sigs , [0,2])

% 7) a rúd melyik keresztmetszetében lesz maximális a feszültség?
% kezdeti érték az ábráról 0.6
lmax = fminunc(@(x) -sigs(x), 0.6) %lmax =  0.6667
% berajzoljuk
hold on; plot(lmax,sigs(lmax),'ro')
```

7. minta zárthelyi dolgozat megoldása

```
% 1) felszín, agyag beolvasás
clc;clear all; close all;
%%  A felszín és  az agyagszint magasságainak a beolvasása
Zf=load('felszin.dat')
Za=load('agyag.dat')
%  A rácspontok x,y koordinátái
x=3200:100:3800;
y=2000:-100:1400;     % Az adatok sorrendje!
%  A rács előállítása
[X,Y]=meshgrid(x,y)

%% 2)  Alkalmazzunk köbös spline interpolációt mindkét felületre
F=@(u,v)interp2(X,Y,Zf,u,v,'cubic');
A=@(u,v)interp2(X,Y,Za,u,v,'cubic');
figure(1);hold on;
ezsurf(F,[3200,3800,1400,2000]) %  A felszín megjelenítése
ezsurf(A,[3200,3800,1400,2000]) %  Az agyag réteg megjelenítése

%% 3)  A terep alatti földtömeg számítása a Simpson szabály alapján
(integrálás)
Vf=integral2(F,3200,3800,1400,2000) % 4.6027e+07
Va=integral2(A,3200,3800,1400,2000) % 4.4968e+07
%  A tereprendezés térfogata
Vagyag=Vf-Va %  1.0594e+06

%% 4) Agyagkitermelés a megadott tartományon
% a) feladat
agyag = @(x) integral2(A,x,x+150,1400,2000)-120*150*600

% b) a maximális x koord. 3650 m lehet, hogy a sáv ne lógjon ki a
területről
x50 = 3200:50:3650;
for i=1:length(x50)
    agyag50(i) = agyag(x50(i));
end
figure(2)
plot(x50,agyag50,'r');

% c) feladat
intsp = @(u) spline(x50,agyag50,u);
figure(2); hold on;
ezplot(intsp,[3200,3650])
intsp1 = @(x) -intsp(x) % maximum kereséshez -1 szeres fv.
maxhely = fminsearch(intsp1,3300) % 3.2997e+03
maxagyag = intsp(maxhely) % 5.9344e+05
plot(maxhely,maxagyag,'ro')
% A keresett terület bal alsó koordinátái: 3299.7; 1400

%% 5) Maximálisan kitermelhető agyag a teljes területen 120 m-ig
agyagosszes = integral2(A,3200,3800,1400,2000)-120*600*600 %  1.7680e+06
%  Százalékos arány számítása
agyag_szazalek = maxagyag/agyagosszes*100 % 33.5658
```

FELHASZNÁLT IRODALOM

Detrekői Ákos (1991): Kiegyenlítő számítások, Tankönyvkiadó, Budapest

Gilat, Amos; Subramaniam, Vish (2011): Numerical methods, An introduction with applications using MATLAB, John Wiley & Sons, Inc

Paláncz Béla (2012): Numerikus módszerek példatár, https://oktatas.epito.bme.hu/local/coursepublicity/mod/folder/view.php?id=47427

Stoyan, Gisbert; Takó Galina; Gyurkovics Éva (2005): Numerikus módszerek I., Typotex kiadó

Young, Todd; Mohlenkamp, Martin J. (2018): Introduction to Numerical Methods and Matlab Programming for Engineers, Department of Mathematics, Ohio University, http://www.ohiouniversityfaculty.com/youngt/IntNumMeth/book.pdf

MELLÉKLETEK

1. MELLÉKLET –MATLAB FÜGGVÉNYEK TÉMAKÖRÖNKÉNT

MATLAB BEVEZETÉS

help	-	matlab helpjének kategóriái, vagy segítség megadott témakörhöz, függvényhez
rand	-	Véletlen számok 0-1 között egyenletes eloszlásban
randn	-	Véletlen számok sztenderd normális eloszlásban, 0 várható értékkel és 1 szórással
doc	-	részletes dokumentáció adott függvényhez, parancshoz
lookfor	-	keresés a help-ben adott szóra, szórészletre
clc	-	kitörli a command window ablak tartalmát
clear, clear all	-	kitörli a megadott változókat, vagy az összes változót
close, close all	-	bezárja az aktuális ábrát, vagy az összeset
CTRL+C	-	félbeszakítja az adott parancsot (kilépés pl. végtelen ciklusból)
%	-	megjegyzés (a program figyelmen kívül hagyja ami ez után van a sorban)
;	-	parancs végén a ; hatására nem jelenik meg az eredmény a Command Window-ban
Tab gomb	-	elkezdett parancsot kiegészíti
preferences	-	megnyitja a beállítások ablakot
prefdir	-	annak a könyvtárnak a neve, ahol a beállítások, history stb. található
↑↓ gombok	-	korábbi parancsokat vissza lehet hozni a Command Window-ba
pi	-	3.14…. (pi szám)
exp(1), exp(n)	-	e^1 =2.71…, e^n
^	-	hatványozás
format long	-	több tizedes jegy megjelenítése
format short	-	rövidebb megjelenítés
[1, 2, 3; 4, 5, 6]	-	vektor, mátrix megadása
'	-	vektor, mátrix transzponáltja
[A,B] vagy [A B]	-	mátrixok összefűzése egymás mellé (sorok száma egyenlő)
[A;B]	-	mátrixok összefűzése egymás alá (oszlopok száma egyenlő)
A(1,:)	-	mátrix első sora
A(:,1), A(:,end)	-	mátrix első/utolsó oszlopa
linspace(x1,x2,n)	-	[x1,x2] intervallumban n pont felvétele egyenletesen
ones	-	egyesekből álló mátrix
zeros	-	nullákból álló mátrix
eye	-	egységmátrix
figure	-	új ábra nyitása
plot	-	összetartozó pontpárok felrajzolása
xlabel, ylabel	-	x,y tengely feliratozása

title	-	ábra címe
sin, cos, tan	-	szögfüggvények (alapértelmezett mértékegység a radián!)
log, log10	-	természetes alapú logaritmus, 10-es alapú logaritmus
sqrt	-	négyzetgyök
abs	-	abszolút érték
hold on, hold off	-	felülírja, vagy ne írja felül a meglévő ábrát az új ábrával
fplot, ezplot	-	függvények felrajzolása
.* ./ .^	-	elemenkénti szorzás, osztás, hatványozás vektoroknál
clf	-	ábra törlése (nem zárja be az ablakot)
legend	-	jelmagyarázat

disp	-	Szöveg, változók tartalmának kiírása a parancssorba
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, save	-	Adatok betöltése/elmentése (Matlab adatállományból/ba (*.mat), és egyszerű szöveges fájlból/ba)
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

SZÁMÍTÁSOK HIBÁI

==	-	Logikai egyenlőség
~=	-	Logikai ’nem egyenlő’
eps	-	Gépi epszilon/gépi pontosság nagysága,
factorial	-	Faktoriális, n!
inv	-	Mátrix inverze
cond	-	Kondíció szám
loglog	-	Ábrázolás logaritmikus skálán (mindkét tengelyen)
syms	-	Szimbolikus változók, kifejezések definiálása
simplify	-	Szimbolikus kifejezések egyszerűsítése
matlabFunction	-	Szimbolikus kifejezések függvénnyé alakítása

NEMLINEÁRIS EGYENLETEK

set	-	Grafikus elem tulajdonságainak beállítása (pl. Color, LineWidth)
and(felt1, felt2), felt1 && felt2,	-	Logikai ÉS
diff	-	Szimbolikus deriválás
sym	-	Kifejezések, változók szimbolikussá alakítása
fzero	-	Egyváltozós egyenlet gyökeinek megkeresése numerikusan
det	-	Mátrix determinánsa
solve	-	Algebrai polinom gyökei szimbolikusan
roots	-	Algebrai polinom gyökei numerikusan
double	-	Szimbolikus kifejezésként megadott szám lebegőpontos számmá alakítása
real	-	Képzetes szám valós része
sym2poly	-	Szimbolikusan megadott algebrai polinom együtthatóinak kigyűjtése egy vekorba
eig	-	Mátrix sajátértékeinek, sajátvektorainak meghatározása

LINEÁRIS EGYENLETRENDSZEREK

rank	-	Mátrix rangja
lu	-	LU felbontás
linsolve	-	Lineáris egyenletrendszer megoldása kiegészítő opciókkal (pl. alsó/felső háromszögmátrix, szimmetrikus, pozitív definit). Általános négyzetes mátrix esetén LU felbontást használ.
pascal	-	Előállíthatjuk a binomiális együtthatókat tartalmazó szimmetrikus Pascal mátrixot
diag	-	Kivehetjük egy mátrix főátlójából az elemeket vagy egy vektorból csinálhatunk vele diagonális mátrixot
min	-	Egy vektor legkisebb eleme
max	-	Egy vektor legnagyobb eleme
chol	-	Cholesky felbontás

norm	-	Vektor/mátrix normája (’hossza’)
tic, toc	-	Időmérés kezdete, vége
\ vagy mldivide	-	Általános lineáris egyenletrendszer megoldása (négyzetes mátrix esetén LU vagy Cholesky felbontással)
qr	-	QR felbontás
svd	-	SVD felbontás
pinv	-	Pszeudoinverz számítás SVD felbontással
type	-	Szöveges fájl tartalmának képernyőre írása
tril	-	Egy mátrix alsó háromszögmátrixa
nargin	-	Függvény megadott bemenő paramétereinek a száma
gmres	-	Lineáris egyenletrendszer iteratív megoldása
sparse	-	Ritka mátrixok tárolása

NEMLINEÁRIS EGYENLETRENDSZEREK

fimplicit	-	f(x,y)=0 implicit alakban megadott függvények ábrázolása
axis equal	-	Egyenlő beosztás a tengelyeken
jacobian	-	Jacobi mátrix kiszámítása (egyenletet parciális deriváltjai)
fsolve	-	Nemlineáris egyenletrendszerek megoldása numerikusan
solve	-	Algebrai polinomokból álló egyenletrendszer megoldása szimbolikusan

REGRESSZIÓ

axis	-	Tengelyek minimális, maximális értékeinek megadása
mean	-	Vektor elemeinek számtani közepe, átlaga
sum	-	Vektor elemeinek összege
corr2	-	Lineáris korrelációs együttható
polyfit	-	Megadott fokszámú polinom illesztése az adatokra
polyval	-	Együttható vektorral megadott polinom értékének kiszámítása
bar	-	Ábrázolás oszlopdiagrammon
subplot	-	Egy grafikus ablakon belül több ábra

INTERPOLÁCIÓ

vander	-	Vandermonde mátrix
interp1	-	Egydimenziós interpoláció (módszer: lineáris – ’linear’, legközelebbi szomszéd - ’nearest’, köbös másodrendű – ’spline’, köbös elsőrendű Hermite interpoláció – ’pchip’)
spline	-	Egydimenziós, köbös másodrendű spline interpoláció

2D INTERPOLÁCIÓ

diff	-	vektor elemeinek különbsége, közelítő numerikus derivált, szimbolikus derivált számítása
cumsum	-	vektor elemeinek folyamatos összegzése
meshgrid	-	2-3 dimenziós rács előállítása vektorban tárolt x,y(,z) koordinátákból
plot3	-	Pontok 3D megjelenítése
mesh	-	Rácshálóban adott 3D pontok megjelenítése térbeli rácsként
surf	-	Rácshálóban adott 3D pontok megjelenítése színezett felületként (kitöltött térbeli rács)
contour	-	Rácshálóban adott 3D pontok alapján szintvonalak rajzolása
set	-	grafikus objektum (pl. h) megadott tulajdonságainak beállítása, pl. szintvonalak feliratozása (set(h,'ShowText','on') contour parancs esetén, vagy set(h,'Show',on) ezcontour esetén)
interp2	-	2D interpoláció rácshálóban adott pontokból tetszőleges pontra (módszer: lineáris – 'linear', legközelebbi szomszéd - 'nearest', spline inetrpoláció – 'spline', 2D köbös spline (bicubic) – 'cubic')
integral2	-	Kettős integrál számítása numerikusan, szabályos téglalap tartományon
scatter3	-	Szórt pontok 3D megjelenítése
regress	-	Többváltozós lineáris regresszió legkisebb négyzetek módszerével
fsurf, ezsurf	-	3D felületek kirajzolása megadott függvény alapján
fcontour, ezcontour	-	Szintvonalak kirajzolása függvény alapján
griddata	-	Interpoláció szórt pontok alapján tetszőleges pontra vagy rácsra (módszer: háromszög alapú lineáris interpoláció (TIN modell) – 'linear', legközelebbi szomszéd - 'nearest', háromszög alapú köbös interpoláció – 'cubic', biharmonikus spline inetrp. – 'v4')

NUMERIKUS DERIVÁLÁS

polyder	-	Algebrai polinom deriváltjának számítása
diff(f,x,2)	-	f szimbolikus kifejezés 2. deriváltja x szerint
gradient	-	Gradiensek számítása numerikusan, szimbolikusan
quiver	-	vektormező megjelenítése
hessian	-	Hesse-mátrix, az f(x) függvény második parciális deriváltjainak a mátrixa

OPTIMALIZÁCIÓ

subs	-	szimbolikus változóba konkrét értékek behelyettesítése
fminsearch	-	Egy/többváltozós függvény minimának megkeresése Nelder-Mead szimplex módszert alkalmazva
fminunc	-	Feltétel nélküli szélsőérték keresés kvázi-Newton minimalizálást alkalmazva.

NUMERIKUS INTEGRÁLÁS

trapz(x,y)	-	Numerikus integrálás diszkrét pontok alapján trapéz szabállyal
quad(fun,a,b)	-	Függvény numerikus integrálása Simpson-szabállyal
integral(fun,a,b)	-	Függvény numerikus integrálása adaptív kvadratúrával
integral2	-	Kettős integrál számítása numerikusan, szabályos téglalap tartományon
integral3	-	Hármas integrál számítása numerikusan, szabályos téglatest tartományon
rectangle	-	Téglalap rajzolás
haltonset(n)	-	n dimenziós Halton sorozat előállítása
net(hset,n)	-	n pont kiválasztása a Halton sorozatból
inpolygon	-	egy zárt poligonon belül lévő pontok meghatározása
nnz	-	nem nulla elemek száma

DIFFERENCIÁLEGYENLETEK

ode45	-	Közönséges differenciálegyenlet rendszer kezdeti érték problémájának megoldása Runge-Kutta módszerrel
odeset	-	Közönséges differenciálegyenlet kezdeti érték feladatát megoldó függvények opcionális paramétereinek megadása (pl. RelTol, AbsTol, MaxStep, InitialStep)
bvp4c	-	Közönséges differenciál egyenletek peremérték feladatának megoldása kollokációval
bvpinit	-	Kezdeti értékek becslése közönséges differenciálegyenletek peremérték feladatának megoldásához
bvpset	-	Közönséges differenciálegyenlet peremérték feladatát megoldó függvények opcionális paramétereinek megadása (pl. RelTol, AbsTol)
deval	-	Közönséges differenciálegyenlet megoldásának kiértékelése adott pontban

2. MELLÉKLET –SAJÁT FÜGGVÉNYEK TÉMAKÖRÖNKÉNT

MATLAB BEVEZETÉS

Másodfokú megoldóképlet, megoldások száma

```
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
```

Fok-perc-másodpercben történő kiíratás

```
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
```

SZÁMÍTÁSOK HIBÁI

e^{-x} kétféle közelítése

```
function [f g] = emx(x,n)
    f = 1; % első közelítés 1 - x + x^2/2 - x^3/6 + ...
    p = 1; % első közelítés a nevezőre (1 + x + x^2/2 + x^3/6 + ...)
    for i=1:n
        s = x^i/factorial(i);
        f = f +(-1)^i*s;
        p = p + s;
        g = 1 / p;
    end
end
```

NEMLINEÁRIS EGYENLETEK ZÉRUSHELYE

Intervallum felezés módszere

```
function [c, i] = intfelezes(f, a, b, delta, N)
% Az eljárás az intervallum felezés módszer segítségével határozza
% meg egy egyváltozós nemlineáris egyenlet megoldását
%
% Bemenő paraméterek:
%   f - egyváltozós függvény, aminek a gyökhelyét keressük
%   a,b - a zárt intervallum bal és jobb oldala
%   delta - leállási feltétel, közelítés küszöbértéke
```

```
%   N - leállási feltétel, maximális iteráció szám
%
% Kimenet:
%   c - A megoldás helye
%   i - Iterációk száma
    c = (a+b)/2;    % Intervallumfelezés (1. iteráció)
    i = 1;          % Iterációk száma
    % Leállási feltétel:
    % elérjük a közelítés küszöbértékét vagy a maximális iteráció számot
    while abs (f(c)) > delta && i <= N
        if f(c)*f(a) < 0
            b = c;
        else
            a = c;
        end;
        i = i + 1;
        c = (a+b)/2;
    end;
end
```

Húr módszer

```
function [c, i] = hur(f, a, b, delta, N)
% Az eljárás a húr módszer segítségével határozza meg
% egy egyváltozós nemlineáris egyenlet megoldását
%
% Bemenő paraméterek:
%   f - egyváltozós függvény, aminek a gyökhelyét keressük
%   a,b - a zárt intervallum bal és jobb oldala
%   delta - leállási feltétel, közeltítés küszöbértéke
%   N - leállási feltétel, maximális iterácó szám
%
% Kimenet:
%   c - A megoldás helye
%   i - Iterációk száma

    % A húrnak az x tengellyel való metszése - 1. iteráció
    c = (a*f(b) - b*f(a))/(f(b) - f(a));
    i = 1;          % Iterációk száma
    % Leállási feltétel:
    % elérjük a közelítés küszöbértékét vagy a maximális iteráció számot
    while abs (f(c)) > delta && i <= N
        if f(c)*f(a) < 0
            b = c;
        else
            a = c;
        end;
        i = i + 1;
        c = (a*f(b) - b*f(a))/(f(b) - f(a));
    end;
end
```

Newton módszer

```
function [x1, n] = newtonsys (f, J, x0, eps, nmax)
% Az eljárás a newton módszer segítségével egy többváltozós
% nemlineáris egyenletrendszer megoldását határozza meg.
%
% Bemenő paraméterek:
%   f - többváltozós egyenletrendszer
%   J - az egyenletrendszer Jacobi mátrixa
%   x0 - kezdő érték
%   eps - leállási feltétel, közelítés küszöbértéke
%   nmax - leállási feltétel, maximális iterácó szám
% Kimenet: x1 - a megoldás helye, n - iterációk száma
    dx = J(x0)\-f(x0); % első iteráció
    x1 = x0 + dx;      % első iteráció
    n = 1;
    while norm(x1-x0)>eps && n<=nmax
        x0 = x1;
```

```
        dx = J(x0)\-f(x0);
        x1 = x0 + dx;
        n = n + 1;
    end;
end
```

LINEÁRIS EGYENLETRENDSZEREK

Iteratív megoldás

```
function [x,i,X]=iterativ(A,b,e,imax,method)
% A*x=b Lineáris egyenletrendszer megoldása iteratív módon,
% Jacobi vagy Gauss-Seidel módszerrel (alapértelmezett: Jacobi).
% Bemenet: A, b, e-hibahatár, imax - maximális iteráció szám
% method - 'Jacobi' vagy 'GaussSeidel' (nem kötelező megadni)
% Kimenet: x-megoldás, i -iteráció szám, X-iterációs lépések

% Módszer meghatározása
    if nargin==4; method='Jacobi'; end
    switch method
        case 'Jacobi'
            B = diag(diag(A));  % A mátrix főátlója mátrixban
        case 'GaussSeidel'
            B = tril(A);  % A mátrix alsó háromszög
        otherwise % Jacobi
            B = diag(diag(A)); % A mátrix főátlója mátrixban
    end
% Első iterációs lépés
Ai=eye(5)-inv(B)*A;
bi = inv(B)*b;
x0=ones(size(b)); % kiinduló érték
i=1; x1=Ai*x0+bi; % Első iteráció
X=[x0,x1];
% Iterációs ciklus
while i<=imax && norm(x1-x0)>e
    x0=x1;
    x1=Ai*x0+bi;
    X=[X x1];
    i=i+1;
end
x=X(:,end); % megoldás
end
```

NEMLINEÁRIS EGYENLETRENDSZEREK

Többváltozós nemlineáris egyenletrendszerek megoldása Newton módszerrel

```
function [x1, n] = newtonsys (f, J, x0, eps, nmax)
    dx = J(x0)\-f(x0)); % első iteráció
    x1 = x0 + dx;       % első iteráció
    n = 1;
    while norm(x1-x0)>eps && n<=nmax
        x0 = x1;
        dx = J(x0)\-f(x0);
        x1 = x0 + dx;
        n = n + 1;
    end;
end
```

NUMERIKUS DERIVÁLÁS

Numerikus deriválás differencia formulákkal (centrális, jobb és baloldali)

```
function dx = derivalt(x,y)
% Numerikus deriválás differencia hányadosok alkalmazásával
    n = length(x);
    dx(1) = (y(2)-y(1))/(x(2)-x(1)); % jobboldali differencia
    for i = 2:n-1
```

```
        dx(i) = (y(i+1)-y(i-1))/(x(i+1)-x(i-1)); % centrális diff.
    end
    dx(n) = (y(n)-y(n-1))/(x(n)-x(n-1)); % baloldali differencia
end
```

FELTÉTEL NÉKÜLI OPTIMALIZÁCIÓ

Intervallum módszer (ternary search)

```
function [x, i] = intervallum(f, a, b, tol)
% Az eljárás az intervallum módszer segítségével határozza
% meg egy egyváltozós nemlineáris függvény minimumuhelyét
%
% Bemenő paraméterek:
%   f - egyváltozós függvény
%   a - az intervallum bal oldala
%   b - az intervallum jobb oldala
%   tol - leállási feltétel, intervallum hossza
% Kimenet:
%   x - A megoldás helye
%   i - Iterációk száma

 i = 1;
 x1 = a + 1/3*(b-a);
 x2 = b - 1/3*(b-a);
 while abs(x2-x1) > tol
   if f(x1) < f(x2)
     b = x2;
   else
     a = x1;
   end
   i = i+1;
   x1 = a + 1/3*(b-a);
   x2 = b - 1/3*(b-a);
 end
 x = (x1+x2)/2;
end
```

Aranymetszés módszer

```
function [x, i] = aranymetszes(f, a, b, tol)
% Az eljárás az aranymetszés módszer segítségével határozza
% meg egy egyváltozós nemlineáris függvény minimumuhelyét
%
% Bemenő paraméterek:
%   f - egyváltozós függvény
%   a - az intervallum bal oldala
%   b - az intervallum jobb oldala
%   tol - leállási feltétel, intervallum hossza
% Kimenet: x - A megoldás helye, i - Iterációk száma

i = 1;
R = (sqrt(5)-1)/2;
x1 = b - R*(b-a);
x2 = a + R*(b-a);
f1 = f(x1); f2 = f(x2);
while abs(x2-x1) > tol
  if f1 < f2
     b = x2;
     x2 = x1; f2 = f1; % ezt átvesszük az előző iterációból!
     x1 = b - R*(b-a);
     f1 = f(x1);
  else
     a = x1;
     x1 = x2; f1 = f2; % ezt átvesszük az előző iterációból!
     x2 = a + R*(b-a);
     f2 = f(x2);
  end
```

```
  i = i+1;
end
x = (x1+x2)/2;
```

Aranymetszés rekurzióval

```
function [x i] = golden(f, a, b, r, Tolx, Tolf, i)
% Az eljárás rekurzív aranymetszés módszer segítségével határozza
% meg egy egyváltozós nemlineáris függvény minimumuhelyét
%
% Bemenő paraméterek:
%   f - egyváltozós függvény
%   a - az intervallum bal oldala
%   b - az intervallum jobb oldala
%   r - a felosztás paramétere, egyenlő felosztás esetén: 2/3
%   Tolx - leállási feltétel, intervallum hossza
%   Tolf - leállási feltétel, függvényértékek eltérése
%   i - iterációk száma, a függvény hívásakor i = 0
% Kimenet: x - A megoldás helye, i - Iterációk száma
%
% az iterációk száma egyel nő, ahányszor csak meghívjuk a függvényt:
i = i + 1;
h = b - a;
c = a + (1 - r)*h;
d = a + r*h;
fc = f(c); fd = f(d);
% Leállási feltétel:
% vagy:
% 1. az intervallum hossza a megadott 'Tolx' küszöbérték alá csökkent
% 2. a függvényértékek különbsége az intervallum végpontjaiban
%     a megadott 'Tolf' küszöbérték alá csökkent
if (abs(h) < Tolx || abs(fc - fd) < Tolf) % || logikai 'vagy' művelet
   if fc <= fd
      x = c;
   else
       x = d;
   end
else
    if fc < fd
      [x i] = golden(f, a, d, r, Tolx, Tolf, i);  % rekurzív
függvényhívás az új intervallumra
   else
      [x i] = golden(f, c, b, r, Tolx, Tolf, i);  % rekurzív
függvényhívás az új intervallumra
    end
end
```

Többváltozós Newton módszer optimalizációhoz

```
function [x i X] = gradmulti(grad, hesse, x0, eps, nmax)
% Az eljárás a Newton-módszer segítségével határozza
% meg egy kétváltozós függvény minimumuhelyét
%
% Bemenő paraméterek:
%   grad  - a gradiens vektor számítására szolgáló függvény
%   hesse - a Hesse mátrix számítására szolgáló függvény
%   x0 - a kezdőérték vektora
%   eps   - leállási feltétel, minimális lépéshossz értéke
%   nmax  - leállási feltétel, maximális iterácó szám
% Kimenet: x-megoldás helye, i-iterációk száma, X-iterációs lépések

x1 = x0 - pinv(hesse(x0))*grad(x0);
i=1;
X=[x0 x1];

% Leállási feltételek:
% 1., a lépéshossz a megadott 'eps' küszöbérték alá csökkent,vagy
% 2., elértük a maximális iterációk 'nmax' számát
```

```
while and(norm(x1 - x0) > eps, i < nmax)
    x0 = x1;
    x1 = x0 - pinv(hesse(x0))*grad(x0);
    i = i + 1;
    X = [X x1];
end
x = x1;
```

DIFFERENCIÁLEGYENLETEK – KEZDETI ÉRTÉK PROBLÉMA

Euler-módszer

```
function [t,y] = euler (f, y0, a, b, h)
% Közönséges differenciál egyenletek kezdeti érték problémájának
megoldása
% Euler módszerrel
% Bemenet: f - diff. egy. rsz., y0 - kezdeti értékek,
% a,b - intervallum eleje/vége, h - felosztás nagysága
% Kimenet: t-független változó értékei, y - függő változó és deriváltjai
n = round((b - a)/h);
t(1) = a;
y(1) = y0;
for i = 1 : n
    y(i + 1) = y(i) + h*f(t(i), y(i));
    t(i + 1) = t(i) + h;
end
```