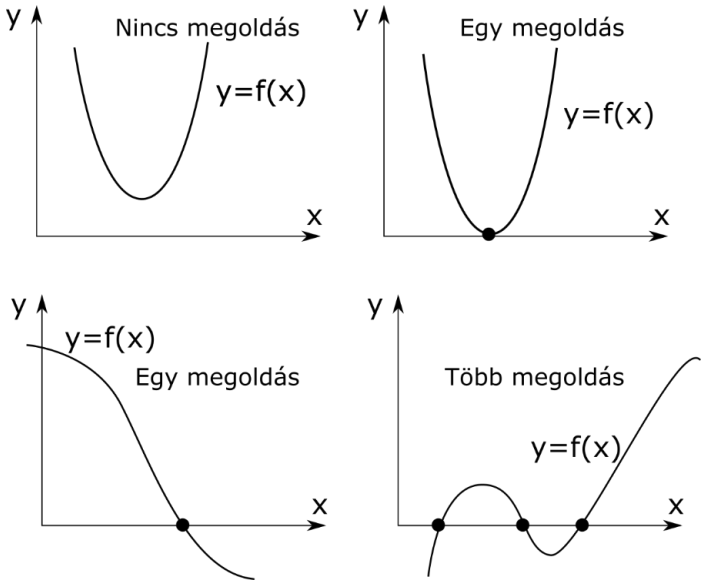
4. NEMLINEÁRIS EGYENLETEK GYÖKEI

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.

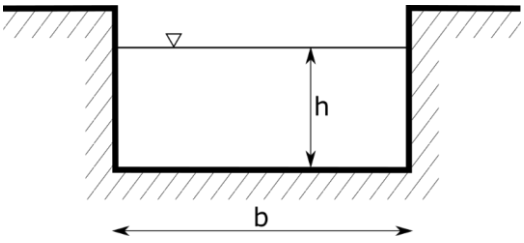
Az $f(x) = 0$ egyenlet megoldása sok esetben csak numerikusan, iterációkkal lehetséges, azaz akkor fogadjuk el a megoldást, ha egy megadott Δ hibakorlát alatt van, $f(x) \leq \Delta$. A megoldási módszerek többsége ún. lokális módszer, azaz az iterációhoz szükség van egy vagy több induló értékre. A megoldásra 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: $S = 0.0008; n = 0.02; Q = 3\ {}^{m^3}/_{s}; b = 2\ m$[1]

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!

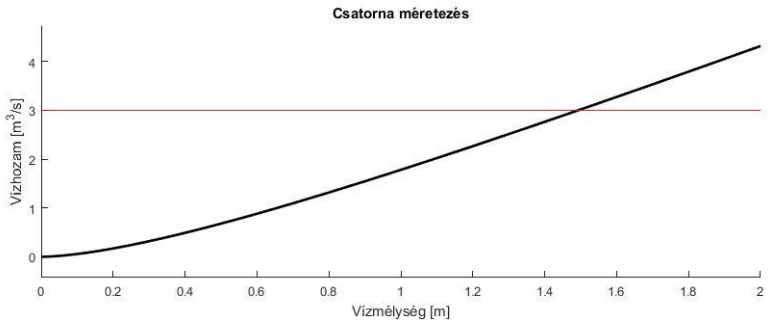
```
> clear all; clc;close all;
> % Változók értékadása
> S = 0.0008; n = 0.02; Q = 3; b = 2;
> % Vízhozam egysoros függvényként: h független változó
> Q=@(h) sqrt(S)/n*(b*h).^(5/3)./(b+2*h).^(2/3);
```

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;
> fplot(Q, [0 2],'Color','k','LineWidth',2);
> % y=3 vonal berajzolása a [0 2] intervallumon két pontot megadva
> plot([0,2],[3,3],'r')
> % vagy használhatjuk az xlim (ábra x irányú határai) parancsot is
> plot(xlim,[3,3],'r')
> % Ábra feliratozás
> title('Csatorna méretezés');
> xlabel('Vízmélység [m]');
> ylabel('Vízhozam [m^3/s]');
```



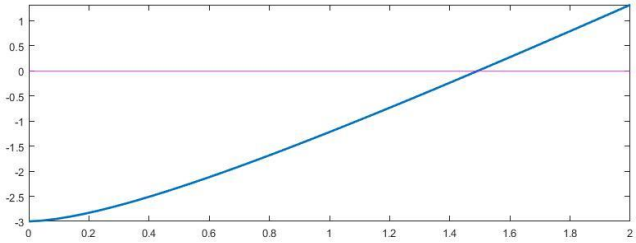
Az ábrából látszik, hogy a megoldás, *Q*=3, valahol 1.4 és 1.6 között lesz. A megoldás megtalálásához használható algoritmusok viszont mindig a zérushelyet/gyökhelyet

[1] 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:

$$Q(h) = 3 \quad \rightarrow \quad f(h) = Q(h) - 3 = 0$$

```
> f = @(h) Q(h)-3
> figure(2);
> fplot(f, [0 2],'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.

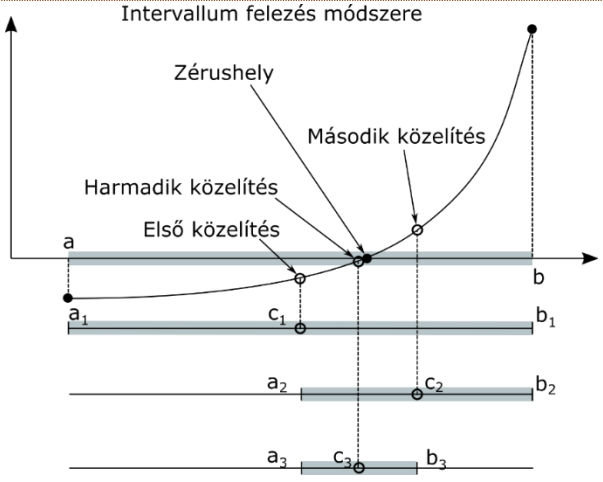
ZÁRT INTERVALLUM MÓDSZEREK

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)

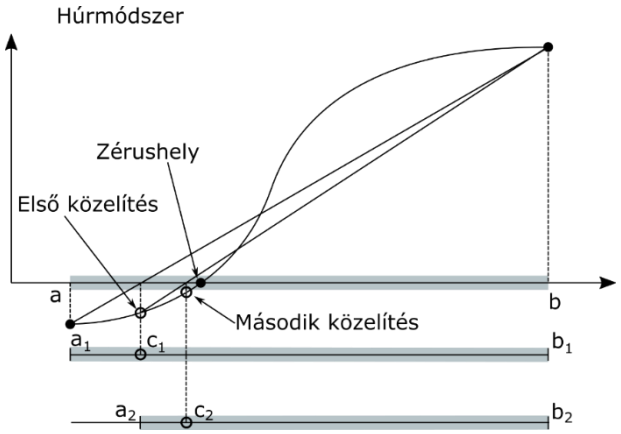
Az intervallumfelezés módszerénél hatékonyabb megoldás a húrmódszer (általában gyorsabban konvergál). Itt az intervallum végein kiszámolt f(a) és f(b) pontokat összekötő húrnak az x tengellyel való metszéspontját határozzuk meg mint új közelítő értéket. Az egyenes x tengellyel való metszéspontja (y=0),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

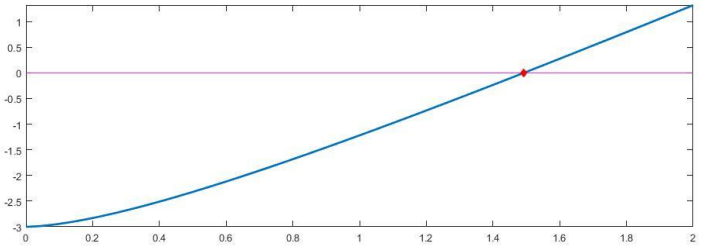
A húr módszer implementálása ugyanígy történhet, csupán a c kiszámításában van különbség az első és a további iterációkban, `c=(a+b)/2` helyett:

```
> 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');
```



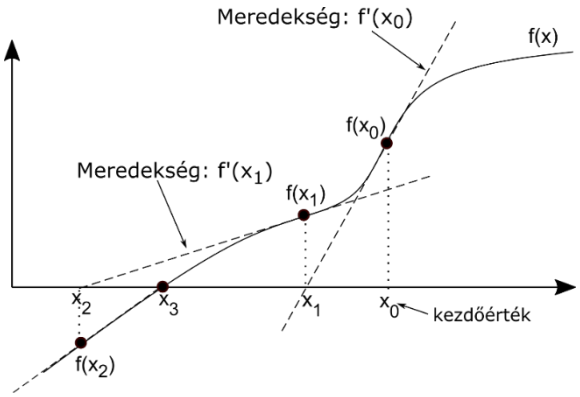
Értékeljük az eredményeket! A megoldások ugyanazok? Melyik volt a gyorsabb algoritmus? Melyik adta a pontosabb eredményt?

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.

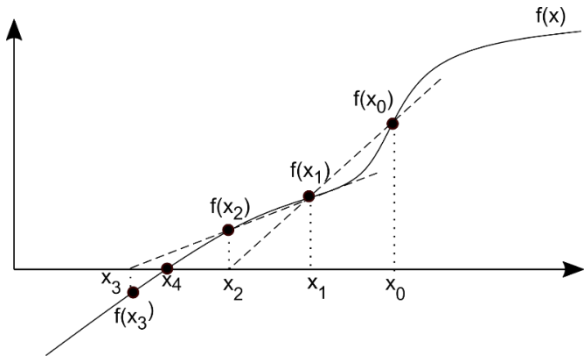
SZELŐ MÓDSZER (SECANT METHOD)[2]

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
```

[2] Otthoni átnézésre

```
>          x1 = x2;
>          x2 = x1 - f(x1)/df(x1);
>          i = i + 1;
>      end
>  end
```

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)[3]

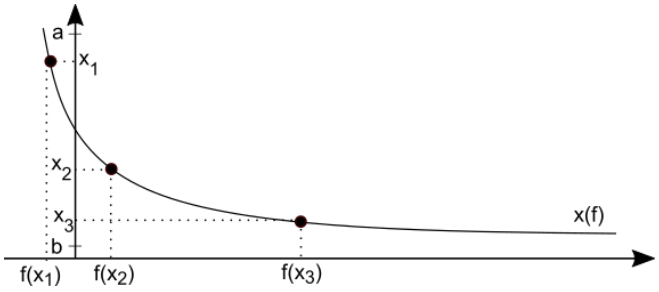
Brent-Dekker módszernek is nevezik, mivel Dekker korábbi módszerét fejlesztette tovább Brent. Hatékony és robosztus módszer, amely kombinálja az intervallum felezést, a húr módszert és az inverz kvadratikus interpolációt. Ezt használja a beépített **fzero** függvény is.

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
```

[3] Otthoni átnézésre

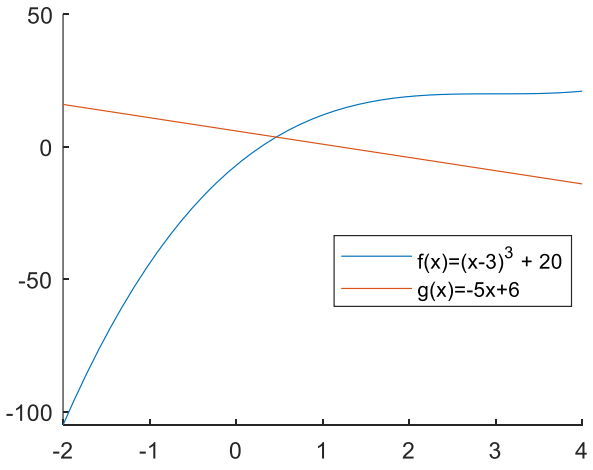
```
>  %     7         1.50949     0.0423222       1.69051      0.507665
   search
>  %     8           1.472    -0.0531432       1.69051      0.507665
   search
>  %
>  % Search for a zero in the interval [1.472, 1.69051]:
>  %  Func-count    x          f(x)             Procedure
>  %     8           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 FÜGGVÉNYEK METSZÉSPONTJA

Két egyváltozós függvény metszéspontjának megkeresését is visszavezethetjük egy egyváltozós függvény zérushelyeinek a megtalálására. Keressük meg a következő két függvény metszéspontját a [-2,4] intervallumon!

$$f(x) = (x-3)^3 + 20$$

$$g(x) = -5\,x + 6$$



A metszéspontban a két függvény egyenlő egymással:

$$f(x) = g(x)$$

Ezt rendezzük 0-ra:

$$f(x) - g(x) = 0$$

Rendeljünk egy új függvényt a 0-ra rendezett alakhoz:

$$h(x) = f(x) - g(x)$$

A fenti *h* függvény gyökeit pedig a korábbi módszerek valamelyikével meghatározhatjuk.

```
>  f = @(x) (x-3).^3 + 20
>  g = @(x) -5*x+6
>  figure(3); hold on;
>  fplot(f,[-2,4]);
>  fplot(g,[-2,4])
>  h = @(x) f(x)-g(x)
>  x = fzero(h,0.5)  % 0.4636
>  plot(x,f(x),'ko')
```

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:

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

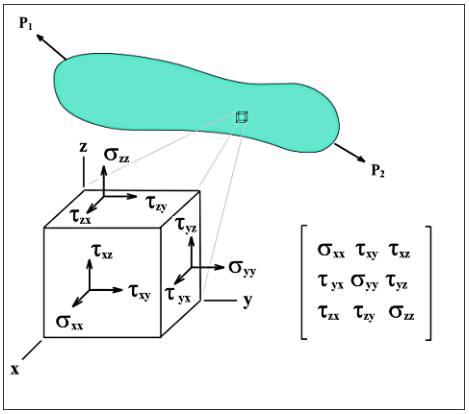
Az $a_n, a_{n-1}, \cdots, a_1, a_0$ együtthatók valós számok, n pedig egy nem negatív egész szám, a polinom fokszáma.

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[4]. 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: $(F - \sigma_e \cdot I) \cdot e = 0$

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.

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

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

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) szimbolikusan 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
```

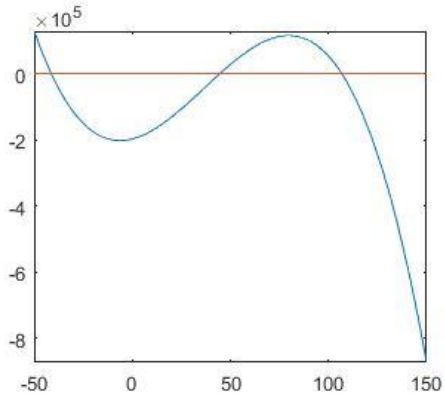
Ez egy harmadfokú algebrai polinom, aminek a gyökei adják a sajátértékeket:

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

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)))
```

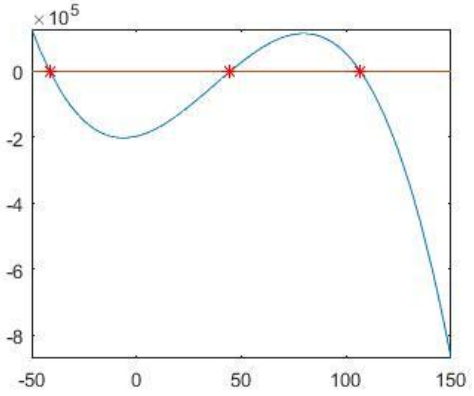
A másik algebrai polinomok esetében használható parancs a **roots**, ami numerikusan oldja meg az egyenletet. Ez is egy lépésben megadja az összes megoldást és itt sem kell kezdőértéket megadni. Itt viszont ki kell gyűjteni a polinom együtthatóit egy vektorba a legmagasabb fokú tagtól kezdve visszafelé 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