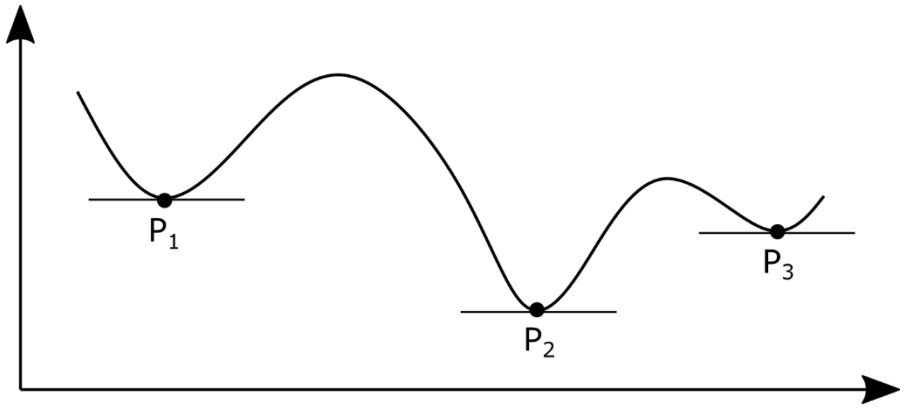
OPTIMALIZÁCIÓ

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).

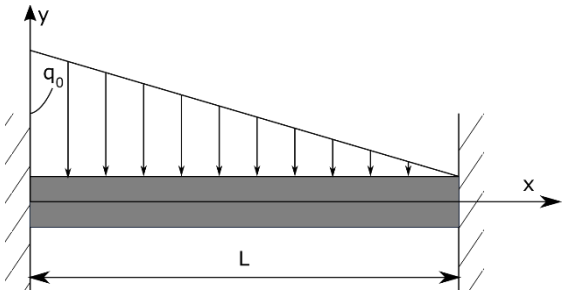


Beszélhetünk feltétel nélküli és feltételes szélsőértékről is (vagy megkötés nélküli és megkötéses optimalizációról). A feltételes szélsőérték feladatok esetében úgy keressük a függvény minimumát, hogy közben a pontoknak ki kell elégíteniük valamilyen megkötést, feltételt is. Itt lehet egy vagy több 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.

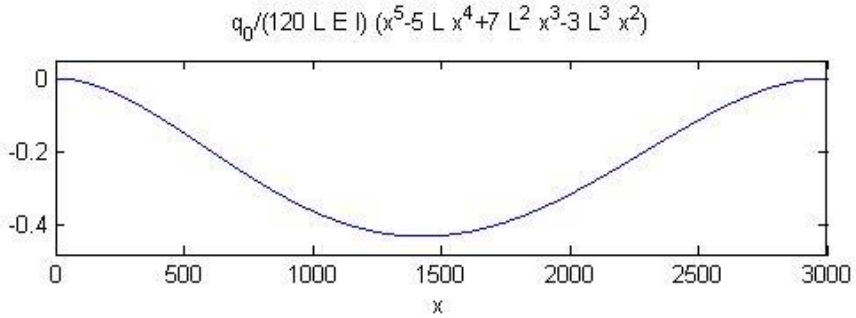
Mekkora lesz a lehajlás 1 és 2 m-nél? Keressük meg azt a helyet, ahol legnagyobb a lehajlás és határozzuk meg a lehajlás értékét!

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
```



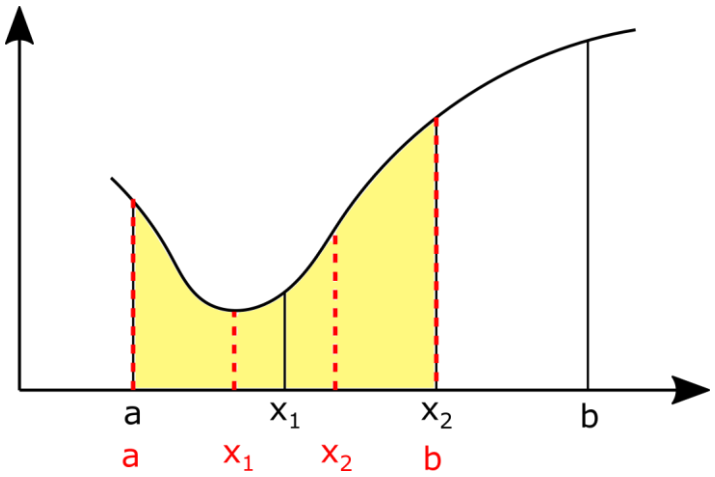
Nézzünk meg két módszert, amivel meg lehet keresni egy egyváltozós függvény minimumát!

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!

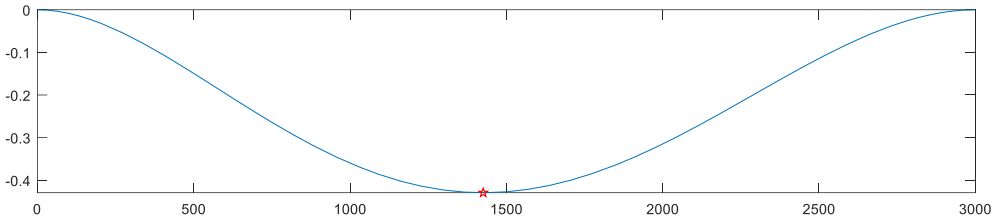
```
> function [x, i] = intervallum(f, a, b, tol)
> i = 1;
>  x1 = a + 1/3*(b-a);
>  x2 = b - 1/3*(b-a);
>
>  while abs(x2-x1) > tol
>    if f(x1) < f(x2)
>      b = x2;
>    else
>      a = x1;
>    end
>    i = i+1;
>    x1 = a + 1/3*(b-a);
>    x2 = a - 1/3*(b-a);
>  end
>  x = (x1+x2)/2;
> end
```

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:

$$L = L \cdot R^2 + R \cdot L$$

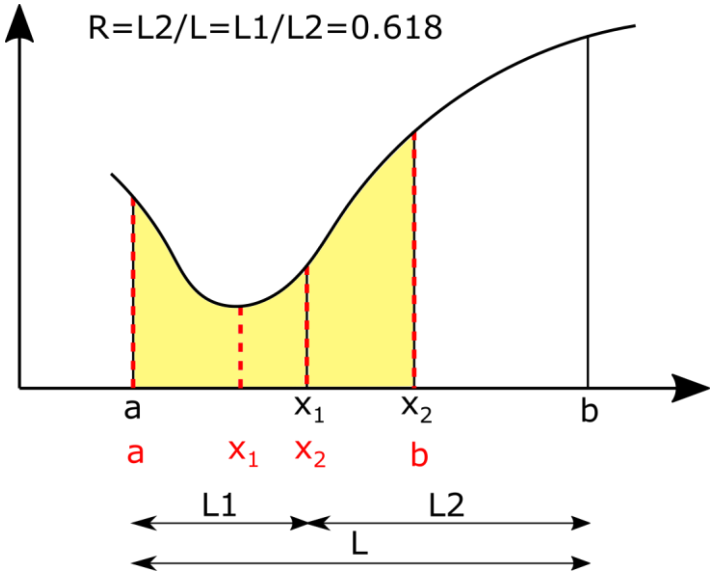
Ezt elosztva L-lel és 0-ra rendezve kapjuk, hogy:

$$R^2 + R - 1 = 0$$

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

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:

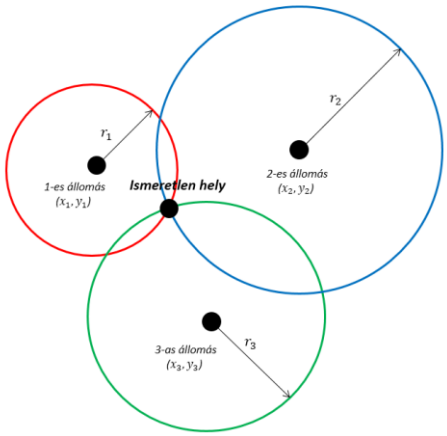
```
> [x,fval,exitflag,output] = fminsearch(y,2000)
> i = output.iterations % i = 25 - iterációk száma
> n = output.funcCount % n = 50 - függvény kiértékelések száma
```

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!

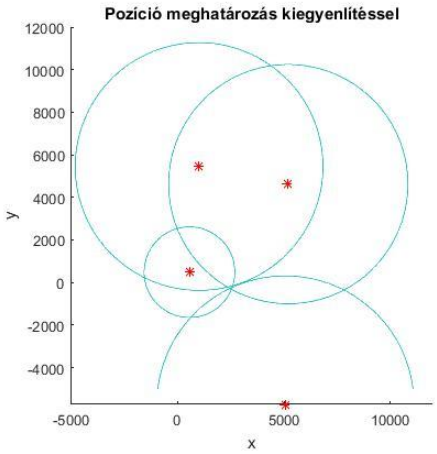
```
clear all; clc; close all;
xt = [561; 5203; 5067; 1012]
yt = [487; 4625; -5728; 5451]
rm = [2130; 5620; 6040; 5820]
% Körök középpontjai
figure(1); hold on;
plot(xt, yt, 'r*')
```

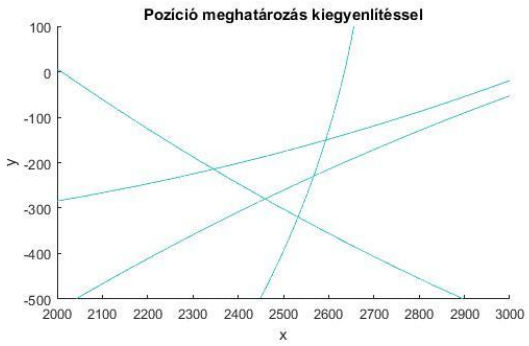
Ezután definiáljuk a kör egyenletét általánosan, és szimbolikus x,y változót használva ábrázoljuk az egyes köröket!

```
% 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')
```



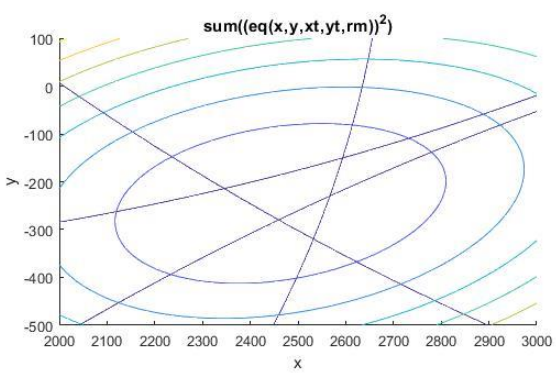


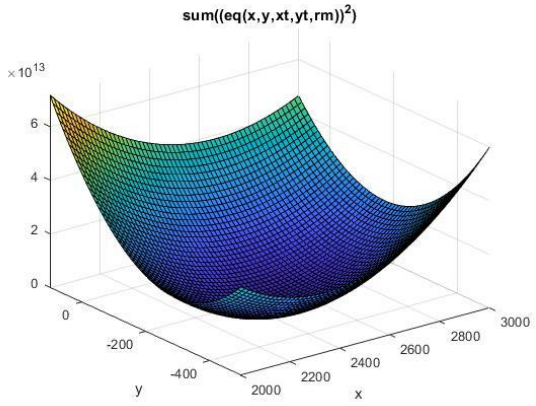
Látjuk, hogy a négy kör nem egy pontban metszi egymást, hanem közrezárnak egy területet, ahol a feltételezett pozíciónk van. Ezt a legvalószínűbb helyzetet határozhatjuk meg a maradék eltérések négyzetösszegének minimalizálásával. Írjuk fel a minimalizálandó függvényt (*f*), ami a maradék eltérések négyzetösszege 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;
```

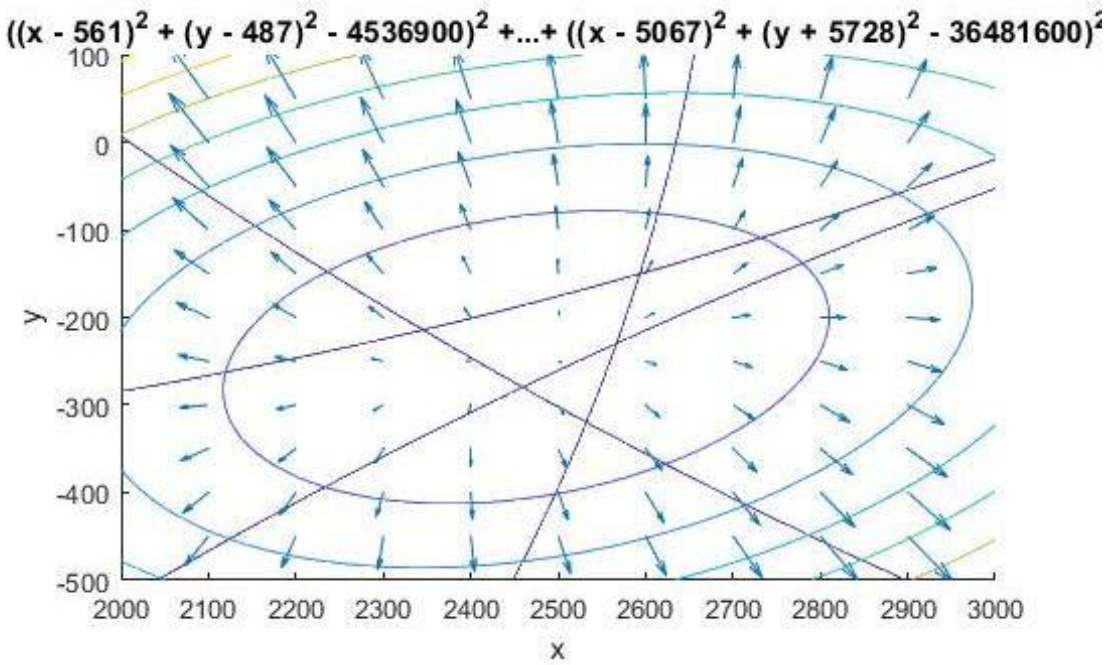
A fenti függvény (gradmulti.m) a többváltozós Newton-módszer megoldása Matlab-ban, ahol a bemenet a gradiens vektor, a Hesse mátrix, egy x0 kezdeti pozíció, eps tolerancia érték a leállási feltételhez és egy nmax maximális iteráció szám.

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)
```

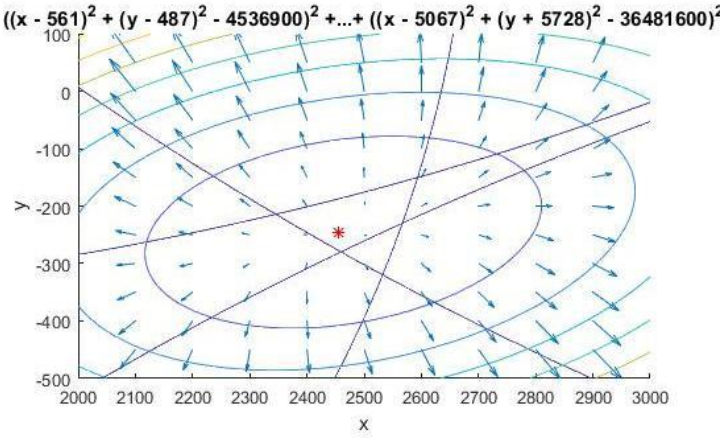


A többváltozós Newton-módszer alkalmazásához nem numerikusan van szükségünk a gradiens vektorra és Hesse mátrixra, hanem vektorváltozós függvény formájában. Ezt meghatározhatjuk szimbolikus számításokkal a **gradient** és a **hessian** parancsok alkalmazásával a szimbolikus *f* függvényre!

```
> % 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

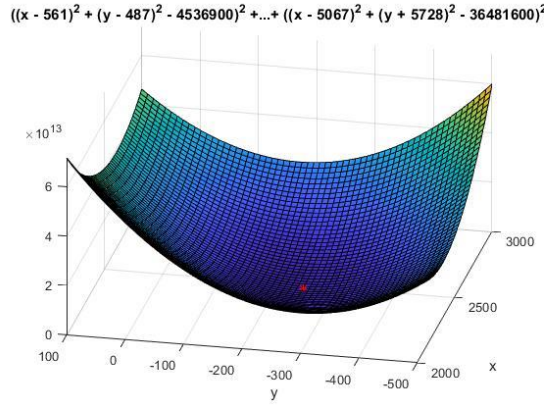
módszert használja, egy másik pedig az **fminunc**, ami kvázi-Newton minimalizálást alkalmaz. Ha a deriváltak nem számíthatóak könnyen, akkor célszerű a szimplex módszert használni. A módszer során felveszünk egy kezdő poliédert (szimplexet), 2 dimenziós esetben egy háromszöget, majd az eljárás során 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
```



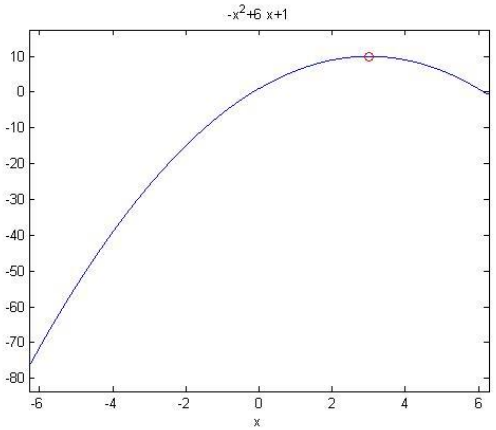
Ábrázoljuk a megoldást a 3D ábrán is!

```
> 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[1]

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!

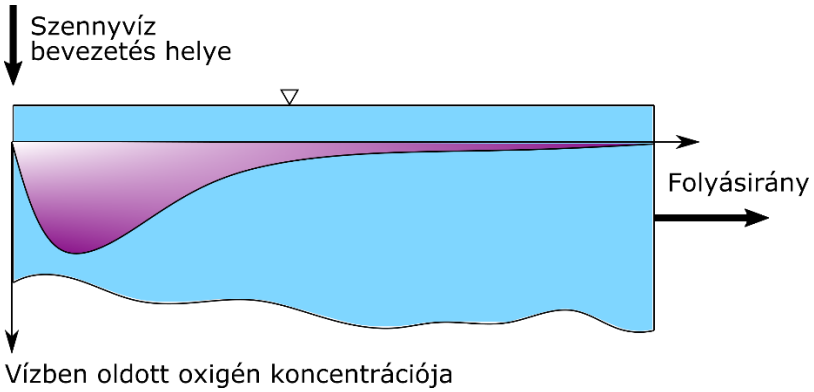



```
> clear all; clc; close all;
> f = @(x) -x.
> ^2 + 6*x + 1
> fplot(f)
> fm = @(x) -f(x)
> xm = fminsearch(fm,4) %  3.0000
> fmax = f(xm) %  10.0000
> hold on; plot(xm, fmax, 'ro')
```

Ne felejtsük el, hogy a maximum értékének megállapításához az eredeti függvénybe kell visszahelyettesíteni!

GYAKORLÓ FELADAT[2]

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!



[1] Otthoni átnézésre

[2] Otthoni átnézésre

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]:

$$c(t) = c_s - \frac{k_d L_0}{k_d + k_s - k_a}\left(e^{-k_a t} - e^{-(k_d+k_s)t}\right) - \frac{S_b}{k_a}(1 - e^{-k_a t})$$

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!

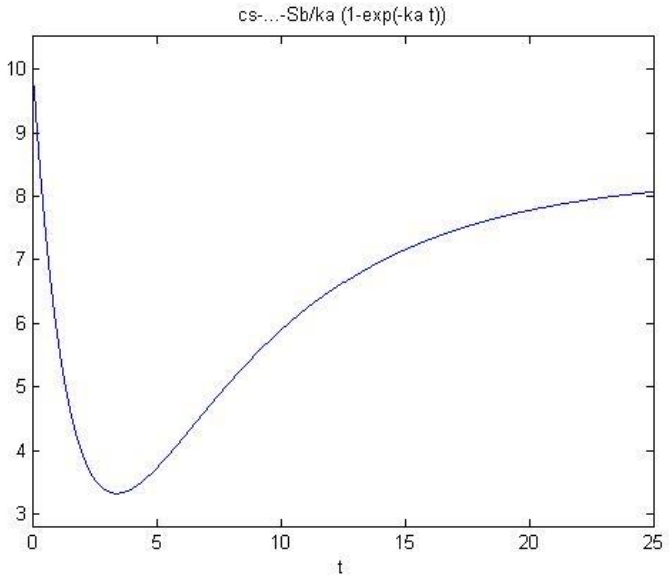
```
> % szennyvíz bevezetés
> clear all; clc; close all;
> cs = 10; L0 = 50; kd = 0.1; ks = 0.05; ka = 0.6; Sb = 1;
>
> c = @(t) cs - kd*L0/(kd+ks-ka)*(exp(-ka*t)-exp(-(kd+ks)*t)) -
  Sb/ka*(1-exp(-ka*t))
```

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)
> 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!

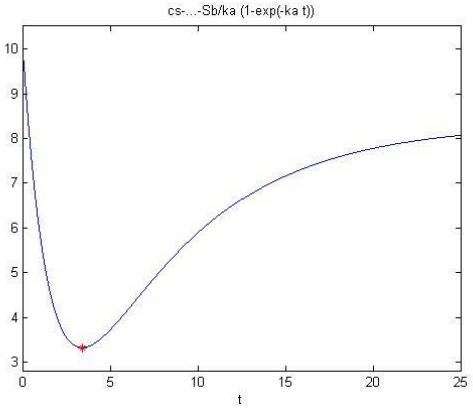
```
> % intervallum módszer - egyenlő felosztás
> [x1 i1] = intervallum(c,0,5,1e-6)
```

```
> % 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?

ÚJ FÜGGVÉNYEK A GYAKORLATON

ÚJ FÜGGVÉNYEK AZ OPTIMALIZÁCIÓ ÓRÁN (12. GYAKORLAT)

subs	-	szimbolikus változóba konkrét értékek behelyettesítése
diff(f,x,2)	-	f szimbolikus kifejezés 2. deriváltja x szerint
fminsearch	-	Egy/többváltozós függvény minimának megkeresése Nelder-Mead szimplex módszert alkalmazva
fminunc	-	Feltétel nélküli szélsőérték keresés kvázi-Newton minimalizálást alkalmazva.
gradient	-	Gradiensek számítása numerikusan, szimbolikusan
quiver	-	vektormező megjelenítése
hessian	-	Hesse-mátrix, az f(x) függvény második parciális deriváltjainak a mátrixa