15. 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.

Optimalizáció során van egy célfüggvény $f(\boldsymbol{x})$, ahol $\boldsymbol{x}$ a bemenő, független változók vektora: $\boldsymbol{x} = [x_1, x_2, \cdots x_n]$. Az optimalizálás során azt a helyet keressük, ahol a célfüggvénynek minimuma vagy maximuma van. Sokféle módszer létezik a feladat megoldására, a legtöbb módszert a függvény minimalizálására dolgozták ki.

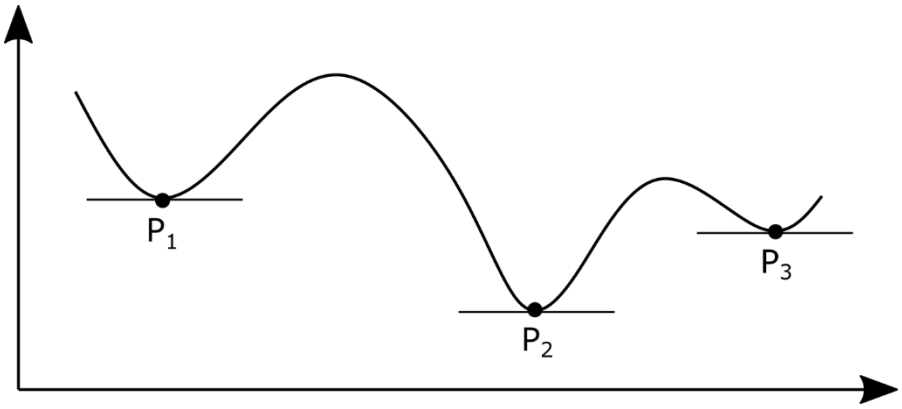
$$f(x) \rightarrow min.$$

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/maximumok vagy
- egy globális minimum/maximum.

Lokális minimumok azok a helyek, 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).

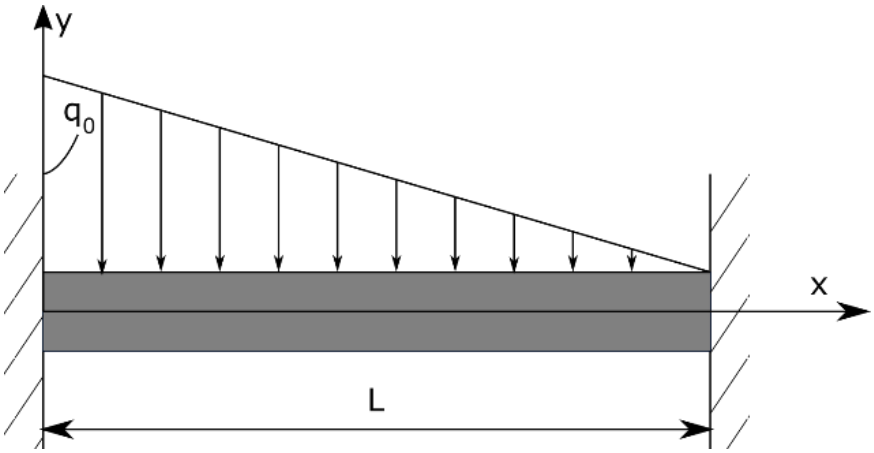


Beszélhetünk feltétel nélküli és feltételes szélsőértékről is (vagy megkötés nélküli és megkötéses optimalizációról). A feltételes szélsőérték feladatok esetében úgy keressük a függvény minimumát, hogy közben a pontoknak ki kell elégíteniük valamilyen megkötést, feltételt is. Itt lehet egy vagy több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

EGYVÁLTOZÓS FÜGGVÉNY SZÉLSŐÉRTÉK KERESÉSE

ÉPÍTŐMÉRNÖKI PÉLDA SZÉLSŐÉRTÉK KERESÉSRE

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.



A gerenda méretei és terhei a következők:

- Gerenda hossza: $L = 3000\ mm$
- Tehetetlenségi nyomaték: $I = 5.29 \cdot 10^7\ mm^4$
- Rugalmassági modulus: $E = 70\,000\ N/mm^2$
- Megoszló terhelés maximális értéke: $q_0 = 15\ kN/m\ = 15\ N/mm$

Az y tengely irányú lehajlás a következő összefüggéssel számítható:

$$y = \frac{q_0}{120LEI}(x^5 - 5Lx^4 + 7L^2x^3 - 3L^3x^2)$$

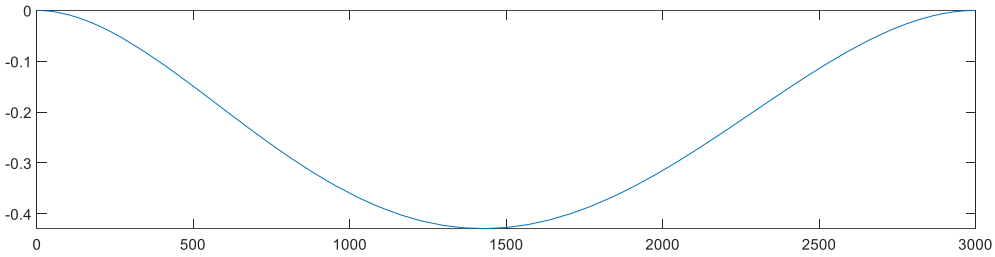
1) Mekkora lesz a lehajlás 1 és 2 m-nél?
2) Hol lesz a lehajlás pont 0.3 mm?
3) Keressük meg azt a helyet, ahol legnagyobb a lehajlás és határozzuk meg a lehajlás értékét!

Először ábrázoljuk a lehajlásokat!

```
> %% Lehajlás számítás
> format longG
> E = 70000; I = 5.29e7; q0 = 15; L = 3000; EI = E*I;
```

Célszerű összevonni az E és I változókat egybe a hajlítómerevségbe (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])
```



1) Mekkora lesz a lehajlás 1 és 2 m-nél?

Az első kérdésre a válasz egy egyszerű behelyettesítéssel megadható. Figyeljünk arra, hogy korábban mindent miliméterben definiáltunk, akkor itt is mm-ben kell behelyettesíteni a számokat!

```
> y(1000), y(2000)  % lehajlás 1 ill. 2 m-nél: -0.3601 és -0.3151 mm
```

2) Hol lesz a lehajlás pont 0.3 mm?

Ezt a kérdést már nem tudjuk egyszerű behelyettesítéssel megoldani, a következő nemlineáris egyenletet kell hozzá megoldanunk:

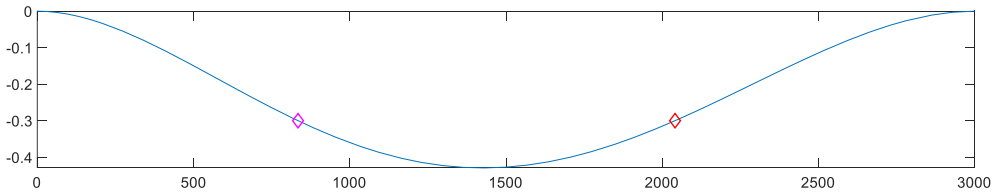
$$\frac{q_0}{120LEI}(x^5 - 5Lx^4 + 7L^2x^3 - 3L^3x^2) = -0.3$$

A lehajlás értékek negítív koordinátákat takarnak! Rendezzük nullára az egyenletet és nevezzük el a kapott egyenletet más néven:

$$h(x) = \frac{q_0}{120LEI}(x^5 - 5Lx^4 + 7L^2x^3 - 3L^3x^2) + 0.3 = 0$$

Ezt már gyökkereséssel meg tudjuk oldani! Két hely is lesz, ahol az y koordináta -0.3 mm, ezekhez kezdőértéket az ábrából tudunk venni.

```
> % hol lesz a lehajlás pont 0.3 mm?
> h = @(x) y(x)+0.3
> h1 = fzero(h,1000) % 834.388
> h2 = fzero(h,2000) % 2040.970
> % ellenőrzés
> y(h1) % -0.3
> y(h2) % -0.3
> hold on; plot(h1,y(h1),'md',h2,y(h2),'rd')
```



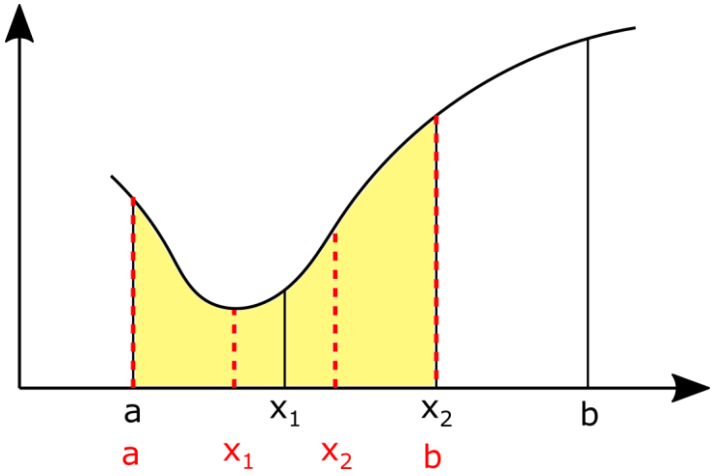
Mielőtt a harmadik kérdésre válaszolnánk, nézzünk meg két módszert, amivel meg lehet keresni egy egyváltozós függvény minimumát!

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 fájlt!

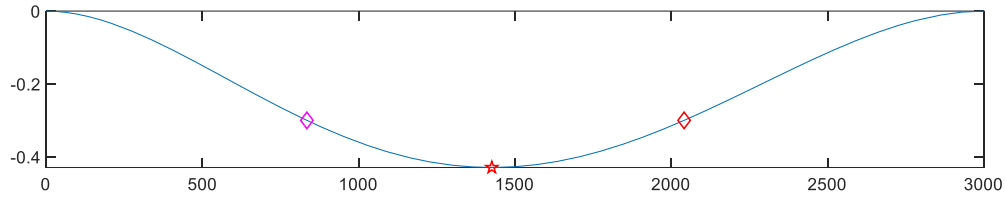
```
> function [x, i] = intervallum(f, a, b, tol)
> i = 1;
>  x1 = a + 1/3*(b-a);
>  x2 = b - 1/3*(b-a);
>
>  while abs(x2-x1) > tol
>    if f(x1) < f(x2)
>      b = x2;
>    else
>      a = x1;
>    end
>    i = i+1;
>    x1 = a + 1/3*(b-a);
>    x2 = b - 1/3*(b-a);
>  end
>  x = (x1+x2)/2;
> end
```

3) Keressük meg azt a helyet, ahol legnagyobb a lehajlás és határozzuk meg a lehajlás értékét!

Keressük meg az intervallum 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)[1]

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:

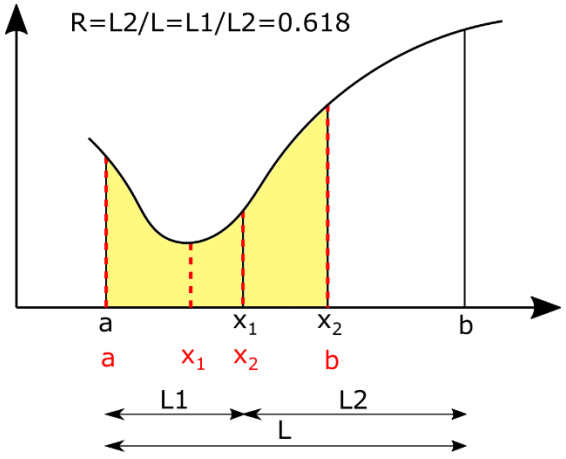
$$R^2 + R - 1 = 0$$

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:

$$R = \frac{\sqrt{5} - 1}{2} = 0.618$$

[1] Kiegészítő anyag otthoni átnézésre.

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)!

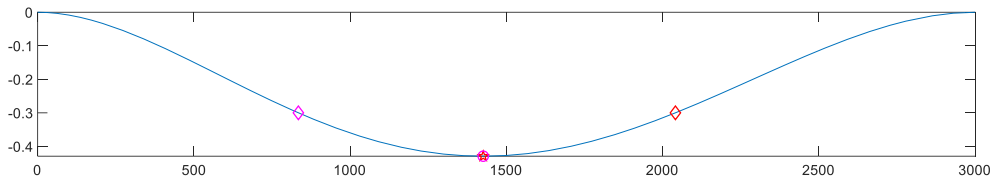
```
> function [x, i] = aranymetszes(f, a, b, tol)
> i = 1;
> R = (sqrt(5)-1)/2;
> x1 = b - R*(b-a);
> x2 = a + R*(b-a);
> f1 = f(x1); f2 = f(x2);
>
> while abs(x2-x1)>tol
>   if f1 < f2
>      b = x2;
>      x2 = x1; f2 = f1; % ezt átvesszük az előző iterációból!
>      x1 = b - R*(b-a);
>      f1 = 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).

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

A Newton módszer iterációs képlete zérushely kereséshez, az $f(x) = 0$ egyenlet megoldásakor:

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

Ugyanez szélsőérték kereséséhez, az $f'(x) = 0$ egyenlet megoldásakor $f(x)$ -et $f'(x)$-re cserélve és $f'(x)$-et $f''(x)$-re cserélve:

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

A Newton módszer előnye, hogy mind minimum, mind maximum számításra használható. Hátránya, hogy 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)
```

A későbbi használathoz szimbolikus kifejezések helyett függvényekre lesz szükségünk! Használjuk a **matlabFunction** parancsot ehhez!

```
> % Alakítsuk függvénnyé a dx szimbolikus kifejezést!
> dxf = matlabFunction(dx)
> % @(EI,L,q0,x)(q0.*(L.*x.^3.*2.0e+1+L.^3.*x.*6.0-x.^4.*5.0-
  L.^2.*x.^2.*2.1e+1).*(-1.0./1.2e+2))./(EI.*L)
```

Az eredményben azt látjuk, hogy egy négyváltozós (*EI,L,*q_0*,x*) függvény jött létre! Ez annak köszönhető, hogy *EI,L,*q_0 változókat felülírtuk, amikor szimbolikusként definiáltuk őket. Nekünk azonban egy egyváltozós függvényre van szükségünk. Ehhez meg kell adnunk újra az *EI, L* és q_0 változók értékeit, majd a kapott szimbolikus eredményeket egyszerűen bemásolhatjuk 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)
```

Illetve kis változtatással itt is használhatjuk a **matlabFunction** parancsot. Ehhez előtte újra definiálni kell *EI,L,*q_0 változókat, majd ezek értékeit behelyettesíteni a szimbolikus kifejezésekbe a **subs** paranccsal. A **subs** parancs segítségével be lehet helyettesíteni egyszerre az összes korábban számként megadott változót, vagy meg lehet adni tetszőleges változó értékét is.

```
> % matlabFunction használata behelyettesítéssel
> E = 70000; I = 5.29e7; q0 = 15; L = 3000; EI = E*I;
> 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,FMINBND)

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
> E = 70000; I = 5.29e7; q0 = 15; L = 3000; EI = E*I;
> 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
```

Meghívhatjuk olyan módon is az **fminsearch** függvényt, hogy ne csak a minimum helyét adja vissza, hanem az értékét is és egyéb részleteket is.

```
> [x,fval,exitflag,output] = fminsearch(y,2000)
> % x = 1425.9; fval = -0.42935
> i = output.iterations % i = 25 - iterációk száma
> n = output.funcCount % n = 50 - függvény kiértékelések száma
```

A Matlabnak van zárt intervallum megadását igénylő egyváltozós optimalizációs parancsa is, az **fminbnd**. A megadása egy [a,b] intervallummal: **fminbnd**(függvény,a,b).

KÉTVÁLTOZÓS 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.

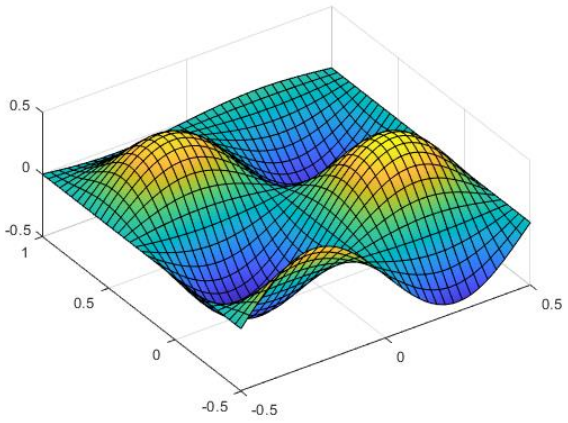
Nézzük először egy függvénnyel megadott felület szélsőértékeinek a meghatározását! A felületet a következő függvénnyel definiálhatjuk:

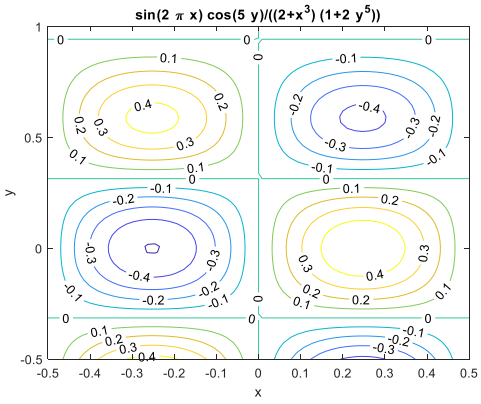
$$f(x,y) = \frac{\sin(2 \cdot \pi \cdot x) \cdot \cos(5 \cdot y)}{(2 + x^3) \cdot (1 + 2 \cdot y^5)}$$

A tartomány, ahol a szélsőértékeket keressük: $-0.5 \leq x \leq 0.5; -0.5 \leq y \leq 1.$

Definiáljuk a felületet és ábrázoljuk térbeli ábrán és szintvonalakkal is, a szintvonalakat feliratozva!

```
> %% Kétváltozós függvény lokális szélsőértékeinek meghatározása
> clc; clear all; close all; format shortG;
> f = @(x,y) sin(2*pi*x).*cos(5*y)./((2+x.^3).*(1+2*y.^5))
> figure(1); fsurf(f,[-0.5 0.5 -0.5 1])
> figure(2); h1 = ezcontour(f,[-0.5 0.5 -0.5 1])
> set(h1,'ShowText','on')
```





Látjuk, hogy a megadott tartományon több lokális minimum és maximum is található. Hogyan tudjuk ezeket megtalálni? Többféle módszert is használhatunk például a Matlab egyik beépített függvénye (fminsearch) által alkalmazott Nelder-Mead módszert, vagy a többváltozós Newton-módszert, ami az egyváltozós esetben alkalmazott módszer általánosítása.

NELDER-MEAD SZIMPLEX MÓDSZER

A Nelder-Mead szimplex módszert eredetileg 1965-ben publikálták (Nelder and Mead, 1965[2]). Ez az egyik legismertebb módszer a többváltozós optimalizációra derivált használata nélkül, egy ún. direkt kereső eljárás (’direct search method’). Ez egy heurisztikus algoritmus, ami egyszerű és könnyen alkalmazható.

Az algoritmus egy szimplexen alapul, ami egy *n* dimenziós térben egy *n+1* csúcsból álló geometriai alakzat, 2D-ben egy háromszög, 3D-ben egy tetraéder. Az eljárásban nincs szükség gradiensek számítására, csak a csúcsok értékének meghatározására. Maga az eljárás különböző geometriai transzformációkat hajt végre a szimplexen, pl. tükrözés, nyújtás, zsugorítás, tágítás, összehúzás, annak érdekében, hogy csökkentse a függvényértékeket a csúcsokban.

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.

[2] J. A. Nelder, R. Mead, A Simplex Method for Function Minimization, The Computer Journal, Volume 7, Issue 4, January 1965, Pages 308–313, https://doi.org/10.1093/comjnl/7.4.308

MEGOLDÁS NELDER-MEAD SZMPLEX MÓDSZERREL MATLAB-BAN

Határozzuk meg a megadott felület lokális szélsőérték helyeit a Nelder-Mead szimplex módszerrel! A Matlab beépített **fminsearch** eljárása az optimalizációhoz ezt az algoritmust használja. Figyeljünk arra, hogy ez a függvény csak minimumkeresésre használható, maximumkeresésre nem (mint az a neve alapján is várható).

Nagyon fontos, hogy a többi Matlab beépített függvényhez hasonlóan az **fminsearch** is vektorváltozós függvényekkel tud csak dolgozni, ezért első lépésben vektorizálni kell a függvényt. A szintvonalas ábrán látszódik, hogy 2 minimum és két maximum található a vizsgált tartományon. Az ábrából könnyen tudunk kezdőértéket választani hozzájuk. A kezdőértékeket oszlopvektorban adjuk meg!

```
> F = @(v) f(v(1),v(2)); % vektorváltozóssá alakítás
> % kezdőértékek a minimumhelyekhez:
> x01 = [-0.2;0]; x02 = [0.2;0.6];
> % kezdőértékek a maximum helyekhez
> x03 = [0.2;0]; x04 = [-0.2;0.6];
```

Először határozzuk meg a 2 minimum helyét és értékét és ábrázoljuk őket a szintvonalas ábrán is!

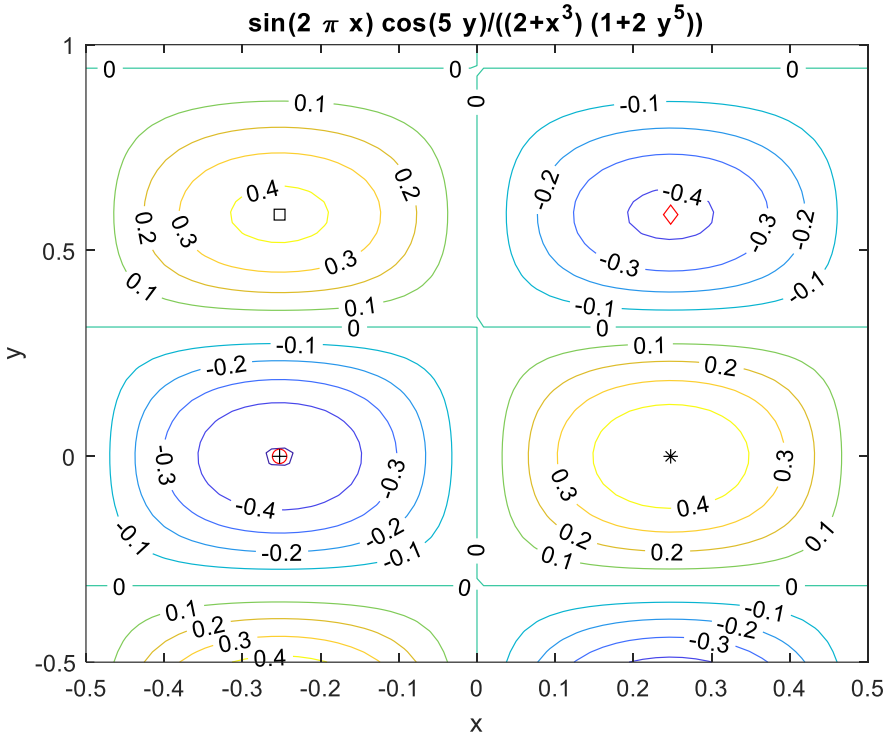
```
> % Minimumhelyek meghatározása
> [sol1 f1] = fminsearch(F,x01) %  -0.25241   8.9111e-06;   -0.504
> [sol2 f2] = fminsearch(F,x02) %  0.24767     0.58715;   -0.42622
> hold on; plot(sol1(1),sol1(2),'ro'); plot(sol2(1),sol2(2),'rd')
```

Majd keressük meg a maximum pontok helyét és értékét is! Először nézzük meg, mi történik, ha a maximum pont közelében adunk egy kezdőértéket az optimalizációs algoritmusunknak!

```
> % Maximumhelyek meghatrozása
> [sol3 f3] = fminsearch(F,x03) %  -0.25241   1.7236e-05; -0.504
> plot(sol3(1),sol3(2),'k+');
```

A kapott érték megegyezik az első megoldásunkkal, ami egy minimumhely volt. Az **fminsearch** csak minimumhely megkeresésére használható, amennyiben maximumot szeretnénk vele megkapni, akkor a függvény definíciót kell módosítanunk, és a függvény -1 szeresét kell definiálnunk. Itt a maximumkeresés problémája minimumkereséssé alakul át. Természetesen, ha a függvény értékét szeretnénk visszakapni, akkor az megtalált maximum helyet az eredeti függvénybe kell visszahelyettesíteni.

```
> Fmax = @(v) -1*F(v) % Függvény -1 szeresének definiálása
> [sol3 f3] = fminsearch(Fmax,x03) %  0.24765  -2.3376e-05; -0.49618
> plot(sol3(1),sol3(2),'k*');
> [sol4 f4] = fminsearch(Fmax,x04) %  0.24765  -2.3376e-05; -0.49618
> plot(sol4(1),sol4(2),'ks');
> % Maximum értékek az eredeti függvénybe visszahelyettesítve
> f3 = F(sol3) % 0.49618
> f4 = F(sol4) % 0.43293
```



TÖBBVÁLTOZÓS NEWTON-MÓDSZER[3]

Természetesen sok más módszer is alkalmazható optimalizációra. Egy másik módszer lehet a többváltozós Newton-módszer, amit az egyváltozós esetből könnyen lehet általánosítani. A Newton módszer előnye, hogy a szélsőérték keresés során minimum és maximum hely is meghatározható vele, hiszen azokat a helyeket keresi meg, ahol a derivált értéke nulla, vagyis vízszintes lesz az érintő. Hátránya is pont ebben van, hogy meg kell határozni hozzá az első és a második deriváltakat is, ami számításigényes.

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:

[3] Otthoni átnézésre.

$$\nabla f(x,y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}; \quad H(x,y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x\,\partial y} \\ \frac{\partial^2 f}{\partial y\,\partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

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.

A lenti függvény (gradmulti.m) a többváltozós Newton-módszer megoldása Matlab-ban, ahol a bemenet a gradiens vektor - *grad*, a Hesse mátrix - *hesse*, egy *x0* kezdeti pozíció, *eps* tolerancia érték a leállási feltételhez és egy *nmax* maximális iteráció szám. Vegyük észre, hogy az eredeti F függvény nem kell a megoldáshoz, csak a Hesse mátrix és a gradiens vektor.

A kimenetben x1 a megoldás, i az iteráció szám, X pedig az egymást követő megoldások lépéseit tartalmazza.

```
> 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 Matlab beépített függvényei között is találunk olyat, ami gradiens alapú optimalizálást végez, ilyen az **fminunc** függvény, ami kvázi-Newton minimalizálást alkalmaz. Az **fminunc** függvény esetében a Newton módszer használatára is lehetőség van, amennyiben megadjuk opcionális bemenetként a gradiens vektort és a Hesse mátrixot. Ennek a pontos megadási módját a Matlab dokumentációban érdemes megnézni.

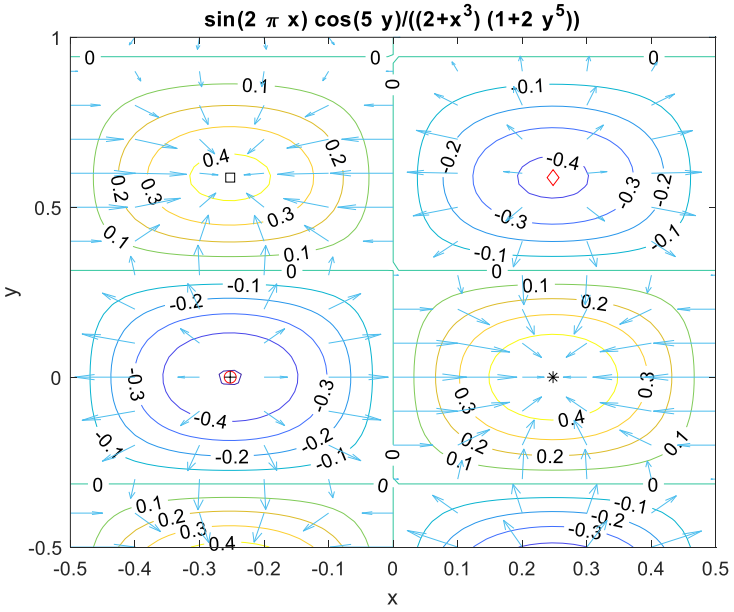
MEGOLDÁS TÖBBVÁLTOZÓS NEWTON-MÓDSZERREL[4]

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! (lásd részletesen

[4] Kiegészítő anyag otthoni átnézésre.

a numerikus deriválás gyakorlat anyagát!) Ezt csak szemléltetés kedvéért mutatjuk be, a megoldáshoz nincs szükség a numerikus gradiensekre.

```
> % Gradiens vektor ábrázolása numerikusan
> [X,Y] = meshgrid(-0.5:0.1:0.5, -0.5:0.1:1);
> Z = f(X,Y); % függvény értékek a rácspontokban
> [px,py] = gradient(Z); % gradiensek számítása numerikusan
> quiver(X,Y,px,py)
```



A többváltozós Newton-módszer alkalmazásához nem numerikus mátrix formában 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!

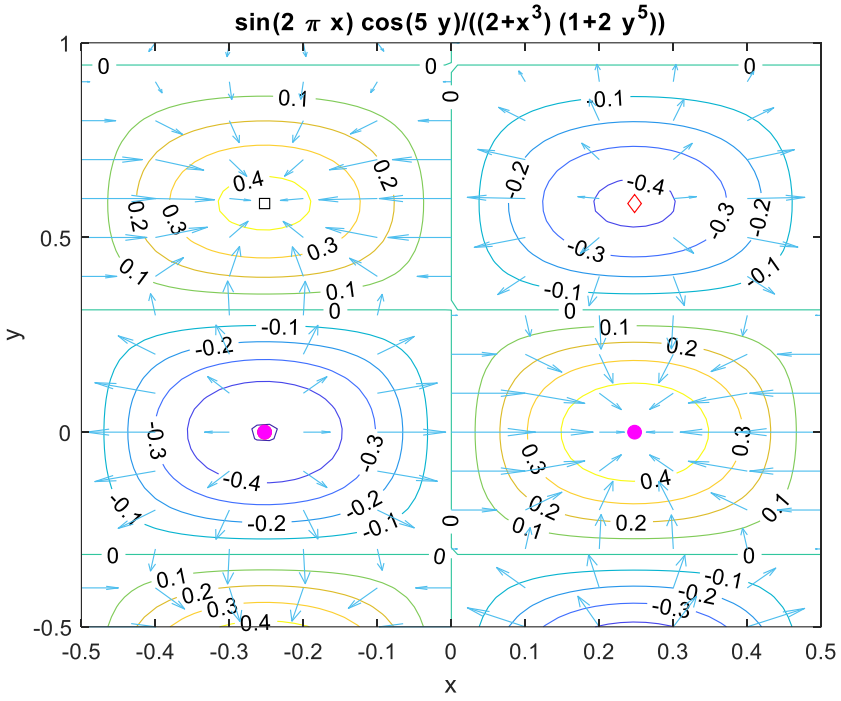
```
> % Függvények szimbolikusan
> syms x y;
> fs = f(x,y) % (cos(5*y)*sin(2*pi*x))/((x^3 + 2)*(2*y^5 + 1))
> G = gradient(fs) % Gradiens vektor szimbolikusan
> % (2*pi*cos(5*y)*cos(2*pi*x))/((x^3 + 2)*(2*y^5 + 1)) -
  (3*x^2*cos(5*y)*sin(2*pi*x))/((x^3 + 2)^2*(2*y^5 + 1))
> % - (5*sin(5*y)*sin(2*pi*x))/((x^3 + 2)*(2*y^5 + 1)) -
  (10*y^4*cos(5*y)*sin(2*pi*x))/((x^3 + 2)*(2*y^5 + 1)^2)
> H = hessian(fs) % Hesse mátrix szimbolikusan
> % 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 módszerrel, akár minimum, akár maximum helyet meg tudunk határozni. Használjuk a korábban megadott kezdőértékeket egy minimu és egy maximum hely megtalálásához!

```
> % Minimum és maximum hely keresése
> [min1 i M1] = gradmulti(G,H,x01,1e-6,100) % az egyik minimumhely
> [max1 i M2] = gradmulti(G,H,x03,1e-6,100) % az egyik maximumhely
> plot([min1(1);max1(1)],[min1(2);max1(2)],'mo','MarkerFaceColor','m')
> F(min1) % -0.504
```

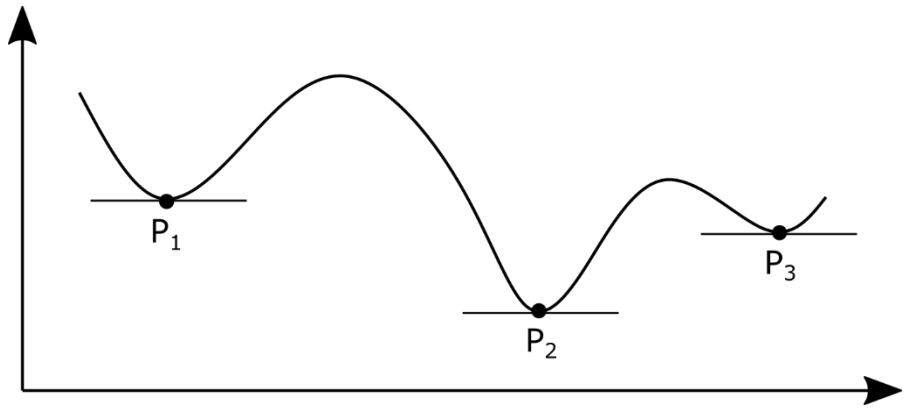
```
> F(max1) % 0.49618
```

Nagyon gyorsan konvergált a módszer, 4 iterációból eljutottunk a minimumhelyre a kívánt pontosságon belül.



GLOBÁLIS OPTIMALIZÁCIÓ

Mint láttuk, egy adott tartományban több lokális minimum is létezhet. Ezek közül a legkisebb lesz a globális minimum (az ábrán P_2). Az eddig ismertetett módszerek, mint az intervallum módszer, Newton módszer, vagy a Matlab által alkalmazott Nelder-Mead szimplex módszer egy kezdeti érték megadását igénylik és általában „elakadnak” a legközelebbi lokális minimumnál. Több minimum esetén több kezdőértékkel kell meghívni őket, és utána ezeket összehasonlítva tudjuk eldönteni, hogy melyik a globális minimum. Használhatunk azonba olyan heurisztikus megközelítést amelyekkel a globális minimumot tudjuk megközelíteni, kezdőérték megadása nélkül, ilyen például a genetikus algoritmus, amellyel a következőkben fogunk foglalkozni.



GLOBÁLIS OPTIMALIZÁCIÓ GENETIKUS ALGORITMUSSAL[5]

Legyen a feladat egy *f(x)* egyváltozós függvény globális minumának meghatározása egy adott $x \in [a, b]$ intervallumban, azaz

$$\min_{x \in [a,b]} f(x)$$

A genetikus algoritmusok populáció alapú speciális evolúciós algoritmusok, olyan heurisztikus keresési technikák, melyekkel optimumot lehet keresni. Technikáikat, szakkifejezéseiket az evolúcióbiológiából kölcsönözték.

A genetikus algoritmusokra általában jellemző, hogy az optimalizálandó célfüggvénynek viszonylag kevés megkötésnek kell megfelelnie. Például a hagyományos eljárásokkal ellentétben nem kell zárt formában megfogalmazhatónak, lineárisnak, vagy deriválhatónak lennie. Az egyetlen feltételezés, amivel ezek az eljárások élnek, az, hogy kis változtatás a függvény paramétereinek értékében kis változást kell hogy eredményezzen a függvény értékében (tehát a függvény „felülete" ne legyen teljesen véletlenszerű).

A célfüggvényt, amelynek a minimumhelyét keressük, hívhatjuk költségfüggvénynek vagy energiafüggvénynek is. Egy másik lehetséges elnevezés a rátermettségi, fitnesz (angolul „fitness") függvény.

A genetikus algoritmusokat számítógépes szimulációkkal implementálják. A keresési tér elemei alkotják a populáció egyedeit, melyeket keresztezni és mutálni lehet, így új egyedek hozhatók létre. A genetikus algoritmus működése során egyrészt új egyedeket hoz létre a keresztezés és a mutáció operátorokkal, másrészt kiszűri a rosszabb fitnesz függvény értékkel rendelkező egyedeket és eltávolítja a populációból.

Lépései:

- Inicializáció

A kezdeti populációt legegyszerűbb véletlenszerűen generálni. A populáció mérete a probléma természetétől függ, de leggyakrabban néhány száz vagy néhány ezer egyedből áll. Hagyományosan az egyedek a keresési téren egyenletesen oszlanak el.

- Kiválasztás

Minden sikeres generációban a jelenlegi populáció egy része kiválasztásra kerül szaporodásra. Általában fitnesz alapján történik, ahol a fittebb egyedek (a fitnesz függvény szerint) valószínűbben kerülnek kiválasztásra. A fitness függvény a példány minőségét méri.

- Szaporítás

Egyedekből újabb egyedeket a keresztezés (vagy rekombináció) és a mutáció művelettel lehet létrehozni, ez a szaporítás. Ezeket az operátorokat általában véletlenszerűen alkalmazzák.

- Leállás

[5] A genetikus algoritmusok elméleti összefoglalója Laky Sándor 2012-es Metaheurisztikus optimalizáció a geodéziában című PhD dolgozata alapján készült.

A genetikus algoritmusok rendszerint addig futnak, amíg egy leállási feltétel nem teljesül. Lehet ez egy adott generációszám elérése, vagy ha a legjobb egyed fitnesz értéke már nem javul jelentős mértékben egy-egy iterációval.

A genetikus algoritmusok részleteibe itt most nem megyünk bele, sok nehéz programozási feladat megoldását segítik, de nem garantálják, hogy megtalálják az optimumot. Mivel a globális minimalizálás relatíve lassú és sok műveletet igényel, ezért a következő stratégia célszerű:

- aránylag kis populációval és kevés generáció engedélyezésével a globális minimum közelébe jutunk,

- majd onnan indítva valamilyen lokális módszerrel pontosítjuk azt

A MATLAB YouTube csatornáján is található egy szemléletes példa a genetikus algoritmusok működéséről: https://www.youtube.com/watch?v=1i8muvzZkPw

GENETIKUS ALGORITMUS HASZNÁLATA MATLAB-BAN

Határozzuk meg a globális minimumát az előző felületnek a megadott tartományon genetikus algoritmust használva! Ehhez a **ga** (genetic algorithm) függvényre lesz szükségünk, illetve a megfelelő beállításokhoz a **gaoptimset** függvényre.

A **gaoptimset** segítségével állíthatjuk be a kezdeti populáció darabszámát és a használni kívánt generációk számát is. használjunk most egy 200 elemből álló populációt és 20 generációt, illetve állítsuk be, hogy az iterációs lépéseket is jelezze ki! A szám megjelenítési formátumát is állítsuk nagyobbra, hogy látszódjon a különbség több futtatás között!

```
> % inicializálás
> format longG;
> options =
  gaoptimset('Generations',20,'PopulationSize',200,'Display','iter');
```

A **ga** algoritmust többféle paraméter megadásával lehet meghívni, általános esetben így néz ki a parancs:

```
[x,fval] = ga(fitnessfcn,nvars,A,b,Aeq,beq,LB,UB,nonlcon,options)
```

A bemenetek között az optimalizálandó célfüggvény a *fitnessfcn*, a változók száma az *nvars* (jelen esetben 2, mivel kétváltozós optimalizálás a feladat). *A,b,Aeq,beq* lineáris megkötések paraméterei (egyenlőtlenség, egyenlet), amik ebben az esetben nincsenek, ha szükséges használni őket, akkor ezek jelentősen le tudják lassítani a futásidőt. LB (lower boundary), UB (upper boundary) a vizsgált tartomány alsó és felső határait tartalmazzák egy-egy vektorban. A *nonlcon* paraméterben nemlineáris feltételeket lehetne megadni. Az utolsó paraméter pedig az opciók, ahol a **gaoptimset**-ben megadott értékeket vehetjük figyelembe. Azon paraméterek helyett, amiket nem kívánunk használni egy-egy üres mátrixot kell megadnunk, mivel a Matlab a megadott sorrendben várja az adatokat. Fontos, hogy a Matlab a többi beépített parancshoz hasonlóan vektorváltozós függvényt vár el! Amennyiben kétváltozós függvény optimalizálása a feladat, akkor először azt vektorváltozóssá kell alakítani. Itt ezt már megcsináltuk korábban, amikor az *f* függvényből az *F* függvényt előállítottuk.

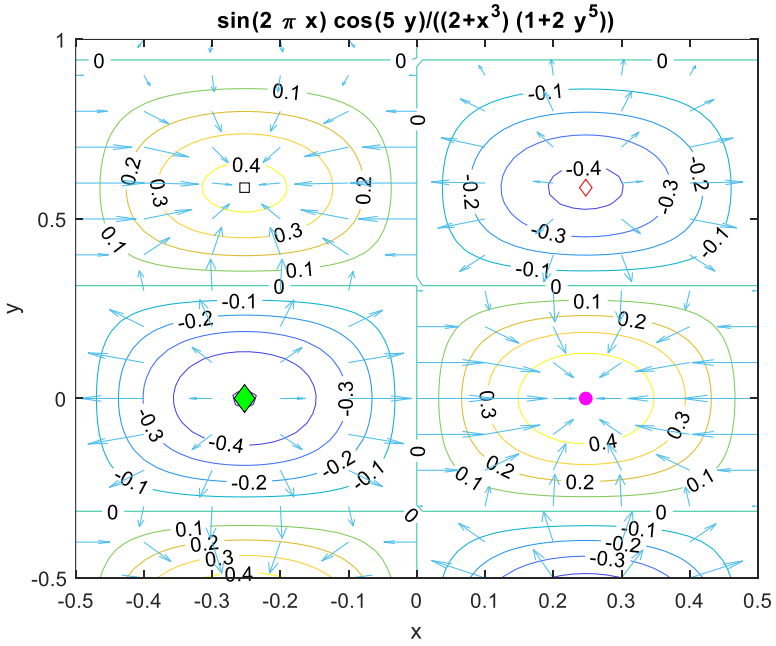
```
> %% Megoldás genetikus algoritmussal
> % [x,fval] = ga(fitnessfcn,nvars,A,b,Aeq,beq,LB,UB,nonlcon,options)
```

```
> A = []; b = []; Aeq = []; beq = []; % lineáris megkötés
> nlc = []; % nemlineáris megkötés
> LB = [-0.5,-0.5]; % lower boundary - alsó határ
> UB = [0.5,1]; % upper boundary - felső határ
> % közelítő minmumhely genetikus algoritmussal
> xga = ga(F,2,A,b,Aeq,beq,LB,UB,nlc,options)
> % xga = -0.252456968480699     -0.000708545893755685
> % fga =  -0.503991955761332
```

Futtassuk le többször az algoritmust! Látni fogjuk, hogy a megoldás minden alkalommal kicsit más lesz, de nagyságrendileg nem változik.

Pontosítsuk az eredményt lokális minimumkereső algoritmus használatával, kezdőértéknek használjuk a genetikus algoritmussal meghatározott minimumot!

```
> % minmumhely pontosítása lokális optimalizációs algoritmussal
> [xy_min zmin]=fminsearch(F,xga)
> % xy_min = -0.252380312148973     -7.22400617317352e-06
> % zmin =  -0.503995085159198
> plot(xy_min(1),xy_min(2),'kd','MarkerSize',10, 'MarkerFaceColor','g')
```



Hasonlítsuk össze a genetikus algoritmussal kapott minimum értékét a lokális algoritmussal pontosított értékkel!

```
> F(xy_min)<F(xga) % logical 1 -> igaz
```

GYAKORLÓ FELADATOK OPTIMALIZÁCIÓHOZ[6]

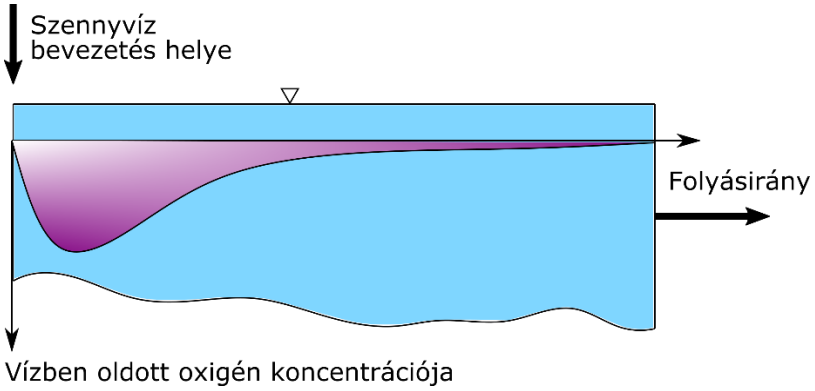
Többféle feladatot átnéztünk az előzőekben lokális és globális optimalizációra, akár minimum, akár maximum keresés kapcsán. Az optimalizáció egy nagyon gyakori feladat, sokféle különböző építőipari területen lehet rá szükség. Nézzünk meg néhány optimalizációra vezető feladatot gyakorlásképp. Az első esetében vizsgáljuk meg egy folyóba torkolló tisztított szennyvízbevezetés esetében a folyó vízben oldott oxigén

[6] Otthoni gyakorláshoz

koncentrációjának minimális értékét, amihez egy egyváltozós optimalizációt fogunk használni. A második esetben egy nemlineáris legkisebb négyzetes feladatot fogunk megoldani, a harmadik esetben pedig egy rácshálóban mért domborzat legmagasabb és legalacsonyabb pontját keressük meg globális optimalizációval.

SZENNYEZŐ ANYAG TERJEDÉSÉNEK, KONCENTRÁCIÓ ELOSZLÁSÁNAK MEGHATÁROZÁSA (EGYVÁLTOZÓS OPTIMALIZÁCIÓ)

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]:

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

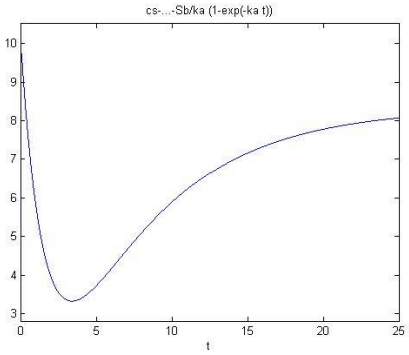
ahol t a tartózkodási idő [nap], c_s a telítési koncentráció (most az értéke: 10 mg/L), L_0 a biokémiai oxigénigény (BOD) a betáplálásnál (50 mg/L), k_d a lebomlási sebesség [0.1 1/nap], k_s a kiülepedési sebesség [0.05 1/nap], k_a az átlevegőzési sebesség [0.6 1/nap], S_b pedig a kiülepedési oxigénigény [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!

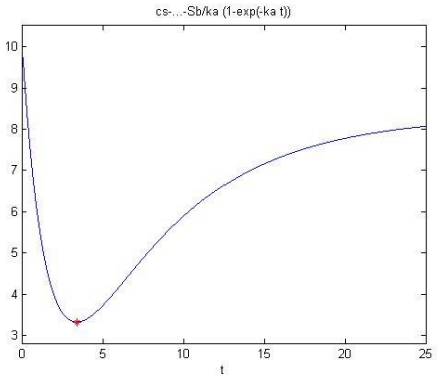
```
> figure(1)
> fplot(c,[0 25])
```

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!

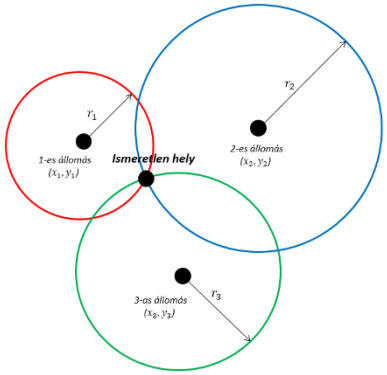
```
> %% Newton módszer
> syms t
> df = diff(c(t),t) % df = (5*exp(-(3*t)/20))/3 - (23*exp(-(3*t)/5))/3
```

```
> 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?

NEMLINEÁRIS LEGKISEBB NÉGYZETES PROBLÉMA (TÖBBVÁLTOZÓS NEWTON MÓDSZER, NELDER-MEAD MÓDSZER, GENETIKUS ALGORITMUS)

Nem csak egy függvénnyel megadott vagy mért felület szélsőértékeinek meghatározását oldhatjuk meg kétváltozós optimalizációval. 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:

$$(x - x_i)^2 + (y - y_i)^2 = {r_i}^2$$

A megoldáshoz először rendezzük 0-ra az egyenletet!

$$(x - x_i)^2 + (y - y_i)^2 - {r_i}^2 = 0,$$

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! A mobiltornyok koordinátái és a mért távoságok megtalálhatóak a tornyok.txt fájlban is, innen is beolvashatjuk az adatok, vagy begépelhetjük őket.

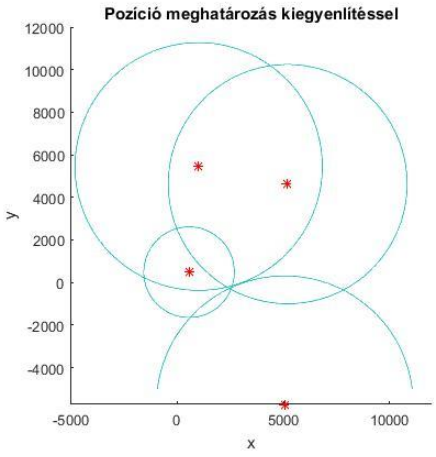
```
> clear all; clc; close all;
> xt = [561; 5203; 5067; 1012]
> yt = [487; 4625; -5728; 5451]
> rt = [2130; 5620; 6040; 5820]
> % Körök középpontjai
> figure(1); hold on;
> plot(xt, yt, 'r*')
```

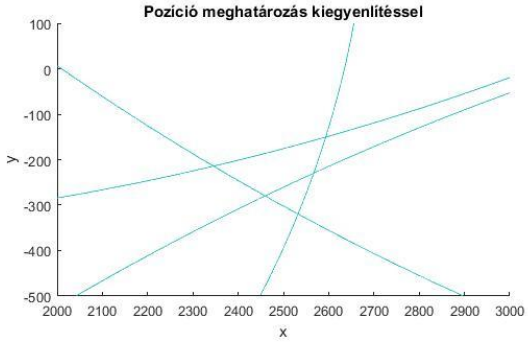
Ezután definiáljuk a kör egyenletét általánosan, és szimbolikus x,y változót használva ábrázoljuk az egyes köröket!

```
> % Kör általános egyenlete
> kor = @(x,y,x0,y0,r) (x-x0).^2 + (y-y0).^2 - r.^2
> % körök ábrázolása
> syms x y
> E = kor(x,y,xt,yt,rt)
> 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')
```





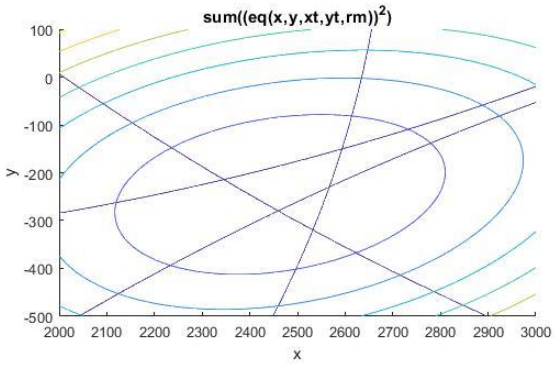
Látjuk, hogy a négy kör nem egy pontban metszi egymást, hanem közrezárnak egy területet, ahol a feltételezett pozíciónk van. Ezt a legvalószínűbb helyzetet

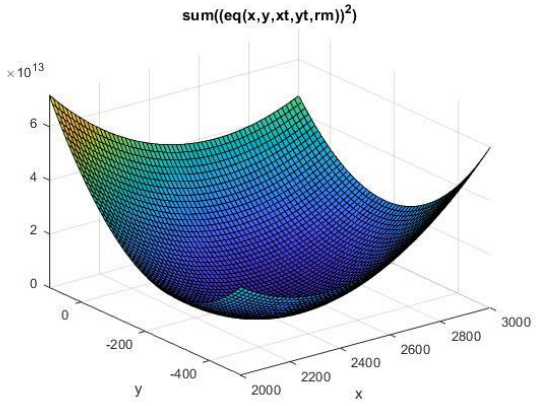
határozhatjuk meg a maradék eltérések négyzetösszegének minimalizálásával. Írjuk fel a minimalizálandó függvényt (*f*), ami a maradék eltérések négyzetösszege lesz

$$f(x,y) = \sum_{i=1}^{n} ((x - x_i)^2 + (y - y_i)^2 - r_i{}^2)^2$$

Definiáljuk a célfüggvényt és ábrázoljuk is Matlab-ban szintvonalakkal és 3D felülettel is!

```
> % minimalizálandó függvény
> f = sum(kor(x,y,xt,yt,rt).^2) % szimbolikus kifejezés
> F = matlabFunction(f) % f szimbolikus kifejezés függvénnyé alakítása
> fcontour(f,[2000 3000 -500 100]);
> figure(3)
> fsurf(f, [2000 3000 -500 100])
```

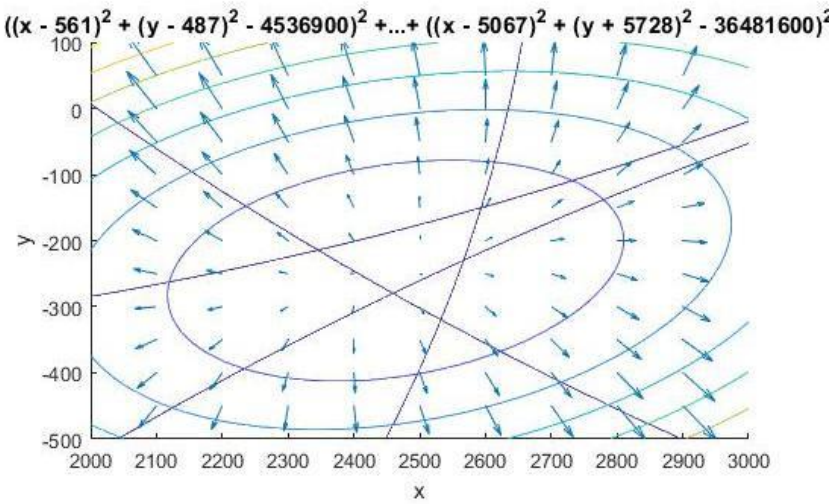




Megoldás többváltozós Newton-módszerrel

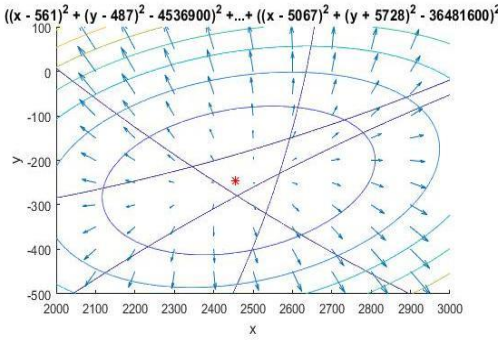
A fenti függvény minimum helye lesz a keresett pozíció legvalószínűbb értéke. Hogyan tudjuk ezt megtalálni? Használhatjuk például a többváltozós Newton-módszert!

```
> % Gradiens vektor ábrázolása numerikusan
> [X,Y] = meshgrid(2000:100:3000, -500:50:100);
> Z = F(X,Y); % függvény értékek a rácspontokban
> [px,py] = gradient(Z); % gradiensek számítása numerikusan
> figure(2)
> quiver(X,Y,px,py)
```




```
> % 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.

Megoldás szimplex módszerrel, beépített Matlab függvénnyel

Oldjuk meg a feladatot szimplex módszerrel, beépített Matlab függvény használatával! 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')
```

Ellenőrzés

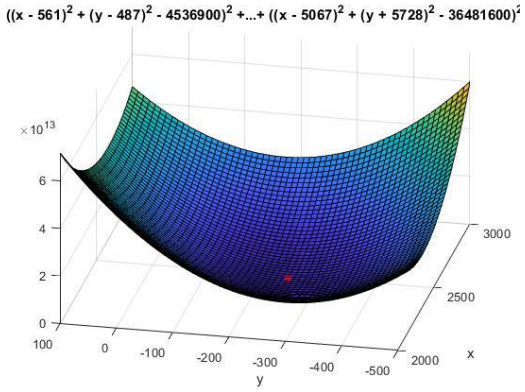
Ellenőrzésként nézzük meg az eltérést a mért távolságok és kiegyenlített álláspont-mobiltornyok távolságai között!

```
> % az álláspont és a mobiltornyok távolságai
> ex = xt - sol(1); ey = yt - sol(2);
> % eltérések a mért értékekhez képet
> er = rt - 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*')
```



Megoldás globális optimalizációval, genetikus algoritmus használatával

Oldjuk meg Matlab-ban genetikus algoritmusok használatával is a mobiltornyokra végzett távolságméréssel történő pozíció meghatározás példáját.

```
>  %% Megoldás genetikus algoritmussal
>  % x = ga(fitnessfcn,nvars,A,b,Aeq,beq,LB,UB,nonlcon,options)
>  format longG;
>  options =
   gaoptimset('Generations',20,'PopulationSize',200,'Display','iter');
>  A = []; b = []; Aeq = []; beq = []; % lineáris megkötés
>  nlc = []; % nemlineáris megkötés
>  LB = [2000,-500]; % lower boundary - alsó határ
>  UB = [3000,100]; % upper boundary - felső határ
>  % közelítő minmumhely genetikus algoritmussal
>  xga = ga(F,2,A,b,Aeq,beq,LB,UB,nlc,options) % 2435.515 -246.959
```

Futtassuk le többször az algoritmust! Látni fogjuk, hogy a megoldás minden alkalommal kicsit más lesz, de nagyságrendileg nem változik.

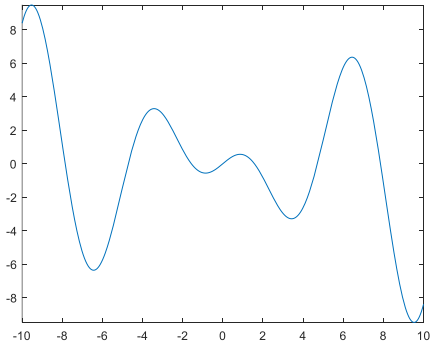
Pontosítsuk az eredményt lokális minimumkereső algoritmus használatával, kezdőértéknek használjuk a genetikus algoritmussal meghatározott minimumot!

```
>  % minmumhely pontosítása lokális optimalizációs algoritmussal
>  [xy_min zmin]=fminsearch(F,xga) % 2454.657 -246.948 861310686413.212
>  hold on; plot3(xy_min(1),xy_min(2),F(xy_min),'r*')
>  F(xy_min)<F(xga) % logical 1 -> igaz
```

GENETIKUS ALGORITMUS HASZNÁLATÁRA EGYVÁLTOZÓS ESETBEN

Keressük meg az alábbi függvény globális minimumát genetikus algoritmus használatával az $x \in [-10,10]$ intervallumon:

$$f(x) = x \cdot \cos(x)$$



```
> %% f(x)=x*cos(x) függvény globális minimuma x in [-10,10]
> clear all; clc; close all;
> format longG;
> f=@(x) x.*cos(x)
>
> % a függvény ábrázolása
> figure(1); fplot(f,[-10,10])
>
> options = gaoptimset('Generations',20,'PopulationSize',200,'Display','iter');
> A = []; b = []; Aeq = []; beq = []; % linear contraints
> nlc = []; % nonlinear constraint
> LB = -10; % lower boundary
> UB = 10; % upper boundary
> tic
> xga = ga(f,1,A,b,Aeq,beq,LB,UB,nlc,options)
> toc
>
>                                      Best           Mean      Stall
> % Generation      Func-count         f(x)           f(x)
Generations
>      1               400          -9.476          -2.38        0
>      2               590          -9.476         -4.872        1
> % ...
>     56             10850          -9.477         -9.477       31
>     57             11040          -9.477         -9.477       32
> % Optimization terminated: average change in the fitness value less than options.FunctionTolerance.
> % xga = 9.52929196398106
>
> Elapsed time is 0.957920 seconds.
> f(xga) %  -9.47729425945419
```

Futtassuk le többször az algoritmust, úgy, hogy ne csak 4 jegyre írassuk ki az eredményt. Látni fogjuk, hogy a megoldás minden alkalommal kicsit más lesz, de nagyságrendileg nem változik. Megpróbálhatjuk változtatni a populáció számot, vagy az egymást követő generációk számát. Állítsuk pl. 10-re a generáció számot az inicializáláskor, nézzük meg mit ír a megoldáshoz ebben az esetben!

```
> Optimization terminated: maximum number of generations exceeded.
> xga =    9.53699499687649
> Elapsed time is 0.420323 seconds.
> f(xga) % -9.47700990127281
```

Gyorsabban megkaptuk az eredményt, de a pontosság még nem érte el az alapértelmezett értéket, előbb befejeződött a 10 generáció. Pontosítsuk az eredményt lokális minimumkereső algoritmus használatával, kezdőértéknek használjuk a genetikus algoritmussal meghatározott minimumot!

```
> % Eredmény pontosítása lokális algoritmussal
> xga2= fminsearch(f,xga) % 9.52930965704313
> fmin2 = f(xga2) % -9.47729425651355
> hold on; plot(xga2,fmin2,'ro')
> % ell.
> fmin2<fmin %  1
> df = fmin-fmin2 % 0.000111857050457687
```

Természetesen, hogy mekkora a javulás, az a futtatásonként eltérő lehet, illetve attól is függ, hogy csak 10 vagy 100 generációt használtunk végül, mennyire volt pontos eredetileg a genetikus algoritmussal kapott érték.

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.
fminbnd	-	Minimum hely megkeresése intervallum megadásával
ga	-	Optimalizáció genetikus algoritmus használatával
gaoptimset	-	genetikus algoritmus inicializálása, paraméterek megadása