KIEGYENLÍTŐ SZÁMÍTÁSOK II.

SÍK ILLESZTÉSE

Olvassuk be a domborzatmodellezéskor már használt mérési állományunkat (meres_coo.txt)! Korábban láttuk a szintvonalas domborzatnál, hogy a terep meglehetősen síknak tekinthető. Illesszünk egy kiegyenlítő síkot a pontokra! Keressük meg a sík paramétereit.

2001 577057.011 188795.517 142.042
2002 577051.903 188783.028 140.821
2003 577051.044 188772.868 140.317
2004 577053.460 188758.714 139.622
…

A sík általános egyenlete a következő:

$$a_0 + a_1 \cdot x + a_2 \cdot y = z$$

Ahány pontunk van, annyi egyenletet tudunk felírni (163), míg összesen 3 ismeretlenünk van az a_0, a_1, a_2 paraméter, tehát ismét túlhatározott lineáris egyenletrendszert kell megoldani. Mátrix alakban felírva a következő egyenletrendszert kell megoldanunk:

$$\begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \dots & \dots & \dots \\ 1 & x_n & y_n \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} z_1 \\ z_2 \\ \dots \\ z_n \end{pmatrix}$$

vagyis:

$A \cdot p = z$, ahol p a paraméter vektor $p = [a_0; \quad a_1; \quad a_2]$.

A megoldást a már korábban levezetett alakban kapjuk meg:

$$p = (A^T \cdot A)^{-1} \cdot A^T \cdot z$$

MEGOLDÁS MATLAB/OCTAVE HASZNÁLATÁVAL

Oldjuk meg ezt Matlabban/Octave-ban, majd jelenítsük meg az eredményt!

```
clear all; close all; clc;
page_screen_output(0); % ez csak Octave-ban kell!

data=load('meres_coo.txt');
x = data(:,2);
y = data(:,3);
z = data(:,4);

A = [ones(size(x)) x y];
p = inv(A'*A)*(A'*z)
```

A futtatás után kapunk egy figyelmeztetést:

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 4.191364e-021.
```

Ez arra utal, hogy a megoldás pontatlan lehet, és nagyon bizonytalan. Kérdezzük le az $A^T \cdot A$ mátrix kondíciószámát!

```
c=cond(A'*A)
```

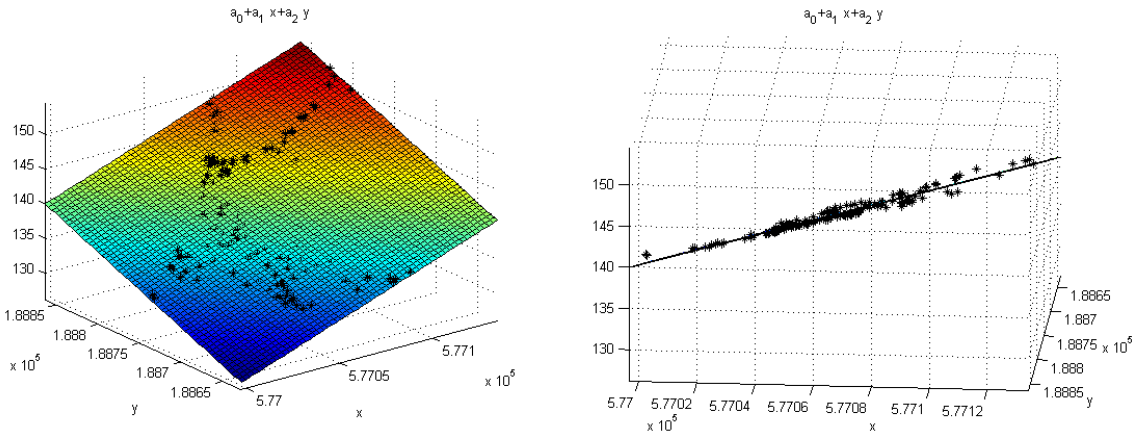
Eredmény:

```
c =  2.1001e+020
```

Minél nagyobb a kondíciószám, annál bizonytalanabb a megoldás, mivel a kondíciószám egy hányados a kimenet és a bemenet relatív hibája között. Egy kis változás a bemeneti adatokban a megoldás nagy változását okozhatja.

Ábrázoljuk azért a megoldást!

```
a0=p(1)
a1=p(2)
a2=p(3)

f = @(x,y) a0+a1*x+a2*y

figure(1); hold on;
ezsurf(f, [min(x) max(x) min(y) max(y)])
plot3(x,y,z, 'k*')
```



Ha megnézzük az adatokat, akkor látszik, hogy több százezres nagyságrendű EOV y,x koordinátákkal dolgoztunk, hozzá 100-200 m körüli z értékekkel. Ilyenkor a numerikus számítás bizonytalanságát csökkenthetjük, ha áttérünk súlyponti koordinátákra és nem a több százezres koordinátákkal dolgozunk.

MEGOLDÁS SÚLYPONTI KOORDINÁTÁKKAL

A súlyponti y,x koordinátákhoz ki kell számolni a koordináták átlagát és kivonni ezt a mérési eredményekből. A végleges eredménynél és megjelenítésnél sem szabad azonban elfelejteni, hogy ezeket az átlagokat ki kell vonni az y,x értékekből!

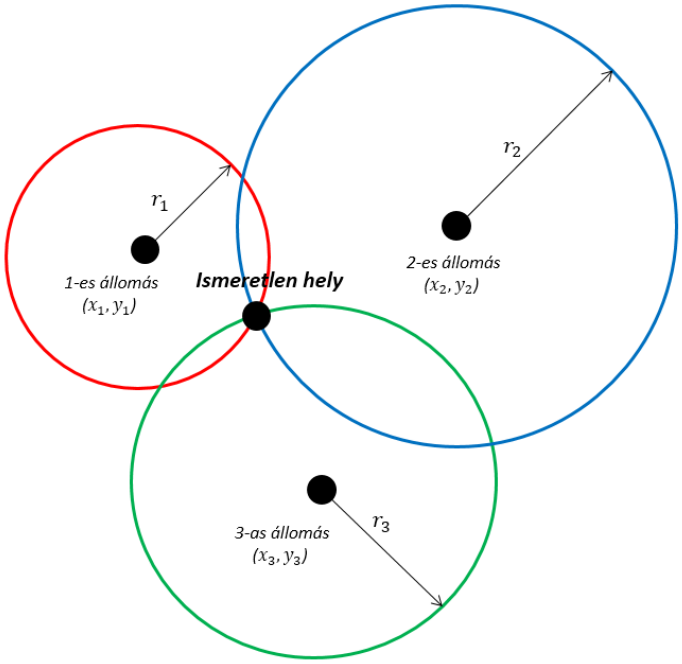
```
% sulyponti koordinatak
XS = mean(x)
YS = mean(y)

xs = x - XS;
ys = y - YS;

As = [ones(size(xs)) xs ys];
ps = inv(As'*As)*(As'*z)
cs=cond(As'*As)

a0s=ps(1)
a1s=ps(2)
a2s=ps(3)

fs = @(x,y) a0s + a1s * (x-XS) + a2s * (y-YS)

figure(2)
ezsurf(fs, [min(x) max(x) min(y) max(y)])
hold on;
plot3(x,y,z, 'b.', 'MarkerSize', 15)
```

A súlypont koordinátáit YS és XS jelöli. A kiszámított súlyponti koordináták pedig ys és xs. A minimális és maximális xs: [-66.342; 71.208], a minimális és maximális ys: [-126.7668; 108.612]. Ezekkel az értékekkel számolva jóval kisebb a numerikus számítás pontatlansága, a kondíciószám 10^{20} nagyságrend helyett mindössze 10^3 nagyságrendű. Most nem is kapunk figyelmeztetést, csak a megoldást.

Egy másik módja a megoldás pontosításának, ha nem az $x = (A^T \cdot A)^{-1} \cdot A^T \cdot b$ alakban oldjuk meg a problémát, hanem használjuk az Octave/Matlab valamelyik beépített megoldó módszerét túlhatározott egyenletekre. Pl az $A \cdot x = b$ egyenletet túlhatározott esetben is megoldhatjuk az $x = A \backslash b$ alakban. Ez Octave esetében SVD (Singular Value Decomposition) felbontással történő megoldást jelent, ami numerikusan sokkal stabilabb. Ha ezzel oldjuk meg a feladatot, akkor az eredeti koordinátákkal sem lép fel a rosszul kondicionáltság problémája. Az eredmények némileg eltérnek a másik módszerrel kapott eredményektől az eredeti koordinátákat használva, míg a súlyponti koordinátáknál megegyeznek.

POZÍCIÓ MEGHATÁROZÁS MOBILTELEFONOKKAL

A mobiltelefonok pozíciójának meghatározásakor rendelkezésre áll az eszköz és a mobiltornyok távolsága. A távolságok egy-egy kört határoznak meg a bázisállomások körül (lásd ábra). A körök másodfokú egyenletek, és ezek metszéspontja adja a mobiltelefon pozícióját. Ez a probléma az ívmetszés (angolul lateration). Amennyiben legalább 3 távolság, azaz 3 kör, valamint a bázisok koordinátái ismertek, az ismeretlen hely meghatározható. Több távolság esetében kiegyenlítésre van szükség, ami tekintve az egyenleteket, most nemlineáris egyenletrendszerre alkalmazott legkisebb négyzetek módszerével történhet.



Az egyes mobiltornyok koordinátáit és a távolságméréseket az alábbi táblázat foglalja össze.

Mobil torony sorszáma	**X koordináta** (x_i) **[m]**	**Y koordináta** (y_i) **[m]**	**Mért bázis-terminál távolság** (r_i) **[m]**
1	561	1487	2130
2	5203	4625	5620
3	5067	-5728	6040
4	1012	5451	5820

Az egyenleteket a következő implicit alakban adhatjuk meg:

$$(x - x_i)^2 + (y - y_i)^2 - r^2 = 0,$$

ahol x_i, y_i a mobiltornyok koordinátái, x,y pedig a keresett álláspont.

A megoldáshoz ismét az eltérések négyzetösszegét kell minimalizálni. Ehhez most az Octave/Matlab beépített szimplex módszerét fogjuk használni, az fminsearch parancsot (de akár az fminunc parancs is használható, ami kvázi-Newton minimalizálást alkalmaz). Itt viszont már szükség lesz kezdőérték megadására, ami lineáris esetben még nem volt követelmény. A kezdőértékeket most ábrából vesszük, így először szükséges ábrázolni az egyenleteket.

Adjuk meg először a mobiltornyok koordinátáit, a mért távolságokat, és ábrázoljuk a pontokat!

```
clear all; clc; close all;
page_screen_output(0); % ez csak Octave-ban kell!

xt = [561; 5203; 5067; 1012]
yt = [487; 4625; -5728; 5451]
rm = [2130; 5620; 6040; 5820]

figure(1); hold on;
plot(xt, yt, 'r*')
```

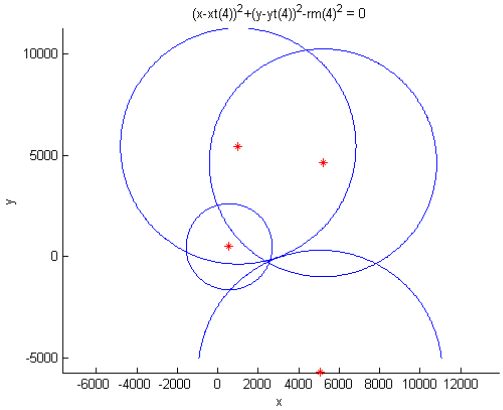
Definiáljuk ezek után a tornyoktól mért távolságokat függvényekkel és ábrázoljuk ezeket!

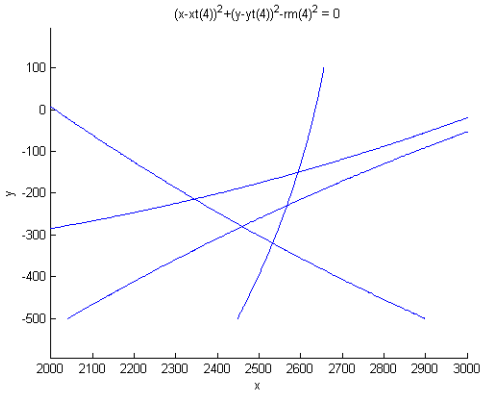
```
eq1 = @(x,y) (x-xt(1)).^2 + (y-yt(1)).^2 - rm(1).^2
eq2 = @(x,y) (x-xt(2)).^2 + (y-yt(2)).^2 - rm(2).^2
eq3 = @(x,y) (x-xt(3)).^2 + (y-yt(3)).^2 - rm(3).^2
eq4 = @(x,y) (x-xt(4)).^2 + (y-yt(4)).^2 - rm(4).^2

ezplot(eq1, [-5000 12000])
ezplot(eq2, [-5000 12000])
ezplot(eq3, [-5000 12000])
ezplot(eq4, [-5000 12000])
axis equal
```

Figyeljünk arra, hogy a függvények definiálásakor használjunk pontot (.) a hatványozás, szorzás, osztás művelete előtt, hogy vektorokra is hívható legyen elemenként a függvény.

Nagyítsunk rá a minket érdeklő területre!

```
figure(2); hold on;
ezplot(eq1, [2000 3000 -500 100])
ezplot(eq2, [2000 3000 -500 100])
ezplot(eq3, [2000 3000 -500 100])
ezplot(eq4, [2000 3000 -500 100])
axis equal
```





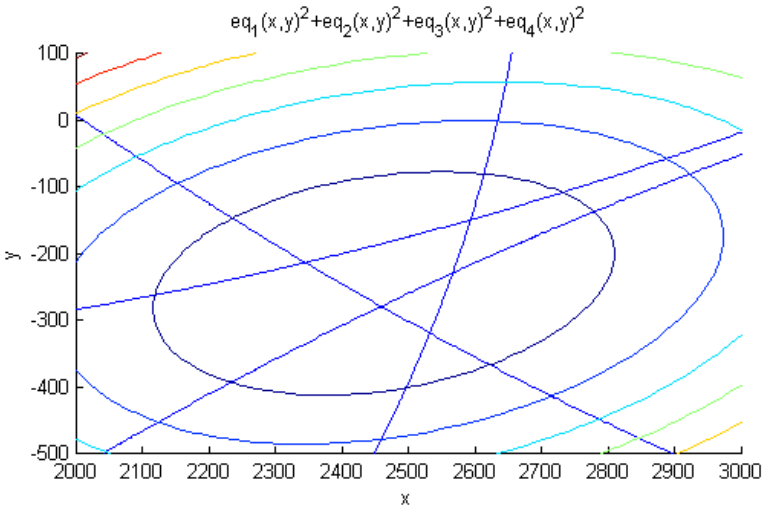
Definiáljuk a minimalizálandó függvényt, az eltérések négyzetösszegét!

```
err = @(x,y) eq1(x,y).^2 + eq2(x,y).^2 + eq3(x,y).^2 + eq4(x,y).^2;
```

Ábrázoljuk ezt is a fenti rajzon szintvonalakkal!

```
ezcontour(err,[2000 3000 -500 100]);
```

(Színezett szintvonalakkal ugyanez: `ezcontourf(err,[2000 3000 -500 100]);`)



A megoldás a fenti függvény minimuma lesz. Ehhez fel kell vennünk egy kezdőértéket. A rajz alapján a kezdőérték legyen:

```
x0 = [2400; -300]
```

A megoldáshoz az ’err’ függvényt vektor változóssá kell alakítanunk! Rajzoljuk ki a megoldást is!

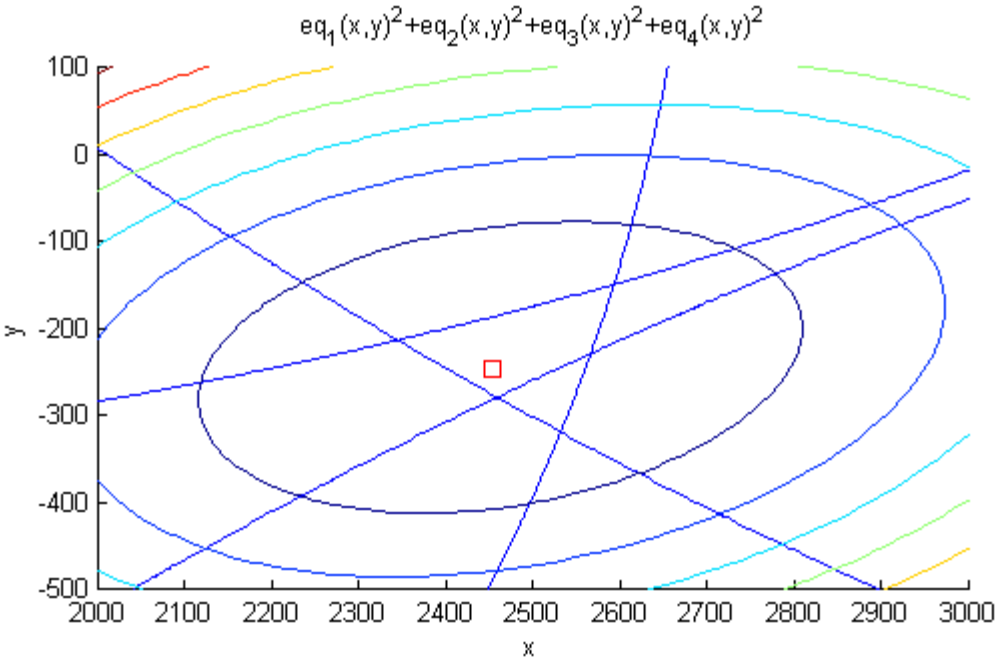
```
err1 = @(x) err(x(1),x(2));
sol = fminsearch(err1,x0)
% sol2 = fminunc(err1,x0)
plot(sol(1), sol(2), 'rs')
plot(xm, ym, 'r*')
```

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)
```

Az eltérések:

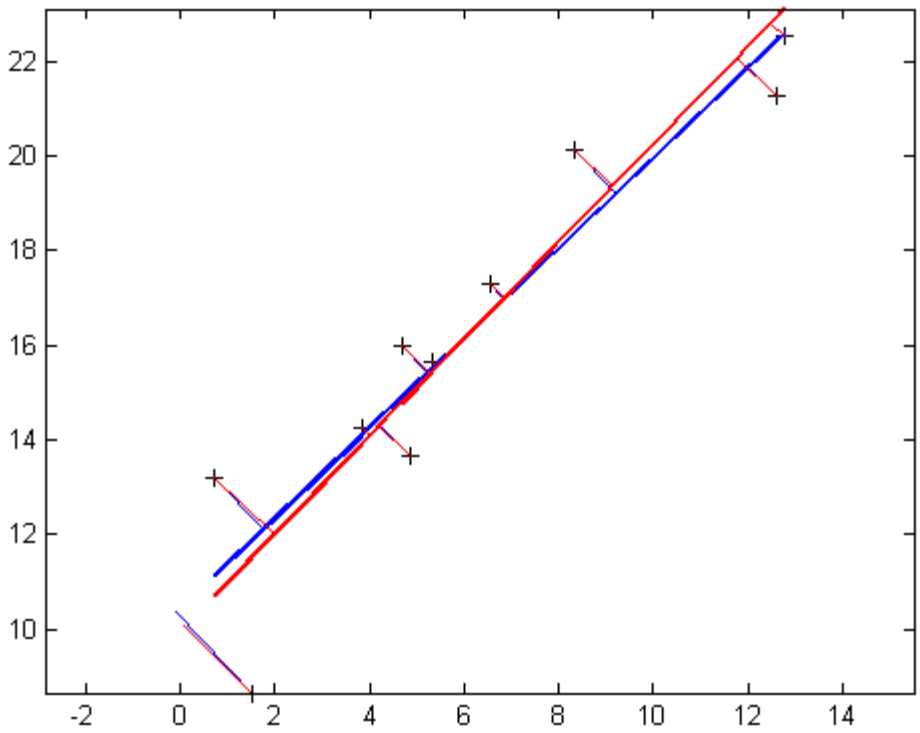
```
er =
   99.0847
   26.3188
  -31.7595
  -57.7440
```



TELJES LEGKISEBB NÉGYZETEK MÓDSZERE

A következőkben egy példa lesz a teljes legkisebb négyzetek módszerével történő egyenes illesztésére. A korábbi egyenes illesztésnél az y irányú eltérések négyzetösszegét minimalizáltuk az egyeneshez képest, feltételeztük, hogy az x koordináta hibátlan. Ez a feltételezés azonban többnyire nem állja meg a helyét, többnyire egy mérésnél mind a két koordináta hibával terhelt, ezért jobb megközelítés, ha a pontok egyenestől való távolságának négyzetösszegét minimalizáljuk.

Az alábbi ábrán az előző gyakorlatban a hagyományos legkisebb négyzetek módszerével illesztett egyenes látható kékkel és pirossal a teljes legkisebb négyzetek módszerével illesztett egyenes, berajzolva az egyenestől mért távolságokat.



A nehézséget az jelenti a feladatban, hogy egy olyan egyenestől való távolságot kell meghatározni, aminek egyelőre nem ismerjük az egyenletét, és maga a probléma sem lineáris. A feladat iterációkkal határozható meg, szükség van kezdőértékekre a paraméterekhez az elején, ez lehet például a hagyományos legkisebb négyzetek módszerével kapott egyenes két paramétere. Fel kell vennünk egy célfüggvényt, amit minimalizálni szeretnénk, ez a minimalizálandó távolság négyzetösszeg lesz, ez egy külön függvényben kerül definiálásra, és szükség lesz egy függvényre, ami pont-egyenes távolságot számol.

A megoldás a következő lesz:

A célfüggvény:

```
function celertek = celfuggveny(p)
  global x;
  global y;

  celertek = 0;
  for i=1:numel(x)
    P = [x(i); y(i)];
    celertek = celertek + pont_egy_tav(p, P)^2;
  end
end
```

Pont-egyenes távolságot számító függvény:

```
function [t n Dy] = pont_egy_tav(p_egyenes, P_pont)
  % Egyenes egyseghosszu iranyvektora
  b = [1; p_egyenes(2)] / sqrt(1^2 + p_egyenes(2)^2);
```

```
  % Egyenesre meroleges egyseghosszu vektor
  n = [b(2); -b(1)];
  % Az egyenes es a pont fuggoleges tavolsaga
  Dy = P_pont(2) - (p_egyenes(1) + p_egyenes(2)*P_pont(1));
  % Ennek vektor megfeleloje
  Dyv = [0; Dy];
  % Vetulete ez egyenes normalisara
  t = Dyv'*n;
  % Az egyseghosszu normalis vektor nyujtasa t hosszura
  n = t * n;
end
```

A teljes legkisebb módszerek szerinti kiegyenlítés (kezdőérték a hagyományos legkisebb négyzetek módszerével), minimalizálás az fminunc függvénnyel.

```
clear all; clc; close all; %
page_screen_output(0);
global x;
global y;

p = [8.765; 1.234];
x = [1:10]';
y = p(1) + p(2)*x;

x = x + randn(numel(x),1);
y = y + randn(numel(y),1);

A = [ones(numel(x),1) x];
p1 = inv(A'*A)*A'*y;
p
p1

figure(1);
hold off;
plot(x, y, 'k+');
hold on;
plot(x, p1(1) + p1(2)*x, 'b', 'LineWidth', 2);
axis('equal');

for i=1:numel(x)
  P = [x(i); y(i)];
  [t n Dy] = pont_egy_tav(p1, P);
%  plot([P(1); P(1)], [P(2); P(2)-Dy], 'g');
  plot([P(1); P(1)-n(1)], [P(2); P(2)-n(2)], 'b');
end

p2 = fminunc('celfuggveny', p1);

plot(x, p2(1) + p2(2)*x, 'r', 'LineWidth', 2);

celfuggveny(p1)
celfuggveny(p2)

for i=1:numel(x)
  P = [x(i); y(i)];
  [t n Dy] = pont_egy_tav(p2, P);
  plot([P(1); P(1)-n(1)], [P(2); P(2)-n(2)], 'r');
end
```