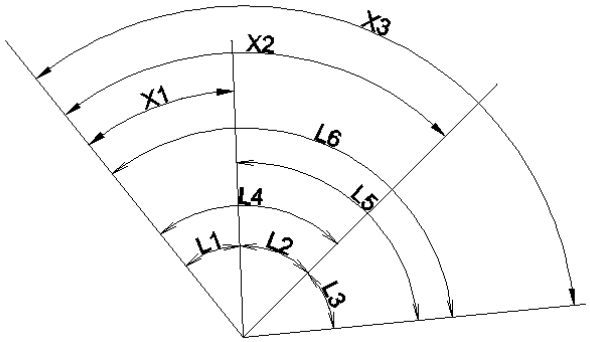
SAJÁT FÜGGVÉNYEK ÍRÁSA, KIEGYENLÍTŐ SZÁMÍTÁSOK ALAPJAI

SZÖGMÉRÉS KIEGYENLÍTÉSE

Határozzuk meg 4 irány által bezárt X_1, X_2 és X_3 szögeket, úgy, hogy a közbezárt szögeket minden kombinációban megmértük (L_1, L_2, L_3, L_4, L_5, L_6)!



(A példát lásd Detrekői: Kiegyenlítő számítások, Tankönyvkiadó, Budapest, 1991., 145-150. oldal)

A mérési eredmények a következőek:

L_1 = 30-40-24 L_4 = 72-46-40

L_2 = 42-06-12 L_5 = 88-18-27

L_3 = 46-12-20 L_6 = 118-58-59

a) A mérésekről először feltételezzük, hogy azonos pontosságúak.

b) Az L_1, L_2, L_3 méréseket μ=2”, az L_4, L_5, L_6 méréseket μ=4” középhibával jellemezhetjük.

ADATOK MEGADÁSA MATLAB/OCTAVE HASZNÁLATÁVAL

Adjuk meg a mérési eredményeket vektorokban, majd fűzzük össze őket egy mátrixba!

```
page_screen_output(0);
clc; clear all; close all;

L1 = [30,40,24];
L2 = [42,06,12];
L3 = [46,12,20];
L4 = [72,46,40];
L5 = [88,18,27];
L6 = [118,58,59];
L = [L1;L2;L3;L4;L5;L6]
```

SAJÁT FÜGGVÉNYEK KÉSZÍTÉSE FOK-PERC-MÁSODPERC ÉRTÉKEK KEZELÉSÉHEZ

FOK-PERC-MÁSODPERC ÁTVÁLTÁSA TIZEDFOKBA

Az Octave/Matlab alapértelmezésként a radiánt tekinti a szög mértékegységének (pl. sin, cos stb függvények esetében), illetve még tizedfokban megadott értékekkel is tud dolgozni (pl. sind, cosd stb) fok-perc-másodperc értékekkel azonban nem.

Első feladat, hogy valahogy kezelhetővé tegyük a fok-perc-másodperc értékeket. Írjunk függvényt, ami átváltja a szöget tizedfokba, illetve egy másikat, ami radiánba! Itt el kell döntenünk, hogyan szeretnénk megadni a fok-perc-másodperc értékeket az adott függvénynek, pl. lehetnek ezek különálló bemenetek vesszővel elválasztva, illetve egy bemenet, ahol vektorban tárolódnak az értékek. Ezt megtehetjük ún. anonym függvény írásával, ahol az m fájlunkban definiáljuk közvetlenül a függvényünket.

Pl. függvény 3 vesszővel elválasztott bemeneti változóval.

```
fpm = @(f,p,m) f + p/60 + m/3600;
```

Teszteljük:

```
fpm(30,40,24)
```

Vagy függvény egy vektor bemenettel, ahol a vektor elemeiben vannak megadva a fok-perc-másodperc értékek.

```
fpm2 = @(f) f(1) + f(2)/60 + f(3)/3600;
fpm2(L1)
```

Célszerűbb lehet azonban külön m fájlban definiálni ezeket, saját függvényként. így egyrészt más feladat során is használhatjuk ezeket, illetve több lehetőségünk van a függvény paraméterezésére. Megadhatunk pl. több kimenetet, ha szükséges, illetve a bemenetek számának, jellegének függvényében változhat a feldolgozás is, így akár egy függvényben tudjuk kezelni mind a kettő fenti esetet, hogy akár vektorként, akár külön bemeneti változókként megadhatjuk a fok-perc-másodperc értékeket.

A legegyszerűbb eset az alábbi:

```
function fok = fpm2fok(f,p,m)
      fok = f + p/60 + m/3600;
end
```

Most a függvénynek az fpm2fok nevet adtuk, ilyenkor fpm2fok.m fájlként is kell elmenteni. A függvény hívása egyszerűen az fpm parancs kiadásával történhet, abból a könyvtárból, ahová ezt az m fájlt mentettük.

Teszteljük:

```
%L1fok = fpm2fok(L1) % hibaüzenetet kapunk
L1fok = fpm2fok(L1(1), L1(2), L1(3))

Lf = L(:,1)
```

```
Lp = L(:,2)
Lm = L(:,3)
Lfok = fpm2fok(Lf, Lp, Lm)
```

Módosítsuk ezt a függvényt, hogy többféle bemenettel is működjön, akár külön-külön vannak megadva a fok-perc-másodperc értékek, akár egy 3 oszlopú sorvektorban, akár egy 3 oszlopú oszlopvektorban több szög egyszerre.

```
function fok = fpm2fok(f,p,m)
% A fuggveny fok-perc-masodperc ertekekbol tizedfokot szamol.
% A bemenet lehet vesszovel elvalasztva fok,perc, masodperc vagy
% lehet vektorban megadva [fok perc másodperc] alakban is.
% Matrixban adott bemenetet is elfogad a fuggveny, ahol
% az 1. oszlop a fok, a 2. a perc, a 3. a masodperc
%
% Kimenet: tizedfok (skalar vagy vektor bemenettol fuggoen)

switch nargin
    case 3
        fok = f + p/60 + m/3600;
    case 1
        fok = f(:,1) + f(:,2)/60 + f(:,3)/3600;
end
end
```

A függvény első sora után megadott megjegyzések (%) alkotják a help parancsra kiírt dokumentációt (a 'help fpm2fok' után ez jelenik meg a parancssorban). Saját függvényeket célszerű jól ledokumentálni, hogy később is tudjuk milyen bemenetet vár a program, és milyen kimenetet kapunk.

A nargin (Number of function input arguments) változó a bemenetek számát tárolja. Ha előre tudjuk, hogy maximum hány változót vár a függvény, akkor egyszerűen megadhatunk ennyi változót bemenetként, és utána a nargin lekérdezése után mindig azt a megoldást hívjuk meg, ami megfelel a változók számának. Ha a maximális változó szám ismeretlen, akkor a változó lista végére meg kell még adni a varargin változót is (Variable-length input argument list).

Használjuk a switch elágazást a változók számának megfelelő parancs hívására!

Ha külön vannak tárolva a fok-perc-másodperc értékek, akkor az f + p/60 + m/3600 parancs működik mind skalár, mind vektor adatokkal, viszont, ha egy 3 oszlopú mátrixban vannaz az adatok, akkor a f(1) + f(2)/60 + f(3)/3600 parancs nem működik az összes tárolt szögre (egy sor egy szög), ahhoz a f(:,1) + f(:,2)/60 + f(:,3)/3600 parancsot kell kiadni.

Teszteljük különböző bemenetekkel:

```
L1fok = fpm2fok(30,40,24)
L1fok = fpm2fok(L1)
Lfok = fpm2fok(L)
Lfok = fpm2fok(Lf,Lp,Lm)
```

Egy függvénynek nem csak egy, hanem több kimenete is lehet, jelen esetben például lehet a függvény egy másik kimenete a tizedfok mellet a szög radiánban kapott értéke is. Ehhez az alábbiak szerint módosítsuk a függvényt:

```
function [fok rad]= fpm2fok(f,p,m)
% A fuggveny fok-perc-masodperc ertekekbol tizedfokot és radiant szamol.
```

```
% A bemenet lehet vesszovel elvalasztva fok,perc, masodperc vagy
% lehet vektorban megadva [fok perc másodperc] alakban is.
% Matrixban adott bemenetet is elfogad a fuggveny, ahol
% az 1. oszlop a fok, a 2. a perc, a 3. a masodperc
%
% 1. kimenet: tizedfok (skalar vagy vektor bemenettol fuggoen)
% 2. kimenet: a szog erteke radianban

switch nargin
    case 3
        fok = f + p/60 + m/3600;
    case 1
        fok = f(:,1) + f(:,2)/60 + f(:,3)/3600;
end
rad = fok *pi / 180;
end
```

Ilyenkor, ha egy kimenettel hívjuk meg a függvényt az első kimenetet fogja csak visszaadni, a tizedfok értékét, több kimenet estében azonban mind a tizedfokot, mind a radián visszaadja. Például:

```
[a b] = fpm2fok(30,40,24)
% a =  30.673
% b =  0.53535

[Lfok Lrad] = fpm2fok(L)
```

FOK ÁTVÁLTÁSA RADIÁNBA (OTTHONI ÁTNÉZÉSRE)

Írjunk egy olyan függvényt, ami csak radiánba vált át fok értéket. Viszont többféle bemenetet kezeljen. A fok értéket meg lehessen adni akár tizedfokban, akár fok-perc-másodperc értékekben, és ez utóbbi is működhet a korábban megadott módokon (3 elemű sorvektorban, vagy vesszővel elválasztva).

```
function rad = fok2rad(f,p,m)
% Fokban lévő szoget radianba alakit.
% Ha egy bemenet van, akkor megvizsgalja, hogy 1 vagy 3 oszlopa van-e.
% Egy oszlop eseten tizedfokban adott szoget alakit at,
% ha 3 oszlopa van, akkor fok-perc-masodperc ertekeket alakit at radianna.
% Ha 3 bemenete van, akkor azt fok-perc-másodpercnek tekinti.

  switch nargin
    case 3
         rad = (f + p/60 + m/3600) * pi / 180;
    case 1
    if size(f,2)==1
        rad = f * pi / 180;
    end
    if size(f,2)==3
        rad = (f(:,1) + f(:,2)/60 + f(:,3)/3600) * pi / 180;
    end
  end
end
```

Ez utóbbi a (*pi/180) átváltáson kívül abban tér el az előzőtől, hogy egy bemenet esetén vizsgálja, hogy az a bemenet 1 vagy 3 oszlopból áll-e. Egy oszlop esetén feltételezi, hogy a szögek tizedfokban adottak, 3 esetén pedig fok-perc-másodpercben.

A size parancs a mátrix méretét adja vissza. size(M) hívása esetén először a sor, utána az oszlopok számát, size(M,1) a sorok számát, size(M,2) az oszlopok számát adja vissza.

TIZEDFOK ÁTVÁLTÁSA FOK-PERC-MÁSODPERCRE (OTTHONI ÁTNÉZÉSRE)

Az első függvényünk fordítottja is kelleni fog, ha az eredményeket fok-perc-másodperc értékben szeretnénk kiíratni, ahogy az geodéziában szokásos.

```
function fpm = fok2fpm(x);
% A fuggveny tizedfokbol fok-perc-masodperc ertekekbe szamol at.
% A bemenet lehet egy skalar vagy vektor is.
% A kimenet egy vektor vagy egy matrix, bemenettol fuggoen, ahol
% az 1. oszlop a fok, a 2. a perc, a 3. a masodperc

  f = fix(x);
  p = fix((x-f) * 60);
  m = ((x-f)*60-p)*60;

  fpm = [f p m];
end
```

Ez a függvény tizedfokból számol át fok-perc-másodpercbe. Ez is működik akár vektorban adott bementtel is, több szög együttes átszámítására. A fix parancs mindig a 0 felé kerekít egészre. Azért célszerű ezt használni round helyett, hogy akár negatív szögekre is működjön az átváltás.

A MAPPING PACKAGE HASZNÁLATA

A fenti függvényeken megtanulhattuk, hogyan tudunk olyan saját függvényeket írni, amik többféle bemenetet is elfogadnak és többféle kimenetet engedélyeznek. A tizedfokból fok-perc-másodpercbe, radiánba és vissza átszámító függvényekre gyakran szükség van a geodéziai számítások során. A feladat gyakorisága folytán természetesen lehet olyan kész függvénykönyvtárat is találni, amiben a fentiek benne vannak. Octave esetében ez a Mapping csomag (package), a Matlabban pedig a Mapping Toolbox. Nézzük meg, hogy milyen függvényeket találhatunk ebben, és hogyan tudjuk ezeket használni!

Ellenőrizzük le, hogy telepítve van-e a Mapping package:

```
pkg list
```

Ha nincs telepítsük az Octave-Forge oldalról:

```
% mapping csomag telepítése internetről közvetlenül:
pkg install -forge mapping
```

Nézzük meg milyen parancsok vannak benne:

```
pkg describe -verbose mapping % kilistázza a mapping csomag parancsait
```

Töltsük be a csomagot!

```
pkg load mapping; % mapping csomag betöltése
```

A számunkra most fontos parancsok:

```
deg2rad;degtorad;rad2deg;radtodeg;
degrees2dm; degrees2dms; dm2degrees; dms2degrees;
```

Teszteljük a dms2degrees és a deg2rad függvényt! Kérdezzük le help-pel milyen bemenetet várnak!

```
L1degree = dms2degrees(L1)
Ldegree = dms2degrees(L)
Lrad = deg2rad(Ldegree)
```

A MEGOLDÁS ELMÉLETE

Az előző segédfüggvények elkészítése (vagy betöltése) után hozzá is láthatunk a megoldáshoz.

Az ábra alalpján a 3 meghatározandó szög értékre 6 egyenletet írhatunk fel a 6 mérési eredmény felhasználásával. Ez egy lineáris egyenletrendszert alkot.

```
% A megoldando linearis egyenlet rendszer
% L1 = X1;
% L2 = X2-X1;
% L3 = X3-X2;
% L4 = X2;
% L5 = X3-X1;
% L6 = X3;
```

Egy lineáris egyenletrendszer általános alakja a következő:

$$A \cdot x = b$$

Ennek a megoldása, ha egyértelmű a megoldás:

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

Jelenleg azonban 6 egyenletünk van 3 ismeretlenre. Ez egy túlhatározott egyenlet rendszer. Ilyenkor nincs egyértelmű megoldás, hanem a maradék eltérések négyzetösszegét minimalizáljuk.

$$\|A \cdot x - b\|^2 \rightarrow min.$$

Ez a célfüggvény. Egy függvénynek akkor lehet minimuma, ha az első deriált értéke zérus.

Nagyon leegyszerűsítve, ha az A és a b nem mátrix, illetve vektor lenne, hanem egy-egy skalár, akkor az $(A \cdot x - b)^2$ függvény deriváltja (felhasználva az összetett függvény deriválására vonatkozó szabályt a $2 \cdot (A \cdot x - b) \cdot A = 2 \cdot A \cdot A \cdot x - 2 \cdot A \cdot b$ lenne. Ha ez egyenlő nullával, akkor 2-vel le lehet osztani nyugodtan, így a megoldandó egyenlet a $A \cdot A \cdot x - A \cdot b = 0$ lenne. Mátrixok, illetve vektorok esetében a megoldandó egyenlet a következő lesz:

$$A^T \cdot A \cdot x - A^T \cdot b = 0$$

Ha ezt megoldjuk x-re, akkor a következő egyenletet kapjuk:

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

Ez a megoldás egységsúlyú esetben igaz, ha a méréseink különböző súlyúak, akkor a súlymátrixot is bele kell vennünk:

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

Ez tulajdonképpen a pszeudoinverzzel történő megoldás.

A MEGOLDÁS MATLAB/OCTAVE HASZNÁLATÁVAL

A korábban felírt lineáris egyenletrendszert írjuk fel $A \cdot x = b$ alakban, ahol A az ismeretlenek (X1, X2, X3) együtthatóit tartalmazza az egyes egyenletekben, b pedig a mérési eredményeket. Az A mátrixokt alakmátrixnak szokás nevezni a geodéziában. A fok-perc-másodper értékeket alakítsuk át tozedfokba a számításhoz!

```
% A megoldando linearis egyenlet rendszer
% L1 = X1;
% L2 = X2-X1;
% L3 = X3-X2;
% L4 = X2;
% L5 = X3-X1;
% L6 = X3;

 A = [1 0 0;
    -1 1 0;
    0 -1 1;
    0 1 0;
    -1 0 1;
    0 0 1]

b = fpm2fok(L)
% vagy
b = dms2degrees(L)
```

A megoldás:

```
X = inv(A'*A)*(A'*b)
```

Megjegyzés: ugyanez az eredmény megkapható a beépítet pszeudoinverz függvény alkalmazásával is, vagy az azzal egyenértékű \ jel használatával:

```
X = pinv(A)*b
```

vagy a legrövidebben:

```
X=A\b
```

Eredmény:

```
X =
   30.6742
   72.7774
  118.9826
```

Alakítsuk át a megoldást fok-perc-másodperccé!

```
Xfpm = degrees2dms(X)
```

Az eredmény:

```
Xfpm =

   30.0000   40.0000   27.0000
   72.0000   46.0000   38.7500
  118.0000   58.0000   57.2500
```

Ha nem lineáris egyenletet szeretnénk megoldani legkisebb négyzetek módszerével, akkor először linearizálni kellene az egyenletek egy kezdőérték körül (Tayor-sorba fejtéssel), és úgy kellene megoldani a feladatot.

Nézzük meg a súlyozott megoldást, amikor az L_1, L_2, L_3 méréseket μ=2”, az L_4, L_5, L_6 méréseket μ=4” középhibával jellemezhetjük! Vagyis az első 3 szöget pontosabban mértük, tehát nagyobb súllyal vesszük őket figyelembe. Ebben az esetben először a súlymátrixot kell felvennünk, ami az egyes mérések súlyát tartalmazó diagonálmátrix.

A súlyok és a középhibák kapcsoltát az alábbi aránnyal írhatjuk le:

$$p_1 : p_2 = \frac{1}{{m_1}^2} : \frac{1}{{m_2}^2}$$

Mivel ez egy arányosság az egészet beszorozhatjuk egy tetszőleges számmal, például a nevezők legkisebb közös többszörösével (least common multiple - lcm), hogy egész számokat kapjunk a súlyokra!

```
disp('Sulyozott megoldas')

m1 = 2; m2 = 4; % középhibák
c = lcm(m1^2,m2^2) % középhiba négyzetek legkisebb közös többszöröse
p1 = c/m1^2 % súlyok
p2 = c/m2^2 % súlyok
% súlymátrix
P = diag([p1 p1 p1 p2 p2 p2])
```

Az eredmény:

```
c2 = 16
p1 = 4
p2 = 1
P =  4     0     0     0     0     0
     0     4     0     0     0     0
     0     0     4     0     0     0
     0     0     0     1     0     0
     0     0     0     0     1     0
     0     0     0     0     0     1
```

A megoldás:

```
X2 = inv(A'*P*A)*(A'*P*b);
X2fpm = degrees2dms(X)
```

Az eredmény:

```
x2fpm =
   30.0000   40.0000   25.2293
   72.0000   46.0000   37.3268
  118.0000   58.0000   56.7561
```

EGYENES ILLESZTÉSE

A következő feladat egy egyenes illesztése lesz legkisebb négyzetek módszerével.

Adjuk meg az egyenes pontjait, mint mérési eredményeket. Egy adott egyenes 10 pontját „rontsunk el” mérési hibákkal!

Az egyenes egyenletét az $y = m \cdot x + b$ egyenlettel adjuk meg. Az egyenes két paramétere az y tengellyel vett metszete (b) és a meredeksége (m).

```
disp('Egyenes illesztése')
b = 8.765; m = 1.234;
f = @(x) b + m*x
figure(1);
h = ezplot(f,[0, 11])
set(h, 'LineWidth', 2)
```

Az egyenes 10 pontját számoljuk ki 1:10 közötti x koordináták esetében, majd adjunk hozzá mind x, mind y értékekhez egy normális eloszlású véletlen hibát (randn).

```
x = [1:10]'
y = f(x)
x = x + randn(numel(x),1)
y = y + randn(numel(y),1)
hold on;
plot(x,y,'r*')
```

Illesszük ezekre a pontokra a legkisebb négyzetek értelmében legjobban illeszkedő egyenest! A 10 pontra 10 egyenletet tudunk felírni, miközben két ismeretlenünk van, ez egy erősen túlhatározott egyenlet.

$$b + m \cdot x_1 = y_1$$
$$b + m \cdot x_2 = y_2$$
$$\dots$$
$$b + m \cdot x_{10} = y_{10}$$

A két paramétert adjuk meg a p vektorban, p(1)=b és p(2)=m.

Mátrixosan felírva $A \cdot p = y$:

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \dots & \dots \\ 1 & x_{10} \end{pmatrix} \cdot \begin{pmatrix} b \\ m \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_{10} \end{pmatrix}$$

Állítsuk elő az A alakmátrixot a Matlab-ban! Ehhez egy 10 elemű (amilyen hosszú az x vektor) egyesekből álló oszlopvektorhoz hozzá kell fűzzük az x értékeket tartalmazó vektort. A numel parancs visszadja egy mátrix vagy vektor elemeinek a

számát. A ones(n,m) parancs egy csupa egyesből álló mátrixot hoz létre, n sorral, m oszloppal.

```
A = [ones(numel(x),1) x]
```

Oldjuk meg az egyenletrendszert a legkisebb négyzetek elve szerint!

```
p = inv(A'*A)*A'*y
% vagy
p = A\y
```

Az eredmény (természetesen többszöri futtatásra a randn függvény miatt nem állandó), pl.:

```
p =
   7.3466
   1.4525
```

Ábrázoljuk:

```
b1 = p(1); m1 = p(2);
f1 = @(x) m1*x + b1
h = ezplot(f1, [0,11])
set(h,'Color','m','LineWidth', 2)
axis('equal');
print egyenes.jpg;
```