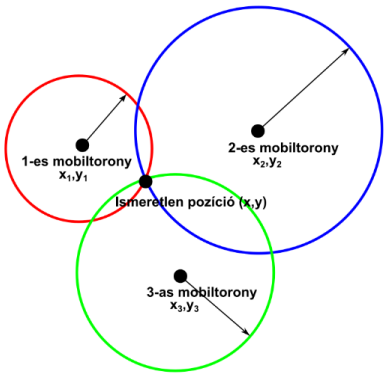
7. NEMLINEÁRIS EGYENLETRENDSZEREK MEGOLDÁSA

Az építőmérnöki problémák gyakran igénylik nemlineáris egyenletek megoldását, például amikor több nemlineáris egyenlet metszéspontját keressük. Ilyen példák jellemzően a különböző geodéziai pontmeghatározási feladatok (ívmetszés, előmetszés, hátrametszés stb.), de a rúdszerkezetek egyensúlyi egyenletei és a súlytámfalak tervezése során is nemlineáris egyenletrendszereket kell megoldanunk. Most nézzünk meg egy mobiltelefonnal történő pozíció meghatározási feladatot fix mobil adótornyokhoz képest, amikor megmérjük a telefon és a környező adótornyok távolságait. Ez tulajdonképpen egy ívmetszési feladat.

A mobiltelefonok pozíciójának meghatározásakor rendelkezésre áll az eszköz és a bázisállomások mért távolsága. A távolságok egy-egy kört határoznak meg a bázisállomások körül (lásd ábra). A körök másodfokú egyenletek, és két mért távolság esetén ezek metszéspontja adja a mobiltelefon lehetséges pozícióját. Az egyenleteket a következő implicit alakban adhatjuk meg, két ismert bázis esetén:



$$(x - x_1)^2 + (y - y_1)^2 = r_1^2$$
$$(x - x_2)^2 + (y - y_2)^2 = r_2^2$$

Amennyiben legalább 3 távolság, valamint a mobil tornyok koordinátái ismertek, az ismeretlen hely x,y koordinátái meghatározhatóak, de ez már egy túlhatározott feladat lesz 3 egyenlettel 2 ismeretlenre. 2 mért távolság esetében 2 lehetséges megoldást kapunk a pozícióra (a két metszéspontot). Most ez utóbbi esettel fogunk foglalkozni, amikor 2 egyenletünk és 2 ismeretlenünk van.

A két másodfokú algebrai polinomból álló kétváltozós egyenletrendszer megoldása visszavezethető egy 4. fokú polinom gyökeinek megtalálására is, de most az általános nemlineáris egyenletrendszer megoldásával fogunk foglalkozni.

EGYENLETRENDSZER VEKTOROS JELÖLÉSMÓDJA

A nemlineáris egyenletrendszerek általános megoldásához vezessük be a vektoros jelölésmódot, így egyszerűbb lesz dolgozni az egyenletekkel és a Matlab beépített függvényei is ilyen alakban igénylik a megadást.

Általában akkor tudjuk megtalálni egy egyenletrendszer megoldását, amikor az ismeretlenek száma megegyezik az egyenletek számával. Az egyenletrendszereket a következő alakban szokás megadni:

$$f_1(x_1, x_2, x_3, \dots, x_n) = 0$$
$$f_2(x_1, x_2, x_3, \dots, x_n) = 0$$
$$f_3(x_1, x_2, x_3, \dots, x_n) = 0$$
$$\vdots$$
$$f_n(x_1, x_2, x_3, \dots, x_n) = 0$$

A vektoros jelölésmódban a különböző változók megoldásait egy vektorban tároljuk. Legyen ez az $\boldsymbol{x}$ vektor, amelynek elemei: $\boldsymbol{x} = (x_1, x_2, x_3, \dots, x_n)$. A különböző egyenleteket is egy vektorban tároljuk ($\boldsymbol{f}$), amelynek az elemei: $\boldsymbol{f} = (f_1, f_2, f_3, \dots, f_n)$. Ezzel a jelöléssel az egyenletrendszert egyszerűen le tudjuk írni.

$$\boldsymbol{f}(\boldsymbol{x}) = 0$$

Egyváltozós esetben egy x_0 kezdeti értéket kell felvennünk. Többváltozós esetben egy $\boldsymbol{x_0}$ vektorban adjuk meg az összes változó kezdeti értékeit. Minél több változónk van, annál több jó kezdeti értéket szükséges meghatároznunk, ami nem mindig könnyű feladat. Gyakorlatban azonban kihasználhatjuk az adott probléma mérnöki ismereteit és ezek alapján gyakran tudunk adni jó közelítő értékeket akár többváltozós esetre is. Ha tudjuk ábrázolni az egyenleteket (például kétváltozós esetben), akkor az ábra alapján is meghatározhatunk kiinduló adatokat.

PÉLDA NEMLINEÁRIS EGYENLETRENDSZERRE

A feladatunk pozíció meghatározás lesz mobiltelefonnal. A példában tegyük fel, hogy csak 2 mobil adótoronytól ismerjük a távolságot, így a pozícióra két lehetséges helyet fogunk kapni. Az egyes bázisállomások koordinátáit és a távolságméréseket az alábbi táblázat foglalja össze (kilométer mértékegységben).

Bázis sorszám	**X koordináta** (x_i)	**Y koordináta** (y_i)	**Bázis-terminál távolság** (r_i)
1	1	1	5
2	10	8	8

Az egyenleteket a következő implicit alakban adhatjuk meg, ahol az *x,y* koordinátákat keressük:

$$(x-1)^2 + (y-1)^2 = 5^2$$
$$(x-10)^2 + (y-8)^2 = 8^2$$

Első lépésként rendezzük nullára az egyenleteket:

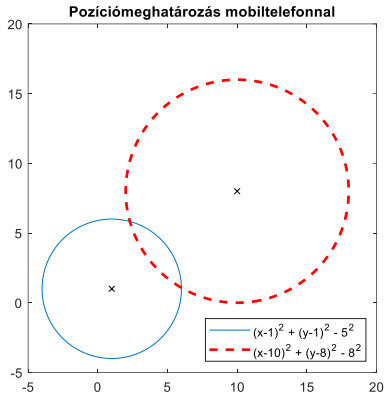
$$(x-1)^2 + (y-1)^2 - 5^2 = 0$$
$$(x-10)^2 + (y-8)^2 - 8^2 = 0$$

Ezek az egyenletek nem a szokásos explicit formában megadott függvények: $y = f(x)$, hanem implicit formában megadott görbék: $f(x,y) = 0$. Különböző Matlab parancsok használhatóak ezek megjelenítésére. Az explicit függvények megjelenítésére az **fplot** és az **ezplot**, az implicit formában adott görbékre pedig az **fimplicit** és **ezplot** használható. Az **fplot** parancs csak $y = f(x)$ alakú egyváltozós függvények ábrázolására szolgál, implicit alakban megadott görbéket nem lehet vele megjeleníteni.

egyenlet	*alak*	*ajánlott Matlab parancs*	*egyéb Matlab parancs (régebbi Matlab verziók és Octave esetén is használható)*
$y = f(x)$	explicit	**fplot**	**ezplot**
$f(x,y) = 0$	implicit	**fimplicit**	**ezplot**

Először ábrázoljuk a két implicit alakban adott függvényt az **`fimplicit`** paranccsal. A színek, vonaltípus stb. megadása a plot parancshoz hasonlóan történhet.

```
> clear all; clc; close all;
> f1 =@(x,y) (x-1).^2 + (y-1).^2 - 5^2;
> f2 =@(x,y) (x-10).^2 + (y-8).^2 - 8^2;
> figure(1);
> g1=fimplicit(f1,[-5 20 -5 20]); hold on;
> fimplicit(f2,[-5 20 -5 20],
  '--r','LineWidth',2);
> axis equal;
> legend('1. torony','2.
  torony','Location','SE')
> title('Pozíciómeghatározás
  mobiltelefonnal')
```



Hogyan tudjuk megtalálni a két implicit formában $(f(x,y)=0)$ megadott egyenlet metszéspontjait? Ezeket a görbéket többváltozós függvényként definiáltuk az ábrázoláshoz. Egyváltozós esetben $(f(x)=0)$ a nemlineáris függvény zérushelyeinek meghatározásához használtuk a Newton-módszert is. Többváltozós esetben is alkalmazható a Newton-módszer, némi átalakítással. Nézzük ezt meg!

TÖBBVÁLTOZÓS NEWTON MÓDSZER

Nézzük meg részletesen az egyik legismertebb módszert, a többváltozós Newton-módszert, a nemlineáris egyenletrendszerek megoldására, hogy könnyebben megértsük a megoldás egy lehetséges módját. Ez az egyváltozós eset megoldásából általánosítható. Egyváltozós esetben a Newton módszer a függvény linearizálásából volt levezethető:

$$f(x_{i+1}) \approx f(x_i) + f'(x_i) \cdot (x_{i+1} - x_i) = 0$$

Ebből adódott a Newton módszer iterációs képlete:

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

Többváltozós esetben az iterációs képlet nagyon hasonló:

$$\boldsymbol{x_{i+1}} = \boldsymbol{x_i} - \boldsymbol{J}(\boldsymbol{x_i})^{-1} \cdot \boldsymbol{f}(\boldsymbol{x_i})$$

ahol $\boldsymbol{f}(\boldsymbol{x_i})$ az egyenletrendszer egy oszlopvektorban, $\boldsymbol{x_i}$ a változók értékei egy vektorban, az $1/f'(x_i)$ helyett szereplő $\boldsymbol{J}(\boldsymbol{x_i})^{-1}$ pedig az n×n-es Jacobi mátrix inverze az $\boldsymbol{x_i}$ helyen. A Jacobi mátrix elemei az egyenletek parciális deriváltjai:

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \frac{\partial f_2}{\partial x_3} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

Pl. az alábbi egyenletrendszer Jacobi mátrixa:

$$\begin{matrix} 3x^2 + 2y + 1 = 0 \\ -x^3 - 5y + 2 = 0 \end{matrix} \quad \rightarrow \quad J(x) = \begin{bmatrix} 6x & 2 \\ -3x^2 & -5 \end{bmatrix}$$

A többváltozós iterációs formula számításához reciprok helyett inverze számítás szükséges, amit többnyire igyekszünk kerülni, mert lassú és numerikusan instabil, pontatlan. Hogyan tudjuk az $\boldsymbol{x_{i+1}}$-t inverz számítás nélkül meghatározni? Keressünk olyan alakot, ahol a lineáris egyenletrendszereknél tanult mátrix felbontásokat tudjuk használni! Rendezzük át az iterációs egyenletet!

$$\boldsymbol{x_{i+1} - x_i = -J(x_i)^{-1} \cdot f(x_i)}$$

Vezessük be a $\boldsymbol{\Delta x = x_{i+1} - x_i}$ jelölést és szorozzuk be mindkét oldalt $\boldsymbol{J(x_i)}$-vel! Így egy lineáris egyenletrendszert kapunk, amit meg tudunk oldani a mátrix felbontásokkal is!

$$\boldsymbol{J(x_i) \cdot \Delta x = -f(x_i)}$$

ahol $\boldsymbol{J(x_i)}$ egy *nxn*-es ismert mátrix, a Jacobi matrix $\boldsymbol{x_i}$ helyen, $\boldsymbol{-f(x_i)}$ egy ismert *nx1*-es vektor. A lineáris egyenletrendszerek megoldására használjuk most az $x = A\backslash b$ alakú megoldást, ami négyzetes esetben LU felbontást alkalmaz. Miután $\Delta\boldsymbol{x}$ értékét megkaptuk, második lépésben számítható $\boldsymbol{x_{i+1}}$ is.

Az egyes iterációkban a megoldandó feladat tehát:

- $\boldsymbol{J(x_i) \cdot \Delta x_i = -f(x_i)}$ lineáris egyenletrendszer megoldása Δx_i-re
- $\boldsymbol{x_{i+1} = x_i + \Delta x_i}$ számítása addig, amíg $\boldsymbol{f(x) \approx 0}$ vagy $\boldsymbol{\Delta x \approx 0}$ (amíg kisebb nem lesz egy megadott tolerancia értéknél).

TÖBBVÁLTOZÓS NEWTON-MÓDSZER MATLAB-BAN

Írjunk függvényt, ami megvalósítja a többváltozós Newton módszert a fentiek alapján! Legyen a függvény neve newtonsys. (Mentsük a fájlt a **newtonsys.m** fájlba!) A leállási feltétel legyen most az, hogy az egymást követő iterációs megoldások egy megadott toleranciánál kevésbé térjenek el egymástól. Illetve a ciklus álljon le akkor is, ha elértük a maximális iteráció számot. Bemenet lesz az egyenlet rendszer (*f*), a Jacobi mátrix (*J*), a kezdőértékek (x_0), a tolerancia (ε) és a leállási feltételt meghatározó maximális iterációszám. Két kimenete lesz a függvénynek, a megoldás és az iteráció szám.

```
> function [x1, n] = newtonsys (f, J, x0, eps, nmax)
>     dx = J(x0)\-f(x0)); % első iteráció
>     x1 = x0 + dx;       % első iteráció
>     n = 1;
>     while norm(x1-x0)>eps && n<=nmax
>         x0 = x1;
>         dx = J(x0)\-f(x0);
>         x1 = x0 + dx;
>         n = n + 1;
>     end;
> end
```

MEGOLDÁS NEWTON-MÓDSZERREL

a) Jacobi mátrix előállítása

A többváltozós Newton módszerhez, mint láttuk szükség lesz az eredeti egyenletek és a kezdőértékek mellett az egyenletek parciális deriváltjaira is a Jacobi mátrixhoz. Ezt szimbolikus számításokkal tudjuk előállítani. Lehet külön-külön a parciális deriváltakat is számítani a **diff** paranccsal, és abból összeállítani a Jacobi mátrixot, de a Matlab-ban van egy parancs a Jacobi mátrix közvetlen előállítására (**jacobian**). A jacobian parancs használatához definiáljuk most szimbolikusan az egyenletrendszert!

```
> %% Megoldás többváltozós Newton-módszerrel
> syms x y
> fs1 = (x-1).^2 + (y-1).^2 - 5^2
> fs2 = (x-10).^2 + (y-8).^2 - 8^2
> disp('Jacobi mátrix')
> js = jacobian([fs1; fs2])
> % [  2*x - 2,  2*y - 2]
> % [ 2*x - 20, 2*y - 16]
```

Definiáljuk függvényként a Jacobi mátrixot, szimbolikus kifejezés helyett! Ezt megtehetjük úgy, hogy bemásoljuk az egyes elemeket a megfelelő helyekre CTRL+C/CTRL+V használatával, vagy használhatjuk itt is a matlabFunction parancsot, ami szimbolikus kifejezésből függvényt állít elő (lehetnek olyan esetek, amikor ez nem működik).

```
> J=@(x,y) [2*x - 2,  2*y - 2; 2*x - 20, 2*y - 16] % vagy:
> J = matlabFunction(js)
```

b) Az egyenletrendszer és a Jacobi mátrix vektorizálása

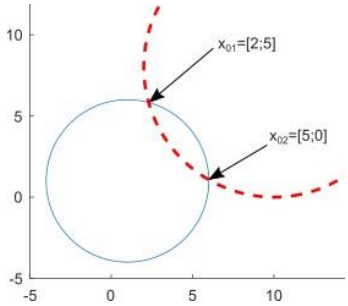
A megoldáshoz x,y változók helyett vektorváltozókat kell alkalmazzunk, ahol az ismeretlenek az $\boldsymbol{x}$ vektorban (x_1=x és x_2=y), az egyenletek pedig az $\boldsymbol{f}$ vektorban legyenek. $\boldsymbol{x} = \begin{pmatrix} x \\ y \end{pmatrix}; \boldsymbol{f} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$

A Jacobi mátrix esetében ugyanez szükséges! Vektorizáljuk az egyenleteinket és a Jacobi mátrixot is!

```
> % definiáljuk f, et és J-t is vektorváltozókkal
> f = @(x) [f1(x(1),x(2)); f2(x(1),x(2))];
> J = @(x) J(x(1),x(2));
```

c) Megoldás Newton-módszerrel

Oldjuk meg a problémát az általunk definiált **newtonsys** függvénnyel. Ehhez a **newtonsys.m** fájlnak ugyanabban a könyvtárban kell lennie ahová mentjük az aktuális script fájlt is. A numerikus megoldásokhoz, mint arról korábban már szó volt szükséges kezdőérték megadása. Minél közelebb van a kezdőérték a tényleges megoldáshoz annál gyorsabban meg tudjuk oldani a feladatot, annál valószínűbb, hogy konvergál a módszer. Az ábrából vegyünk kezdőértékeket, azokat az x és y koordinátákat, ahol a két görbe közelítőleg metszi egymást!




```
> % Oldjuk meg a problémát Newton módszerrel
> % kezdőértékek megadása az ábra alapján
> x01 = [2; 5] % egyik metszésponthozhoz
> x02 = [5; 0] % másik metszésponthoz
```

```
> [x1 i1] = newtonsys (f, J, x01, 1e-6, 100); % Az 1. megoldás
> [x2 i2] = newtonsys (f, J, x02, 1e-6, 100); % A 2. megoldás
```

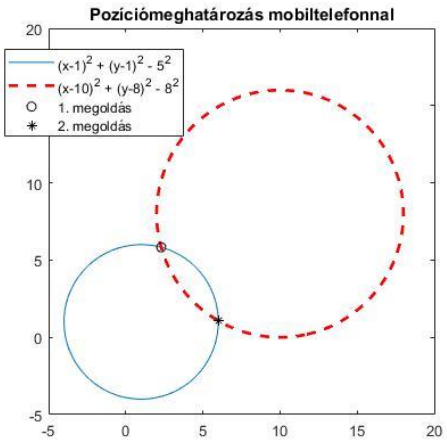
Az eredményeket írassuk ki formázva, és rajzoljuk is be az ábrába őket! Ellenőrzésnek helyettesítsük be a kapott eredményeket az implicit egyenletekbe, hogy tényleg 0-t kapunk-e (többváltozós esetben az eltérésvektor normáját szokás megadni, azaz az eltérésvektor hosszát)!

```
> disp('Lehetséges pozíciók Newton módszerrel:')
> fprintf('1. megoldás: %.4f,%.4f, iterációk: %d\n',x1, i1)
> fprintf('2. megoldás: %.4f,%.4f, iterációk: %d\n',x2, i2)
> % ellenőrzés
> norm(f(x1)) % 1.4648e-14
> norm(f(x2)) % 0
> plot(x1(1),x1(2),'ko')
> plot(x2(1),x2(2),'k*')
> legend('1. torony','2. torony',...
>     '1. megoldás','2. megoldás','Location','best')
```

A hiba a numerikus pontosságon belül 0-nak tekinthető.

Az eredmények:

```
Lehetséges pozíciók Newton módszerrel:
1. megoldás: 2.3005,5.8279, iterációk: 5
2. megoldás: 5.9995,1.0721, iterációk: 5
```



Nézzük meg a megoldást egy másik (x=1 és y=1) kezdőértékkel!

```
> % másik kezdőérték választása
> disp('Adjunk meg másik kezdőértéket!')
> x0 = [1; 1]
> [x3 i3] = newtonsys (f, J, x0, 1e-6,
  100);
> fprintf('3. megoldás: %.4f,%.4f,
  iterációk: %d\n',x3, i3)
```

Az eredmény:

```
Warning: Matrix is singular to working precision.
> In newtonsys at 2
  In mobiltornyok at 55
Warning: Matrix is singular, close to singular or badly scaled. Results may
be inaccurate. RCOND = NaN.
> In newtonsys at 7
  In mobiltornyok at 55

3. megoldás: NaN,NaN, iterációk: 2
```

Mi történt? Miért nincs megoldásunk? (NaN = Not a Number)

Azért, mert a Jacobi mátrix szinguláris. A geometriai ok, hogy a kezdeti érték egybe esik az egyik állomás koordinátájával, a kör középpontjával. Nem mindegy hogyan választjuk meg a kezdőértéket, ahogy az sem, milyen módszerrel oldjuk meg a feladatot, nem biztos, hogy konvergálni fog a megoldás.

MEGOLDÁS NUMERIKUSAN FSOLVE SEGÍTSÉGÉVEL

Természetesen a Matlab-nak is vannak beépített függvényei a nemlineáris egyenletrendszerek megoldásához. Nézzük meg először a numerikus eljárást használó **fsolve** parancsot! Az **fsolve**-ba több algoritmus is be van építve, ezek

alapján próbálja meg az optimális megoldást megtalálni a program minimalizálva a maradék eltérések négyzetösszegeit (egyváltozós esetben a lényegesen gyorsabb **fzero** is használható). Az **fsolve** esetében több opció is megadható, amivel gyorsítani lehet az eljárást, például megadhatunk Jacobi mátrixot, ami a többváltozós Newton módszerhez is kell. A kezdőértékek maradjanak a korábbiak. Az egyenletrendszert itt is vektoros formában kell megadni, viszont a Jacobi mátrix megadása nem követelmény. A két különböző megoldás megtalálásához itt is kétszer kell lefuttatnunk az **fsolve** parancsot, egyszer az egyik megoldáshoz közeli kezdőértékekkel, egyszer a másik megoldáshoz közeli kezdőértékekkel.

```
> % numerikus megoldás - fsolve
> x1 = fsolve(f,x01) % x1 = [2.3005;  5.8279]
> x2 = fsolve(f,x02) % x2 = [5.9995;  1.0721]
```

Természetesen ugyanazokat ez eredményeket kaptuk mint korábban. Numerikus ellenőrzés végett helyettesítsük vissza a kapott eredményeket az eredeti egyenletrendszerbe és nézzük meg a 0-tól való eltéréseket.

```
> % ellenőrzés
> norm(f(x1)) % 7.4621e-12
> norm(f(x2)) % 7.0300e-08
```

A hiba a numerikus pontosságon belül itt is 0-nak tekinthető. Az **fsolve** alkalmazásakor megadható a kívánt pontosság az **optimset** változó használatával, hasonlóan az **fzero**-hoz, a *'**TolFun**'* illetve a *'**TolX**'* változókkal. A '***TolFun***' használatával a függvényértékhez adható meg tolerancia, a '***TolX***' használatával pedig az egymást követő megoldások eltéréseire. Az alapérték mindkettőnél 10^{-6}. Ha a feladatot 10^{-9}-en pontossággal akarjuk megoldani, akkor így tehetjük meg:

```
> x1 = fsolve(f,x01,optimset('TolFun',1e-9))
```

Az **fsolve**-t több kimenettel is meghívhatjuk, így olyan módon is, hogy rögtön megadja a függvényértékeket is, anélkül, hogy vissza kellene helyettesíteni az ellenőrzéshez, illetve az **optimset** használatával az egyes iterációs lépések is kiírathatók:

```
> opt = optimset('TolFun',1e-9,'Display','iter');
> [X,FVAL] = fsolve(f,x01,opt)
```

Az eredménye:

```
X = 2.3005
    5.8279
FVAL = 1.0e-11 *
    0.5059
    0.5485
```

Természetesen az `fsolve` parancs is használható egyváltozós nemlineáris egyenlet gyökeinek a megkeresésére is, mint az **fzero**, viszont ez utóbbi sokkal hatékonyabban működik egyváltozós esetben (de többváltozós esetben nem alkalmazható). Hasonló módon megoldhatunk **fsolve** segítségével nem csak algebrai polinomokat, hanem egyéb trigonometriai, logaritmus, exponenciális stb. függvényeket tartalmazó egyenlet rendszereket is.

MEGOLDÁS SZIMBOLIKUSAN SOLVE SEGÍTSÉGÉVEL

A példa, amit megoldottunk egy algebrai polinom. Az algebrai polinomoknak általában létezik szimbolikus megoldása is. Ilyenkor egyszerre az összes megoldást

megkaphatjuk kezdőértékek megadására nélkül, éppúgy, mint az egyváltozós esetben. Itt most a parancs is megegyezik az egyváltozós esetben megismerttel, ez a **solve** parancs. A **solve**, szemben az **fsolve**-val, csak algebrai polinomok esetében használható.

A **solve** használatához szimbolikusan kell megadni az egyenleteket, ezt már megtettük a Jacobi mátrix előállítása során (fs1 és fs2), használjuk most ezeket itt is! Az eredmény tartalmazza mindkét megoldást egyszerre, egzakt alakban, szimbolikus struktúra formában (*xs*). A tényleges változók értékeit a struktúra neve (*xs*) után ponttal megadva lehet lekérdezni. Célszerű ezeket utána számmá alakítani a **double** paranccsal.

```
> xs = solve(fs1, fs2)
> %    x: [2x1 sym]
> %    y: [2x1 sym]
> xs.x, xs.y % x,y változók értékei szimbolikusan
> %  (77*39^(1/2))/260 + 83/20         69/20 - (99*39^(1/2))/260
> %  83/20 - (77*39^(1/2))/260         (99*39^(1/2))/260 + 69/20
> xs = [double(xs.x) double(xs.y)] % x,y változók értékei numerikusan
> %     5.9995    1.0721
> %     2.3005    5.8279
```

Mind a **solve**, mind az **fsolve** parancs ugyanazt a megoldást adta. Az `fsolve` esetében a vektorváltozós nullára rendezett egyenletrendszert kellett megadni bemenetként, és annyiszor kellett meghívni különböző kezdőértékkel, ahány megoldásunk van. A `solve` esetében a szimbolikusan definiált nullára rendezett egyenleteket kellett megadni. Ez utóbbi kezdőérték nélkül egy lépésben megadta az összes megoldást, viszont csak algebrai polinomoknál használható.

PARAMÉTERESEN MEGADOTT GÖRBÉK ÉS FÜGGVÉNYEK METSZÉSPONTJA

Paraméteres alakban vagy poláris koordinátákkal megadott görbékkel eddig még nem foglalkoztunk, pedig sok görbe nem írható fel a hagyományos derékszögű (x,y) koordinátákkal, csak poláris koordinátákkal vagy paraméteres alakban (például spirál). Vannak olyan görbék is, amelyek hagyományos derékszögű koordinátákkal (x,y) és paraméteres alakban is felírhatóak (pl. kör).

Kör egyenlete derékszögű koordinátákkal:

$$x^2 + y^2 = a^2$$

Kör paraméteres egyenlete:

$$x = a \cdot \cos(t)$$

$$y = a \cdot \sin(t)$$

Kör egyenlete poláris koordinátákkal:

$$r = a$$

Sok érdekes görbét találunk összegyűjtve például a következő oldalon: https://mathshistory.st-andrews.ac.uk/Curves/

Oldjunk most meg Matlab-ban egy olyan feladatot, amikor egy paraméteresen megadott görbe metszéspontját keressük egy függvénnyel.

Adott az alábbi paraméteres görbe:

$$X(t) = t \cdot \cos(2 \cdot \pi \cdot t)$$

$$Y(t) = t \cdot \sin(2 \cdot \pi \cdot t)$$

$$t \in [0,2]$$

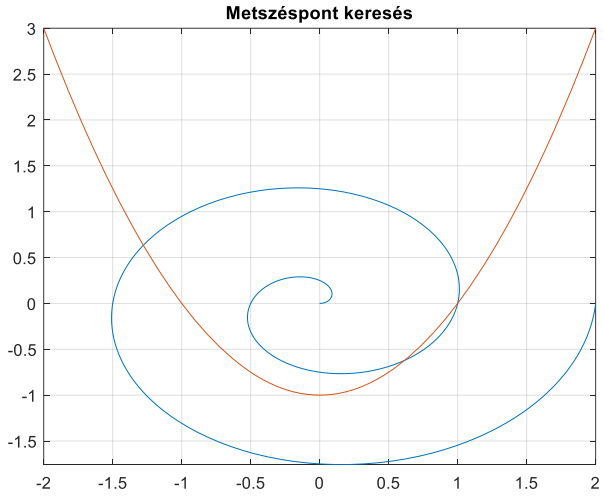
és egy parabola egyenlete:

$$y = x^2 - 1$$

Keressük meg az összes metszéspontjukat! Ehhez először ábrázoljuk a görbéket. Matlabban paraméteres görbét a következő módon ábrázolhatunk:

- fplot(X_függvény,Y_függvény,t_intervallum)

```
> % Ábrázolás
> clc; clear all; close all;
> % Paraméteres görbe definiálása
> xp = @(t) t.*cos(2*pi*t)
> yp = @(t) t.*sin(2*pi*t)
> XY = @(t) [xp(t),yp(t)] % spirál pontjainak koordinátái a paraméter
  függvényében
> figure(1); fplot(xp,yp,[0,2])
> % parabola megadása
> f = @(x) x.^2-1
> hold on; fplot(f,[-2,2])
> grid on; title('Metszéspont keresés')
```



A megoldást többféleképpen is megkaphatjuk. Az egyik megoldás, hogy behelyettesítjük a paraméteres egyenletekkel megadott X,Y értékeket a parabola egyenletébe és ezzel visszavezetjük a feladatot egy egyváltozós nemlineáris egyenlet megoldására. Először rendezzük nullára a parabola egyenletét:

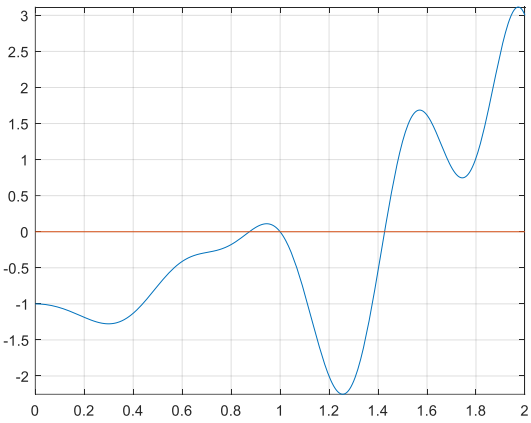
$$x^2 - 1 - y = 0$$

Behelyettesítve a paraméteres egyenleteket:

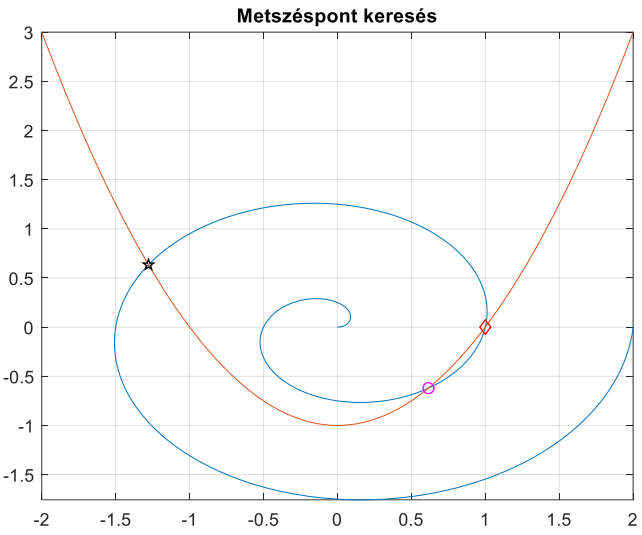
$$(t \cdot \cos(2 \cdot \pi \cdot t))^2 - 1 - (t \cdot \sin(2 \cdot \pi \cdot t)) = 0$$

Ezzel a feladatot visszavezettük egy egyváltozós nemlineáris egyenlet gyökeinek megkeresésére. Az új egyenletet felrajzoljuk egy másik ábrába és ebből tudunk kezdőértékeket venni a gyökök megkereséséhez.

```
> % Metszéspont keresés
> % parabola egyenlete 0-ra rendezve: x^2-1-y=0, behelyettesítve
> g = @(t) (t.*cos(2*pi*t)).^2-1-t.*sin(2*pi*t)
> % vagy egyszerűen használhatjuk a korábban definiált függvényeket:
> g = @(t) (xp(t)).^2-1-yp(t)
> figure(2); fplot(g,[0,2]);
> hold on; plot(xlim,[0,0]); grid on;
```




```
> % kezdőértékek az ábrából, megoldás
> t1 = 0.9; t2 = 1.1; t3 = 1.4;
> sol1 = fzero(g,t1) % 0.8744
> sol2 = fzero(g,t2) % 1
> sol3 = fzero(g,t3) % 1.4267
>
> % Metszespontok koordinátái és berajzolása
> M1 = XY(sol1) % 0.6159   -0.6207
> M2 = XY(sol2) % 1.0000   -0.0000
> M3 = XY(sol3) % -1.2782    0.6338
> figure(1)
> plot(M1(1),M1(2),'mo')
> plot(M2(1),M2(2),'rd')
> plot(M3(1),M3(2),'kp')
```



A feladat egy másik lehetsége megoldása, hogy egy 3 változós egyenletrendszerként oldjuk meg, ahol a három változó x,y,t. Az egyenletrendszer felírásakor figyeljünk oda, hogy 0-ra rendezett alakot kell megadnunk! Ebben az esetben mind a három változóhoz kell kezdőértéket választanunk, ezt megtehetjük úgy, hogy az első ábrán a kurzort a keresett metszéspont közelében a spirál fölé visszük és a Matlab által ott megjelenített körülbelüli értéket használjuk.

A nullára rendezett egyenletrendszer:

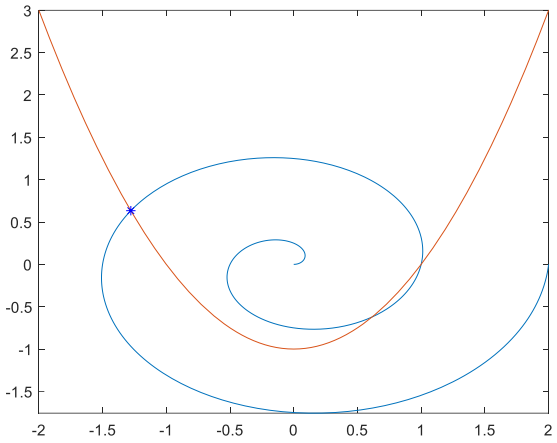
$$x - t \cdot \cos(2 \cdot \pi \cdot t) = 0$$

$$y - t \cdot \sin(2 \cdot \pi \cdot t) = 0$$

$$y - x^2 + 1 = 0$$

Az egyik metszéspont megkeresése ezzel a módszerrel Matlab-ban (kezdőértékek az első ábrából felvéve):

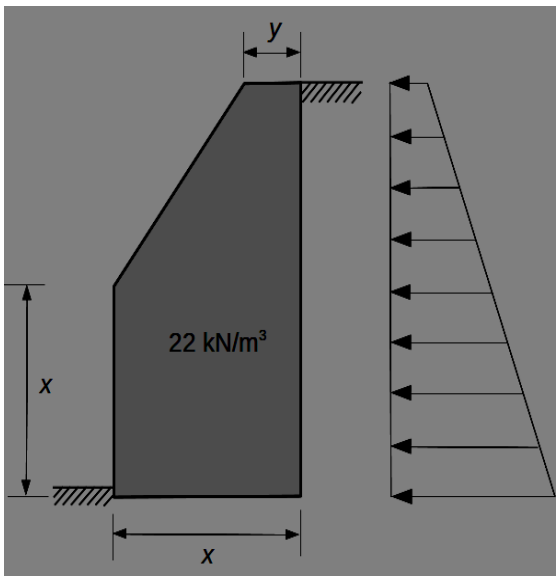
```
>  % megoldás 3 ismeretlenes egy. rsz.-ként
>  F = @(x,y,t) [x-t.*cos(2*pi*t);
>                        y-t.*sin(2*pi*t);
>                        y-x.^2+1]
>   % más megoldás lehet:
>   % F = @(x,y,t) [x-xp(t); y-yp(t); y-f(x)]
>   F = @(v) F(v(1),v(2),v(3))
>   v01 = [-1; 1; 1.5] % kezdőért. ábráról
>   sol1 = fsolve(F,v01)
>  %        -1.2782
>  %       0.63383
>  %        1.4267
>  plot(sol1(1),sol1(2),'b*')
```


3
2.5
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

GYAKORLÓ FELADATOK

1) Támfal méretezése kicsúszás és felborulás ellen

Adott az ábrán látható súlytámfal, ahol a tervezés során szeretnénk az *x* és *y* méreteket úgy meghatározni, hogy a támfal mind az elcsúszás, mind a felborulás ellen biztosítva legyen. A támfal magassága 2.6 méter, a talajnyomás megoszló terhe pedig 1.6 kN/m^2-től 5.4 kN/m^2-ig változik. Az egyenletek felírásához szükség van az önsúly parciális tényezőjére (0.9), a talajnyomáséra (1.4), a talaj és a beton közötti csúszó súrlódás együtthatóra (0.3), és feltételezzük, hogy az elfordulás a támfal alsó élének 1/10-e körül jön létre. Kiszámítható, hogy a támfal az elcsúszás és felborulás ellen akkor még éppen biztosított, ha teljesülnek az alábbi egyenletek:

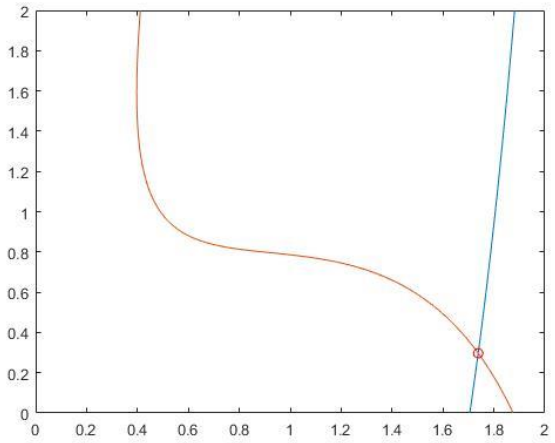


$$2x^2 - (2.6 - x)(x + y) = 4.29$$

$$6.24x^2 - (2.6 - x)(x - y)(7x - y) = 4.111$$

Ábrázoljuk az egyenleteket a [0,2]×[0,2] tartományon! Határozzuk meg az egyenletrendszer megoldását numerikusan és szimbolikusan is! A megoldást rajzoljuk be az ábrába, és ellenőrizzük az egyenletekbe behelyettesítve őket!

```
> % Súlytámfal méretezése elcsúszás és kifordulás ellen
> clc; clear all; close all
> %% A függvények ábrázolása
> f1 = @(x,y) 2*x.^2-(2.6-x).*(x+y)-4.29
> f2 = @(x,y) 6.24*x.^2-(2.6-x).*(x-y).*(7*x-y)-4.111
> figure(1);
> fimplicit(f1,[0,2,0,2]); hold on;
> fimplicit(f2,[0,2,0,2]);
> %% numerikus megoldás
> % a függvények vektorizálása
> f=@(x) [f1(x(1),x(2));f2(x(1),x(2))]
> % kezdőértékek az ábrából
> x0=[1.8; 0.3];
> [x, fval] = fsolve(f,x0)
> % x =
> %     1.7383
> %     0.2969
> % fval = 1.0e-10 *
> %     0.0504
> %     0.9412
> % berajzoljuk az ábrába
> %% szimbolikus megoldás
> plot(x(1),x(2),'ro')
> syms x y
> fs1 = 2*x.^2-(2.6-x).*(x+y)-4.29
> fs2 = 6.24*x.^2-(2.6-x).*(x-y).*(7*x-y)-4.111
> xs = solve(fs1, fs2)
> %    x: [4x1 sym]
```



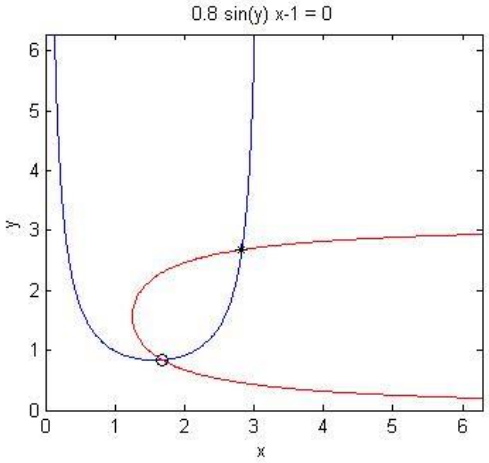
```
>  %    y: [4x1 sym]
>  xs = [double(xs.x) double(xs.y)] % x,y változók értékei numerikusan
>  %   -0.4219 + 0.0407i  -0.8806 - 0.0810i
>  %   -0.4219 - 0.0407i  -0.8806 + 0.0810i
>  %    1.7383 + 0.0000i   0.2969 + 0.0000i
>  %    2.3294 + 0.0000i  21.9170 + 0.0000i
>  xs1 = real(xs(3,:))
>  % xs1 = 1.7383    0.2969
```

A szimbolikus megoldásnál 4 megoldást is kaptunk, azonban ebből kettő komplex megoldás, egy pedig kívül esik a megadott tartományon. Marad a harmadik sorban lévő megoldás (a 0 komplex rész elhagyható a **real** parancs használatával).

2) Gyakorlásképp oldjunk meg egy nem algebrai polinomokból álló egyenletrendszert is a Matlab beépített numerikus módszerével és a Newton módszerrel is!

$$1.2\sin(x)\cdot y = 1$$
$$0.8\sin(y)\cdot x = 1$$

Ábrázoljuk az egyenletrendszert az $x = 0\ldots 2\pi$, $y = 0\ldots 2\pi$ tartományon, majd határozzuk meg az egyenletrendszer megoldásait! Ellenőrizzük a megoldásokat visszahelyettesítéssel, és az ábrába is rajzoljuk be a megoldásokat!




```
>  clc; clear all; close all;
>  f1 = @(x,y) 1.2*sin(x).*y-1
>  f2 = @(x,y) 0.8*sin(y).*x-1
>  f = @(x) [f1(x(1), x(2));
   f2(x(1), x(2))];
>  h1 = fimplicit(f1, [0 2*pi]);
>  hold on;
>  h2 = fimplicit(f2, [0 2*pi]);
>  set(h1,'Color','b');
   set(h2,'Color','r')
>  %% Megoldás fsolve segítségével
>  % első megoldás
>  opt = optimset('Display',
   'iter');
>  x01 = [1.6; 0.8]
>  x1 = fsolve(f, x01, opt) % x1 = 1.6810    0.8384
>  norm(f(x1)) %  8.0708e-11
>  plot(x1(1), x1(2), 'ko')
>
>  % második megoldás
>  x02 = [2.8; 2.7]
>  x2 = fsolve(f, [2.8 2.7], opt) % x2 = 2.8258    2.6834
>  norm(f(x2)) %  1.1585e-08
>  plot(x2(1), x2(2), 'k*')
>
>  %% Megoldás Newton módszerrel
>  syms x y
>  f1s = 1.2*sin(x)*y-1
>  f2s = 0.8*sin(y)*x-1
>
>  js = jacobian([f1s; f2s])
>  % [ 1.2*y*cos(x),   1.2*sin(x)]
>  % [   0.8*sin(y), 0.8*x*cos(y)]
>
```

```
> J=@(x,y) [1.2*y*cos(x),   1.2*sin(x);   0.8*sin(y), 0.8*x*cos(y)]
> J = @(x) J(x(1),x(2));
>
> [x1 i1] = newtonsys (f, J, x01, 1e-6, 100) % Az 1. megoldás
> [x2 i2] = newtonsys (f, J, x02, 1e-6, 100) % A 2. megoldás
>
> % x1 = [1.6810 0.8384], i1 = 4
> % x2 = [2.8258 2.6834], i2 = 3
> norm(f(x1)) %   1.1102e-16
> norm(f(x2)) %   3.1402e-16
```

A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

fimplicit	-	f(x,y)=0 implicit alakban megadott függvények ábrázolása
axis equal	-	Egyenlő beosztás a tengelyeken
jacobian	-	Jacobi mátrix kiszámítása (egyenletet parciális deriváltjai)
fsolve	-	Nemlineáris egyenletrendszerek megoldása numerikusan
solve	-	Algebrai polinomokból álló egyenletrendszer megoldása szimbolikusan
set	-	Grafikus elem tulajdonságainak beállítása (pl. Color, LineWidth)