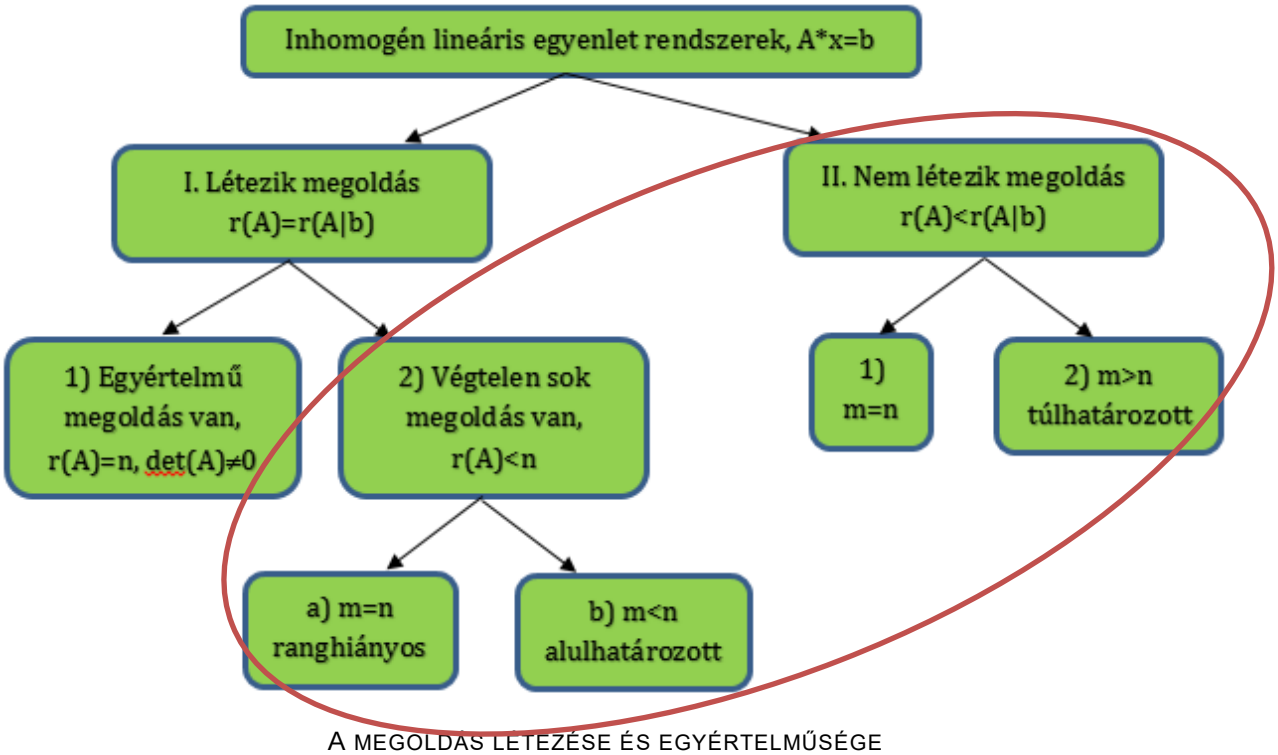
NINCS EGYÉRTELMŰ MEGOLDÁS

Eddig csak olyan eseteket vizsgáltunk, ahol létezett és egyértelmű volt az inhomogén lineáris egyenletrendszer megoldása. Többféle algoritmust is láttunk ezeknek a megoldására. A mérnöki gyakorlatban azonban előkerülnek a homogén lineáris egyenletrendszerek is, amire egy példát láttunk a főfeszültségek meghatározásakor, illetve azok az esetek is, amikor az inhomogén rendszernek nincs vagy végtelen sok megoldása van.



A MEGOLDÁS LÉTEZÉSE ÉS EGYÉRTELMŰSÉGE

Megjegyzés: a Matlabban elágazásokkal könnyen megvizsgálhatjuk, hogy van, vagy nincs megoldás és ha van, akkor egyértelmű-e vagy sem.

```
> if rank(A)==rank([A,b]) disp('Van megoldás')
>     if rank(A)==size(A,2) disp('Egyértelmű megoldás van')
>     else disp('Végtelen sok megoldás van')
>     end
> else disp('Nincs megoldás')
> end
```

VÉGTELEN SOK MEGOLDÁS VAN

Mi a helyzet akkor, amikor van ugyan megoldás, de nem egyértelmű? Például 2 egyenletünk van 3 ismeretlenre? Ilyenkor általában az egyik változó értéke tetszőlegesen felvehető és a másik kettő ennek függvényében számítható. Van megoldásunk, de nem egyértelmű, mivel végtelen sokféleképpen vehetjük fel az általunk megkötött változó értékét. Matematikai megközelítés szerint ez akkor lehetséges, ha a mátrix rangja megegyezik a kibővített mátrix rangjával viszont ez a rang kevesebb mint a mátrix oszlopainak a száma, azaz r(A)=r(A|b) és r(A)<n. Ez

előfordulhat ranghiányos esetben négyzetes együtthatómátrix esetén (m=n) is, vagy amikor kevesebb egyenletünk van, mint ismeretlen. Nézzünk előbbire egy példát:

$$x + y = 3$$
$$2x + 2y = 6$$

A fenti egyenletrendszerre ránézve látható, hogy a második egyenlet az első kétszerese, nem független tőle, tehát tulajdonképpen ugyanaz, mintha csak 1 egyenletünk lenne 2 ismeretlenre, végtelen sok megoldással (pl. x=1,y=2 vagy x=0,y=3, vagy x=10,y=-7 is kielégíti az egyenletrendszert).

Gyakoribb azonban az alulhatározott egyenletrendszer (m<n), amikor kevesebb egyenletünk van, mint ismeretlenünk. Ilyenkor végtelen sok megoldás van, mert a mátrix nullterének bármelyik n vektorát hozzáadhatjuk a megoldáshoz ($A \cdot n = 0 \rightarrow A(x + n) = b$). Ilyenkor megoldásnak a végtelen sok közül egyet szoktak kiválasztani, méghozzá általában (de nem mindig) a legkisebb normájú megoldást: $min|x|$. A legkisebb normájú megoldást matematikailag a következő formulával kaphatjuk meg:

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

Nézzünk egy példát alulhatározott egyenlet rendszerre, ahol 2 egyenletünk van 4 ismeretlennel:

$$7 \cdot x_1 + 2 \cdot x_2 + 2 \cdot x_4 = 1$$
$$x_1 + 8 \cdot x_2 + x_3 + 8 \cdot x_4 = 2$$

Mátrixos alakban felírva:

$$A = \begin{pmatrix} 7 & 2 & 0 & 2 \\ 1 & 8 & 1 & 8 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

Nézzük meg a megoldások számát!

```
> A = [7 2 0 2; 1 8 1 8], b = [1; 2]
> rank(A)     % 2
> rank([A b]) % 2
> size(A,2)   % 4
```

Az A mátrix rangja megegyezik a kibővített mátrix rangjával, tehát van megoldás, méghozzá végtelen sok, mivel r(A)= r(A|b) = 2 < n=4.

Oldjuk meg a feladatot, úgy, hogy a végtelen sok megoldás közül a legkisebb normájú megoldást válasszuk! Használjuk a megoldáshoz a fenti képletet!

```
> x = A'*inv(A*(A'))*b
> % xa =0.0745
> %     0.1195
> %     0.0127
> %     0.1195
> norm(A*x-b) % 0
> norm(x) % 0.1852
```

A megoldás hibátlan, visszahelyettesítéskor az eltérésvektor normája 0, a megoldásvektor hossza pedig 0.1852. Ez az összes lehetséges megoldásvektor közül a 'legrövidebb', legkisebb normájú.

Az előző órán láttuk, hogy az inverz számítás lassú és sokszor pontatlan is, ha lehet akkor el szokták ezt kerülni. Célszerű lenne itt is valamilyen mátrix felbontást alkalmazni, azonban a korábban megismert LU és Cholesky felbontás csak négyzetes

mátrixok esetében működik. Vannak olyan felbontások is, amelyek működnek nem négyzetes esetekben is, ilyen például a QR felbontás és az SVD felbontás.

QR FELBONTÁS

A QR módszer azt a tényt használja ki, hogy bármely A mátrix felbontható egy Q és R mátrix szorzatára ($A = Q \cdot R$), ahol Q egy (négyzetes) ortonormált mátrix, ahol $Q^{-1} = Q^T$, tehát $Q^T\, Q = I$ és R egy m×n-es felső háromszög mátrix. Legyen most A mátrix nem négyzetes (mxn-es, m≠n):

$$A = Q\, R$$

Az $A\, x = b$ egyenletrendszer megoldásához írjuk fel az $A \cdot x = b$ egyenletet a felbontott mátrixokkal:

$$Q\, R\, x = b$$

szorozzuk meg mindkét oldalt balról $Q^{-1} = Q^T$-tal, és az eredmény legyen B:

$$R\, x = Q^T\, b = B$$

Megoldás lépései:

- $B = Q^T\, b$ kiszámítása
- $R\, x = B$ megoldása, ahol már egyszerű visszahelyettesítést végezhetünk, hiszen R egy felső háromszögmátrix. Az eredmény a legkisebb hibájú megoldás lesz.

Oldjuk meg az előző feladatot QR felbontással (Matlab-ban a **qr** parancs)! Ellenőrizzük a felbontás helyességét és, hogy Q tényleg ortonormált-e! A megoldás során használjuk ki a Q és R mátrixok speciális voltát!

```
> %% QR felbontás
> [Q R] = qr(A)
> % Q = -0.9899   -0.1414
> %     -0.1414    0.9899
> %
> % R = -7.0711   -3.1113   -0.1414   -3.1113
> %           0    7.6368    0.9899    7.6368
> norm(A-Q*R)    % 8.8818e-16 (felbontás ellenőrzése)
> norm(Q'*Q)     %  1 (Q tényleg ortonormált, Q'*Q egységmátrix)
> B=Q'*b % Az új jobb oldal
> %    -1.2728
> %     1.8385
```

Végtelen sok megoldás esetén, négyzetes, ranghiányos együtthatómátrixnál az új B vektorban 0 elemek szerepelnek a nem független soroknál, előfordulhat azonban a numerikus pontatlanságok miatt, hogy 0 helyett egy nagyon kicsi szám szerepel, ezeket az értékeket ki kell nullázni a további megoldáshoz.

```
> B(abs(B)<1e-12)=0 % esetleges numerikus pontatlanság javítása
> % A megoldás
> opt.UT=true;
> x=linsolve(R,B,opt)
> % x = 0.0741
> %     0.2407
> %          0
```

```
> %          0
> norm(A*x-b) % 0
> norm(x) % 0.2519
```

QR felbontással nem ugyanazt a megoldást kaptuk, mint korábban, a megoldás itt is hibátlan, hiszen az eltérésvektor normája 0, a megoldásvektor hossza azonban nem 0.1852, hanem nagyobb, 0.2519. Feltűnő viszont, hogy van benne kettő darab nulla elem is. A QR felbontás nem a legkisebb normájú megoldást adja, hanem azt, amiben a legtöbb nulla elem található. Hogy melyik megoldást szeretnénk megkapni, az a konkrét feladattól függ. Előfordulhat olyan eset, amikor ez az előnyösebb, és olyan is, amikor a másik. Találhatunk vajon olyan felbontást, amivel a legkisebb normájú megoldást kapjuk meg? Nézzünk meg egy másik nem négyzetes esetben is használható felbontást, az SVD felbontást!

SVD FELBONTÁS

Nem négyzetes mátrixú lineáris egyenlet rendszereknél egy másik gyakran használt módszer az SVD felbontás. Az SVD felbontás angolul a szinguláris érték szerinti felbontás rövidítése (Singular Value Decomposition). Nézzük meg röviden mit jelent a sajátérték és a szinguláris érték egy *A* mátrixnál!

Sajátértékek (λ): Azt mondjuk, hogy a λ szám az (n×n-es) **A** mátrix sajátértéke, ha létezik olyan nem nulla **x** vektor, melyre $A\,x = \lambda\,x$. Az ilyen **x** vektorokat az **A** mátrix λ sajátértékhez tartozó sajátvektorainak nevezzük. A sajátértékeket megkapjuk az $|A - I\,\lambda| = 0$ karakterisztikus egyenlet megoldásával. Matlab-ban: sajátértékek: **`E = eig(A),`** sajátértékek és sajátvektorok: **`[V,D] = eig(A)`**

Szinguláris érték: Az m×n-es *A* mátrix szinguláris értékei az A^TA (nem nulla) sajátértékeinek négyzetgyökei. Matlab-ban: **`S = svd(A)`**. A szinguláris értékek száma egyenlő a mátrix rangjával.

Bármely (m×n-es) *A* mátrix felírható a következő formában:

$$A = U \cdot S \cdot V^T$$

ahol az *U* és *V* mátrixok ortonormáltak, azaz $U^{-1} = U^T, V^{-1} = V^T$ és *S* diagonálmátrix. Az U mátrix m×m-es, és oszlopvektorai az $A \cdot A^T$ sajátvektorai, a V mátrix n×n-es és oszlopvektorai az $A^T \cdot A$ mátrix sajátvektoraival egyeznek meg. Az S mátrix (m×n-es), főátlójában a $A^T \cdot A$ sajátértékeinek pozitív négyzetgyökei – az ún. szinguláris értékek – állnak, a többi elem nulla. Figyelembe véve, hogy U és V ortonormáltak és S diagonálmátrix, segítségükkel az A mátrix inverze/pszeudoinverze könnyen kiszámítható:

$$A^{-1} = (U \cdot S \cdot V^{-1})^{-1} = (V^{-1})^{-1} \cdot S^{-1} \cdot U^{-1}$$

Egyszerűsítve a pszeudoinverz:

$$A^{-1} = V \cdot S^{-1} \cdot U^T$$

Oldjuk meg az előző feladatot most SVD felbontással!

```
> %% SVD felbontás
> [U,S,V] = svd(A)
> % U = -0.3979   -0.9174
> %     -0.9174    0.3979
```

```
> % S = 12.1209         0         0         0
> %           0    6.3312         0         0
> % V = -0.3055   -0.9515    0.0368   -0.0015
> %     -0.6712    0.2130   -0.0933   -0.7039
> %     -0.0757    0.0629    0.9943   -0.0406
> %     -0.6712    0.2130   -0.0356    0.7092
> %  Ellenőrzések
> norm(A-U*S*V') % felbontás ellenőrzése: 3.0531e-15
> norm(U'*U) % U mxm-es ortonormált, szorzatuk 1
> norm(V'*V) % V nxn-es ortonormált, szorzatuk 1
> diag(S), sqrt(eig(A'*A)) % szinguláris érték A'A sajátértékeinek gyöke
> %    12.1209; 6.3312
> %    0.0000 + 0.0000i; 0.0000 + 0.0000i, 6.3312 + 0.0000i, 12.1209 + 0.0000i
```

Ranghiányos négyzetes A mátrixoknál előfordulhat, hogy a főátlóban a sajátértékeket tartalmazó S mátrixba numerikus pontatlanságból 0 helyett egy nagyon kicsi szám kerül be, ami még a numerikus pontatlanságon belül 0-nak tekintendő. Mielőtt az S pszeudoinverzét előállítjuk, egy általunk megadott toleranciánál kisebb értékeket 0-ra kell módosítanunk a későbbi megfelelő számítás érdekében.

```
> %  A pszeudoinverz előállítása
> S(abs(S)<1e-12)=0 % kinullázzuk az esetleges numerikus
  pontatlanságból adódó 0 közeli értékeket
> invS=(1./S)' % S inverze, a diagonál mátrix reciproka, transzponáltja
> %     0.0825       Inf
> %        Inf    0.1579
> %        Inf       Inf
> %        Inf       Inf
> invS(invS==Inf)=0 % ahol Inf (végtelen) elem volt, ott legyen 0
> %     0.0825         0
> %          0    0.1579
> %          0         0
> %          0         0
> invA=V*invS*U' % A mátrix pszeudoinverze
> %     0.1479   -0.0367
> %    -0.0088    0.0642
> %    -0.0066    0.0097
> %    -0.0088    0.0642
> x=invA*b %  A megoldás
> % x = 0.0745
> %     0.1195
> %     0.0127
> %     0.1195
> norm(A*x-b) % 4.9651e-16
> norm(x) % 0.1852
```

Az SVD felbontás ugyanazt a legkisebb normájú megoldást adta vissza, mint az első esetben láttuk, amennyiben a feladat ezt igényli, akkor célszerű az SVD felbontást használni!

MATLAB BEÉPÍTETT FÜGGVÉNYEK (LINSOLVE, \, PINV)

Természetesen itt is használhatjuk a Matlab beépített függvényeit, azonban, mint láttuk a különböző módszerek eltérő eredményt adnak, így nem mindegy melyik megoldást használjuk. Vigyázni kell arra is, hogy a beépített függvényekkel történő megoldások nem figyelmeztetnek arra, hogy nincs egyértelmű megoldás,

alulhatározott a mátrix és végtelen sok megoldás van, ezért megoldás előtt mindenképp célszerű elvégezni az egyértelműség vizsgálatát.

```
>  %% Beépített Matlab függvények
>  % QR felbontás - legtöbb nullát tartalmazó megoldás
>  xa = A\b % nem négyzetese esetben QR felbontás - legtöbb nulla
>  xb = linsolve(A,b)
>  % xa = xb = 0.0741
>  %           0.2407
>  %           0
>  %           0
>  % SVD felbontás - legkisebb normájú megoldás
>  xc = pinv(A)*b
>  % xc = 0.0745
>  %      0.1195
>  %      0.0127
>  %      0.1195
>  type pinv
```

A Matlab beépített **** (**mldivide**) és a **linsolve** parancsa QR felbontást használ nem négyzetes mátrixok esetében, a **pinv** parancs pedig a pszeudoinverzet számítja SVD felbontással. Alulhatározott esetben az **** (**mldivide**) és a **linsolve** a legtöbb nullát tartalmazó megoldást fogja adni, a **pinv** parancs pedig a legkisebb normájú megoldást.

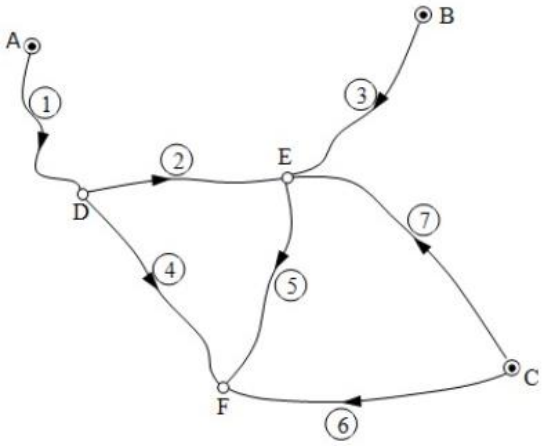
Emlékeztetőül négyzetes esetben a **** (**mldivide**) és a **linsolve** parancs LU vagy Cholesky felbontást használ, ezért négyzetes ranghiányos mátrixok esetében hibát fog jelezni a használata és nem kapunk megoldást. Ilyenkor csak a fent részletezett QR vagy SVD felbontás használható.

NINCS MEGOLDÁS (LEGKISEBB HIBÁJÚ MEGOLDÁS)

Gyakran előforduló eset a mérnöki gyakorlatban, amikor matematikailag nincs megoldása az egyenletrendszernek, mert túlhatározott rendszerünk van, azaz több mérésünk van, mint ismeretlenünk. A geodéziában szinte mindig ilyen feladatokkal találkozhatunk, hiszen a mérési hibák miatt mindig több mérés történik, mint, ami matematikailag tényleg szükséges lenne a feladat megoldásához. A gyakorlatban sokszor előfordulnak a kis mérési hibák mellett durva hibák is, ami miatt sokszor megoldhatatlan lesz a feladat, ha nincs fölös mérésünk.

Nézzünk most egy geodéziai példát, ahol szintezési hálózatban kell meghatározni az alappontok magasságát, úgy, hogy van több fölös mérésünk is! Nézzük meg az alábbi ábrán látható szintezési hálózat kiegyenlítését. Megmértük az 1-7 szintezési szakaszok magasság különbségeit, ismerjük 3 alappont (A,B,C) tengerszint feletti magasságát és keressük a további 3 alappont (D,E,F) magasságát! A szintezési vonalak hosszai nagyjából megegyeznek így most azonos súlyúnak tekintjük az egyes vonalak méréseit. Az ismert magasságok: H_A=183.506m, H_B=192.353m, H_C=191.880m, és a mért magasság különbségek:

$$H_D - H_A = H_D - 183.506 = +6.135 \rightarrow 1.szakasz$$
$$H_E - H_D = +8.343 \rightarrow 2.szakasz$$
$$H_E - H_B = H_E - 192.353 = +5.614 \rightarrow 3.szakasz$$
$$H_F - H_D = +1.394 \rightarrow 4.szakasz$$
$$H_F - H_E = -6.969 \rightarrow 5.szakasz$$
$$H_F - H_C = H_F - 191.880 = -0.930 \rightarrow 6.szakasz$$
$$H_E - H_C = H_E - 191.880 = +6.078 \rightarrow 7.szakasz$$



Írjuk fel az egyenletrendszert mátrixos alakban, a meghatározandó ismeretlenek: H_D, H_E, H_F!

$$\begin{pmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} H_D \\ H_E \\ H_F \end{pmatrix} = \begin{pmatrix} 6.135 + 183.506 \\ 8.343 \\ 5.614 + 192.353 \\ 1.394 \\ -6.969 \\ -0.930 + 191.880 \\ 6.078 + 191.880 \end{pmatrix} = \begin{pmatrix} 189.641 \\ 8.343 \\ 197.967 \\ 1.394 \\ -6.969 \\ 190.950 \\ 197.958 \end{pmatrix}$$

A fenti egyenletrendszer esetében 7 egyenletünk van 3 ismeretlenre (m>n), ez egy túlhatározott egyenletrendszer. Írjuk be Matlab-ba ezt az egyenletrendszert! Bevihetjük a mátrixokat kézzel is, de el is vannak mentve az értékek a **szintezes.txt** állományban.

```
> A = [1 0 0;-1 1 0;0 1 0;-1 0 1;0 -1 1;0 0 1;0 1 0]
> b = [189.641; 8.343; 197.967; 1.394; -6.969; 190.950; 197.958]
```

Vagy

```
> Ab = load('szintezes.txt')
> A = Ab(:,1:3)
> b = Ab(:,4)
```

Ha megnézzük az első ábrát a megoldás létezéséről, akkor láthatjuk, hogy, akkor nincs megoldásunk, ha a mátrix rangja kisebb, mint a kibővített mátrix rangja, azaz r(A)<r(A|b). Ez akkor fordul elő, ha a *b* vektor "kilóg" az *A* oszlopvektorainak teréből. Ritkábban négyzetes alakmátrix esetén is (m=n) előállhat ez az eset, leggyakrabban azonban túlhatározott egyenletrendszernél (m>n), amikor több egyenletünk van, mint ismeretlenünk. Megoldhatatlan négyzetes alakmátrixra példa lehet a következő:

$$x + y = 2$$

$$x + y = 3$$

ahol két egyenlet van két ismeretlennel, még sincs megoldás, mivel egymásnak ellentmondanak az egyenletek. Nézzük meg, hogy a mi példánkban mi a helyzet!

```
> rank(A)      % 3
> rank([A b])  % 4
> size(A)      % 7 sor 3 oszlop
```

Most a mátrix rangja 3, a kibővített mátrixé pedig 4, tehát r(A)<r(A|b), azaz nincs megoldás. A mátrix oszlopainak száma n=3, sorainak száma pedig m=7, m>n, tehát ez egy túlhatározott egyenletrendszer.

A kérdés, hogyan tudjuk megoldani a megoldhatatlan egyenletrendszert? A válasz, hogy nem egy hibátlan megoldást keresünk, hanem a legkisebb hibájú megoldást! Ezt a megoldást úgy találhatjuk meg, hogy a maradék ellentmondások négyzetösszegét minimalizáljuk: $\|A \cdot x - b\| \rightarrow min.$ A minimális hibájú közelítő megoldás matematikailag a következő lesz:

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

Matlab-ban:

```
> x = inv(A'*A)*A'*b  % 189.6153, 197.9588, 190.9830
```

A fenti megoldást nem szokták használni az inverz számítás időigényessége, pontatlansága miatt. Nézzük meg az előbb megismert két mátrix felbontást használva is az eredményeket!

QR FELBONTÁS

Oldjuk meg a szintezési feladatot is QR felbontással (Matlab-ban a **qr** parancs)! Most nem ellenőrizzük le a felbontás helyességét, csak az eltérésvektor normáját nézzük!

```
> [Q R] = qr(A)
> B=Q'*b % A jobb oldal
> % A megoldás
> opt.UT=true;
> x=linsolve(R,B,opt)
> %   189.6153
> %   197.9588
> %   190.9830
> norm(A*x-b) % 0.0506
```

Az eredmény: H_D=189.6153; H_E=197.9588; H_F=190.9830, az eltérésvektor normája pedig körülbelül 5 cm.

Amikor a feladatnak nincs megoldása és négyzetes az együtthatómátrixunk, ebben az esetben a QR felbontással történő legkisebb hibájú megoldás meghatározása nem fog működni Matlab-ban, csak az SVD felbontás.

SVD FELBONTÁS

Oldjuk meg a szintezési hálózat kiegyenlítését most SVD felbontással! Most sem ellenőrizzük le a felbontás helyességét, csak az eltérésvektor normáját nézzük!

```
> [U,S,V] = svd(A)
> %  A pszeudoinverz előállítása
> S(abs(S)<1e-12)=0 % kinullázzuk az esetleges numerikus
  pontatlanságból adódó 0 közeli értékeket
> invS=(1./S)' % S inverze
> invS(invS==Inf)=0 % ahol Inf (végtelen) elem volt, ott legyen 0
> invS*S % ellenőrzés
> invA=V*invS*U' % A mátrix pszeudoinverze
> invA*A % ellenőrzés
> x=invA*b %  A megoldás: x = 189.6153, 197.9588, 190.9830
> norm(A*x-b) % 0.0506
```

Túlhatározott esetben mind a QR, mind az SVD felbontással ugyanazt az eredményt kaptuk, mint az eredeti minimális hibájú megoldáshoz tartozó formulával.

MATLAB BEÉPÍTETT FÜGGVÉNYEK (LINSOLVE, \, PINV)

Természetesen itt is használhatunk Matlab beépített függvényeket . Mint már láttuk, a Matlab beépített **** (**mldivide**) és a **linsolve** parancsa QR felbontást használ nem négyzetes mátrixok esetében, a **pinv** parancs pedig a pszeudoinverzet számítja SVD felbontással. Nézzük meg a megoldásokat ezekkel is!

```
> x1 = A\b
> x2 = linsolve(A,b)
> x3 = pinv(A)*b
> norm(A*x1-b) % 0.0506
```

A beépített függvényekkel történő megoldások nem figyelmeztetnek arra, hogy egyértelmű megoldás helyett most a legkisebb hibájú megoldást kaptuk meg, és hogy mekkora ez a maradék eltérés, ezért mindenképpen szükség van ellenőrzésre is. Nézzük meg a kétféle algoritmus futásidejét is 10000 futtatás esetén!

```
> tic
> for i=1:1000
>     x1 = A\b;
> end
> toc % Elapsed time is 0.005924 seconds.
> tic
> for i=1:1000
>     x3 = pinv(A)*b;
> end
> toc % Elapsed time is 0.063876 seconds.
```

A QR felbontás egy nagyságrenddel gyorsabb volt, mint az SVD felbontás. Túlhatározott esetben célszerű a QR felbontást használó **x=A\b** alkalmazása, a hatékonyság/gyorsaság miatt. Alulhatározott esetben azonban a feladat jellegétől függően, ha a végtelen sok megoldás közül a legkisebb normájúra van szükség, akkor az SVD felbontást használó **x=pinv(A)*b** megoldást alkalmazzuk, ha a legtöbb nullát tartalmazó lenne ideális, akkor pedig a QR felbontást használó **x=A\b** parancsot alkalmazzuk. A **linsolve** parancs alkalmazása akkor célszerű, ha előzetes ismereteink vannak a mátrix speciális voltáról (pl. háromszögmátrix, szimmetrikus, pozitív definit), mivel itt van lehetőségünk ezeket opcióként megadni. Mivel a **** (**mldivide**) négyzetes esetben LU vagy Cholesky felbontást használ, ezért négyzetes ranghiányos mátrixok esetében hibát fog jelezni a használata és nem kapunk megoldást. Ilyenkor a fent részletezett SVD felbontással kaphatjuk meg a megoldást.

ITERATÍV MÓDSZEREK (JACOBI, GAUSS-SEIDEL)

Lineáris egyenletrendszereket megoldhatunk direkt és iteratív módszerekkel is. Amiket eddig néztünk eddig, a mátrix felbontások, azok a direkt megoldások voltak. Nézzük meg mikor lehet célszerű nem direkt, hanem iteratív megoldásokat használni!

Nézzünk most egy másik lineáris egyenletrendszerre vezető példát! Határozzuk meg az egyes vízkezelő reaktorokban a koncentráció értékét az alábbi kapcsolás esetén, tökéletes keveredést (a koncentráció azonos a reaktortér minden pontjában) feltételezve.

A mérlegegyenletek:

$6\,c_1 - c_3 = 50$

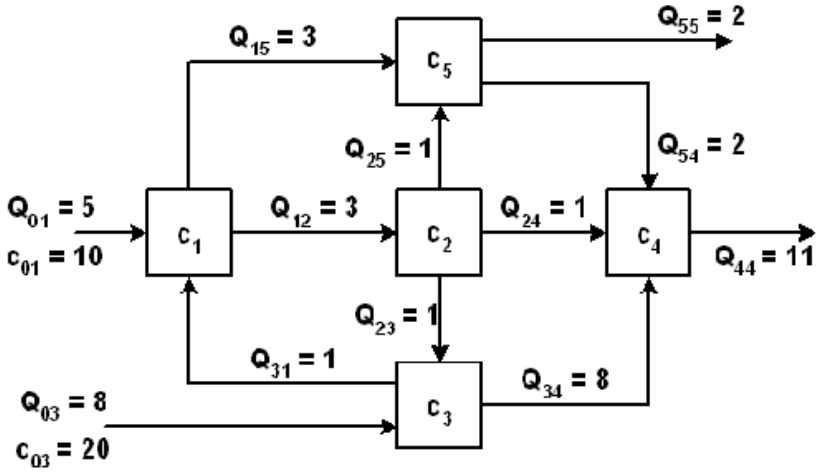
$-3\,c_1 + 3\,c_2 = 0$

$-c_2 + 9\,c_3 = 160$

$-c_2 - 8\,c_3 + 11\,c_4 - 2\,c_5 = 0$

$-3\,c_1 - c_2 + 4\,c_5 = 0$

Az egyenletrendszer mátrixos alakja:



$$A = \begin{pmatrix} 6 & 0 & -1 & 0 & 0 \\ -3 & 3 & 0 & 0 & 0 \\ 0 & -1 & 9 & 0 & 0 \\ 0 & -1 & -8 & 11 & -2 \\ -3 & -1 & 0 & 0 & 4 \end{pmatrix}; \qquad b = \begin{pmatrix} 50 \\ 0 \\ 160 \\ 0 \\ 0 \end{pmatrix}$$

Először vizsgáljuk meg hogy van-e megoldása a feladatnak és egyértelmű-e? Ehhez töltsük be az A mátrixot és b vektort tartalmazó vizkezeles.txt fájlt (vagy beírhatjuk kézzel is a mátrixokat)!

```
> Ab = load('vizkezeles.txt'), A = Ab(:,1:end-1), b = Ab(:,end)
> rank(A), rank(Ab) % 5, 5
> size(A,2) % 5
```

A mátrix rangja (5) megegyezik a kibővített mátrix rangjával is és az A mátrix oszlopainak a számával is, tehát létezik és egyértelmű a megoldás.

Az alakmátrixot vizsgálva megfigyelhetjük, hogy nagyon sok a nulla elem benne, ezeket ritka mátrixoknak nevezzük. Azokban az esetekben, ahol sok a nulla az alakmátrixban és a főátlóban lévő elemek dominálnak sokkal hatékonyabb a direkt megoldás helyett iteratív módszereket használni (és itt most nem az ilyen kis méretű mátrixoknál lényeges a hatékonyság, hanem a több ezerszer több ezres méreteknél!). Eddig a lineáris egyenlet rendszerek direkt megoldási módszereit vizsgáltuk, ahol a megoldást az egyenletekkel végzett elemi átalakításokkal kaptuk. Iteratív módszerek esetében meg kell becsülni minden változónak egy kezdőértéket és ezt használhatjuk utána egy iteratív eljárásában, hogy pontosítsuk az eredményt. A módszer hasonlít a nemlineáris egyenleteknél alkalmazható fixpont módszerhez. Az egyenleteket át kell alakítani explicit formába, minden változót kifejezve a többi változó függvényében.

$$\begin{matrix} a_{11}x_1 + a_{12}\,x_2 + a_{13}\,x_3 = b_1 \\ a_{21}x_1 + a_{22}\,x_2 + a_{23}\,x_3 = b_2 \\ a_{31}x_1 + a_{32}\,x_2 + a_{33}\,x_3 = b_3 \end{matrix} \quad \rightarrow \quad \begin{matrix} x_1 = \big(b_1 - (a_{12}\,x_2 + a_{13}\,x_3)\big)/a_{11} \\ x_2 = \big(b_2 - (a_{21}x_1 + a_{23}\,x_3)\big)/a_{22} \\ x_3 = \big(b_3 - (a_{31}x_1 + a_{32}\,x_2)\big)/a_{33} \end{matrix}$$

Az iterációs folyamatban először megbecsüljük a kezdőértékeket, majd ezeket a jobb oldalba behelyettesítve újabb értékeket kapunk x-re. A második iterációban ezeket az új értékeket helyettesítjük be a jobb oldalba, és így megyünk tovább, amíg a kívánt pontosságot el nem érjük, amikor a két egymást követő iterációban kapott megoldás közötti különbség kisebb az előre megadott tolerancia értéknél.

Az iterációs képlet a fenti explicit egyenletrendszer általános alakja lesz (ez tulajdonképpen a Jacobi módszer iterációs képlete):

$$x_i = \frac{1}{a_{ii}} \left(b_i - \Big(\sum_{\substack{j=1 \\ j \neq i}}^{j=n} a_{ij}\,x_j\Big) \right)$$

Kétféle speciális iterációs módszert nézünk most meg, a Jacobi iterációt és a Gauss-Seidel iterációt. A kettő között a különbség abban van, hogy mikor használjuk fel az újonnan kiszámolt értékeit az ismeretleneknek. Jacobi módszernél az ismeretlenek becsült értékei, amiket az explicit egyenletrendszer jobb oldalán használunk egyszerre kerülnek frissítésre minden iterációs lépés végén, vagyis egy kezdőérték készlettel kiszámoljuk az összes új értéket és utána megyünk tovább. A Gauss-Seidel módszernél a már kiszámolt ismeretlenek még az iterációs lépésen belül frissítésre kerülnek a további ismeretlenek kiszámolása során. Ez azt jelenti, ha egy iteráción belül pl. már kiszámoltunk egy új x_1-et, akkor ezt már x_2, x_3,... számításánál felhasználjuk. Ez utóbbi éppen ezért általában gyorsabban konvergál.

Az iterációs képletet előállíthatjuk mátrixos alakban is, amivel a későbbiekben könnyebben tudunk dolgozni. Alakítsuk át az $A \cdot x = b$ egyenletrendszert iterációs formulává! Adjunk hozzá, majd vonjunk is ki a baloldalból $B \cdot x$-et, ahol *B* mátrix megválasztásától függ majd az iteráció típusa (Jacobi vagy Gauss-Seidel).

$$A \cdot x = b$$

$$B \cdot x + A \cdot x - B \cdot x = b$$

$$B \cdot x + (A - B) \cdot x = b$$

Rendezzük át úgy, hogy mindkét oldalon legyen x:

$$B \cdot x = -(A - B) \cdot x + b$$

Szorozzuk meg balról B^{-1} mátrixxal:

$$x = -\,B^{-1}(A - B)\,x + B^{-1}b = -\,B^{-1}A\,x + B^{-1}B\,x + B^{-1}b$$

Emeljük ki x-et a jobb oldalon:

$$x = (I - B^{-1}A)x + B^{-1}b$$

A fentiek alapján írhatjuk fel az iterációs képletet a *(k+1).* iterációra:

$$x^{(k+1)} = (I - B^{-1}A)x + B^{-1}b$$

legyen $AI = (I - B^{-1}A)$ és $bi = B^{-1}b$, ekkor az iteráció képlete felírható a következő alakban:

$$x^{(k+1)} = AI \cdot x^{(k)} + bi$$

Ebben az általános alakban egységesen megadható a Jacobi módszer és a Gauss-Seidel módszer is. Eltérés csak a *B* mátrixban lesz, amivel *AI*-t és *bi*-t számoljuk. Jacobi módszer esetében *B* mátrix az *A* mátrix főátlója - **diag(diag(A))** (kifejtve az iteráció képletét így ugyanazt kapjuk, mint az explicit alakból meghatározott képlet, ahol mindig a főátló elemeivel (a_{ii}) kellett osztani), Gauss-Seidel módszer esetében pedig *B* mátrix az *A* mátrix alsó háromszögmátrixa - **tril(A)**.

Az iterációs megoldást elsősorban a diagonálisan domináns esetekben lehet hatékonyan alkalmazni, ez azt jelenti, hogy a főátlóban lévő elem abszolút értéke nagyobb, mint az abban a sorban lévő összes többi elem abszolút értékének az összege (a főátlóban lévő elemek dominálnak). A diagonális dominancia elégséges, de nem szükséges feltétele a konvergenciának. Nézzük meg, hogyan valósíthatjuk meg a Jacobi és a Gauss-Seidel iterációt Matlabban!

ITERÁCIÓS MÓDSZEREK MATLABBAN

Nézzük meg, hogyan lehet a kétféle iterációs módszert egységesen leprogramozni Matlabban, az alapértelmezett módszer legyen a Jacobi módszer! Nézzük meg az iterativ.m fájlt!

```
function [x,i,X]=iterativ(A,b,e,imax,method)
% A*x=b Lineáris egyenletrendszer megoldása iteratív módon,
% Jacobi vagy Gauss-Seidel módszerrel (alapértelmezett: Jacobi).
% Bemenet: A, b, e-hibahatár, imax - maximális iteráció szám
% method - 'Jacobi' vagy 'GaussSeidel' (nem kötelező megadni)
% Kimenet: x-megoldás, i -iteráció szám, X-iterációs lépések

% Módszer meghatározása
    if nargin==4; method='Jacobi'; end % ha csak 4 bemenet van
    switch method
        case 'Jacobi'
            B = diag(diag(A));  % Az A mátrix főátlója mátrixban
        case 'GaussSeidel'
            B=tril(A);  % Az A mátrix alsó háromszög
        otherwise % Jacobi
            B = diag(diag(A)); % Az A mátrix főátlója mátrixban
    end
% Első iterációs lépés
n = size(A,1);
Ai=eye(n)-inv(B)*A;
bi = inv(B)*b;
x0=ones(size(b)); % kiinduló érték
i=1; x1=Ai*x0+bi; % Első iteráció
X=[x0,x1];
% Iterációs ciklus
while i<=imax && norm(x1-x0)>e
    x0=x1;
    x1=Ai*x0+bi;
    X=[X x1];
    i=i+1;
```

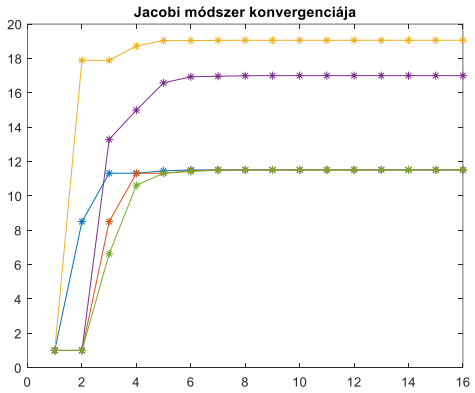
```
>   end
>   x=X(:,end); % megoldás
>   end
```

Bemenet az *A* mátrix, a *b* vektor, *e* a megengedett eltérés két egymást követő iteráció között, a maximális iteráció szám (*imax*) és a módszer (method), ami vagy `'Jacobi'` vagy `'GaussSeidel'` lehet. Alapértelmezett módszer a Jacobi módszer, azaz nem kötelező megadni az utolsó bemenetet. Ezt vizsgálja a **nargin** változó (Number of function input arguments), a megadott bemenő paraméterek számát. Ha csak 4 változót adott meg valaki, akkor a módszer alapértelmezetten Jacobi módszer lesz. A *B* mátrix Jacobi módszer esetében az *A* mátrix főátlója, Gauss-Seidel módszer esetében pedig az *A* mátrix alsó háromszögmátrixa lesz!

A megoldáshoz elő kell állítani az *AI* mátrixot, a *bi* vektort és meg kell adni a kezdő x_0 vektort, ami most egységvektor lesz. Az első iteráció után addig folytatja a program az iterációkat, amíg két egymást követő iteráció között az eltérés egy megadott toleranciánál kisebb, vagy el nem értük a maximális iteráció számot. Három kimenetünk van, a megoldás – x, az iterációk száma – i, és az összes iterációs lépés során kapott megoldás vektorok összessége az X mátrixban, ahol egymás melletti oszlopokban vannak az egymást követő iterációs lépések elemei.

Először Jacobi módszerrel oldjuk meg a feladatot!

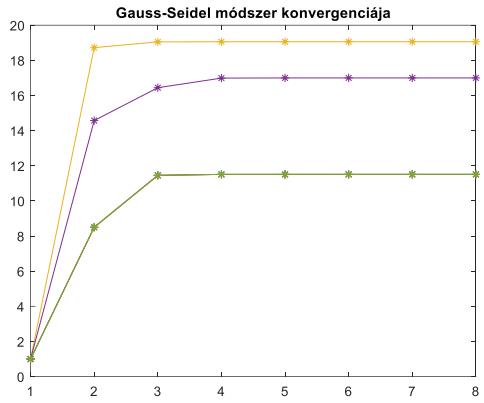
```
>   %% Jacobi módszer
>   [x,i,X] = iterativ(A,b,1e-6,100,'Jacobi')
>   % Ellenőrzés (relatív hiba)
>   h1 = norm(A*x-b);
>   fprintf('Az iterációk száma: %d, relatív hiba: %g \n',i,h1)
>   %Az iterációk száma: 15, relatív hiba:
    2.10763e-06
>
>   %  Grafikus megjelenítés
>   figure(1); plot(X','*-')
>   title('Jacobi módszer konvergenciája')
```



Utána oldjuk meg Gauss-Seidel módszerrel is!

```
>   %% Gauss-Seidel módszer
>   [x,i,X] = iterativ(A,b,1e-
    6,100,'GaussSeidel')
>   % Ellenőrzés (relatív hiba)
>   h1 = norm(A*x-b);
>   fprintf('Az iterációk száma: %d, relatív hiba: %g \n',i,h1)
>   % Az iterációk száma: 7, relatív hiba: 1.28853e-08
```

```
>
> % Grafikus megjelenítés
> figure(2);plot(X','*-');
> title('Gauss-Seidel módszer
  konvergenciája')
```



A megoldásból látjuk, hogy míg a Jacobi módszernek 15 iterációra volt szüksége, addig a Gauss-Seidel módszernek elegendő volt 7.

A Matlab-nak egyébként van egy saját függvénye iteratív megoldásra, ez a **gmres** parancs. Lásd:

```
> [x, flags, relres, iter, resvec] =
  gmres(A,b)
> figure(3); plot(resvec,'b*-') % hibák kirajzolása az iteráció során
```

Illetve ritka mátrixok tárolására van egy saját formátuma, amit nagyméretű ritka mátrixoknál érdemes használni:

```
> AS=sparse(A)
```

A FEJEZETBEN HASZNÁLT ÚJ FÜGGVÉNYEK

rank	-	Mátrix rangja
lu	-	LU felbontás
linsolve	-	Lineáris egyenletrendszer megoldása kiegészítő opciókkal (pl. alsó/felső háromszögmátrix, szimmetrikus, pozitív definit). Általános négyzetes mátrix esetén LU felbontást használ.
pascal	-	Előállíthatjuk a binomiális együtthatókat tartalmazó szimmetrikus Pascal mátrixot
diag	-	Kivehetjük egy mátrix főátlójából az elemeket vagy egy vektorból csinálhatunk vele diagonális mátrixot
min	-	Egy vektor legkisebb eleme
max	-	Egy vektor legnagyobb eleme
chol	-	Cholesky felbontás
norm	-	Vektor/mátrix normája (’hossza’)
tic, toc	-	Időmérés kezdete, vége
\ vagy mldivide	-	Általános lineáris egyenletrendszer megoldása (négyzetes mátrix esetén LU vagy Cholesky felbontással)
qr	-	QR felbontás
svd	-	SVD felbontás
pinv	-	Pszeudoinverz számítás SVD felbontással
type	-	Szöveges fájl tartalmának képernyőre írása
tril	-	Egy mátrix alsó háromszögmátrixa
nargin	-	Függvény megadott bemenő paramétereinek a száma
gmres	-	Lineáris egyenletrendszer iteratív megoldása
sparse	-	Ritka mátrixok tárolása