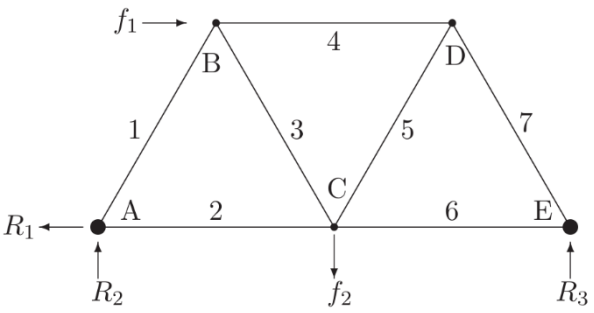
5. LINEÁRIS EGYENLETRENDSZEREK

A lineáris egyenletrendszerek kiemelt jelentőséggel bírnak az alkalmazott matematika és általában a numerikus matematika területén. Lineáris egyenletrendszer megoldására vezet a műszaki alkalmazásokban az a gyakorlat, hogy a mérnök többnyire lineáris fizikai modelleket alkalmaz. Nem csupán azért mert tudatában van annak, hogy a matematikai modellt így könnyebb lesz megoldani, hanem azért is mert sok fizikai jelenséget egy adott állapot környezetében valóban lineárisnak tekinthetünk, például a Hook-törvény. A numerikus matematikában, mint majd később látni fogjuk, nagyon sok numerikus módszer alkalmazása visszavezethető lineáris egyenletrendszer megoldására, például az interpoláció problémája. A lineáris egyenletrendszerek kezelésében alapvető fontosságú lesz a mátrix algebra alkalmazása, amely kimondottan kedvez a numerikus számítógépi megoldások elvégzésének.

Nézzünk most egy példát statika témaköréből. Vizsgáljuk meg az ábrán látható egyenlő oldalú háromszögekből álló rácsos tartóra ható erőket! A csomóponti módszert használva számítsuk ki a rúderőket! Használjuk a $\cos(60°) = 0.5$ értéket és a $\sin(60°)$ helyett a $\frac{\sqrt{3}}{2} \approx 0.8660$ közelítést ($f_1 = 1000N$ és $f_2 = 5000N$).



Az erők vetületi összegének és tetszőleges pontra számított nyomatékösszegének zérust kell eredményezni. Vetületi egyensúlyi egyenletek alapján: $R_1 - f_1 = 0$. Az *E* pontra felírt nyomatéki egyensúlyi egyenlet alapján $2\,R_2 + 0.866\,f_1 - f_2 = 0$. E kettőből kifejezhető R_1, R_2 reakció: $R_1 = f_1,\ R_2 = -0.433\,f_1 + 0.5\,f_2$. A csomópontokra ható erőket vizsgálva a következő lineáris egyenletrendszert írhatjuk fel, ahol T_i-vel jelöltük a rúderőket:

A jelű csomópont, x irányú erők: $0.5\,T_1 + T_2 = R_1 = f_1$

A jelű csomópont, y irányú erők: $0.866\,T_1 = -R_2 = 0.433\,f_1 - 0.5\,f_2$

B jelű csomópont, x irányú erők: $-0.5\,T_1 + 0.5\,T_3 + T_4 = -f_1$

B jelű csomópont, y irányú erők: $0.866\,T_1 + 0.866\,T_3 = 0$

C jelű csomópont, x irányú erők: $-T_2 - 0.5\,T_3 + 0.5\,T_5 + T_6 = 0$

C jelű csomópont, y irányú erők: $0.866\,T_3 + 0.866\,T_5 = f_2$

D jelű csomópont, x irányú erők: $-T_4 - 0.5\,T_5 + 0.5\,T_7 = 0$

A fenti egyenletrendszer kellő idővel és türelemmel megoldható számítógép nélkül is, sorban eliminálva az egyes változókat, majd visszahelyettesítve őket. Nyilvánvaló azonban, hogy számítógéppel megoldva lényegesen egyszerűbb a feladat. A most következőkben megnézzük, hogy hogyan oldjunk meg számítógéppel lineáris egyenletrendszereket. A megoldáshoz a legfontosabb, hogy realizáljuk, hogy a lineáris

egyenletek a mátrixokkal egyenértékűek, amelyek számokat és nem változókat tartalmaznak. A fenti lineáris egyenletrendszer mátrixos alakban $A\,x = b$ felírva a következő lesz:

$$A = \begin{pmatrix} 0.5 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0.866 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.5 & 0 & 0.5 & 1 & 0 & 0 & 0 \\ 0.866 & 0 & 0.866 & 0 & 0 & 0 & 0 \\ 0 & -1 & -0.5 & 0 & 0.5 & 1 & 0 \\ 0 & 0 & 0.866 & 0 & 0.866 & 0 & 0 \\ 0 & 0 & 0 & -1 & -0.5 & 0 & 0.5 \end{pmatrix}, b = \begin{pmatrix} f_1 \\ 0.433\,f_1 - 0.5\,f_2 \\ -f_1 \\ 0 \\ 0 \\ f_2 \\ 0 \end{pmatrix}$$

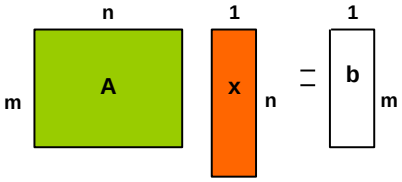
Azt se felejtsük el, hogy a gyakorlatban a mérnököknek igen nagyméretű mátrixokkal is dolgozni kell. Sokszor egy algoritmus, ami egy 2x2 vagy 3x3-as mátrixhoz megfelelő, alkalmatlan lehet pl. egy 2000x2000-es mátrixhoz, ezért az algoritmusok hatékonyságára is oda kell figyelnünk!

MEGOLDÁSOK LÉTEZÉSE ÉS EGYÉRTELMŰSÉGE

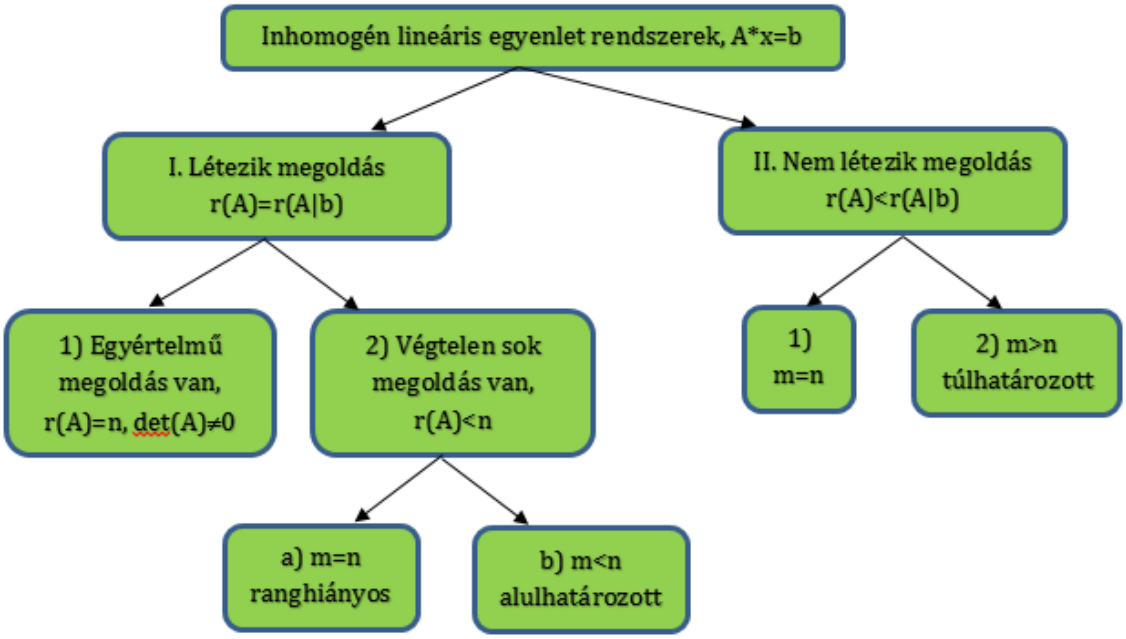
Nézzük meg először a lineáris egyenletrendszerek típusait megoldhatóság szerint! Az egyenletrendszer általános alakja mátrix formában felírva:

$$A \cdot x = b$$

ahol *A m*n*-es alakmátrix, *x* a keresett változók *nx1*-es oszlopvektora és *b* egy *mx1*-es oszlopvektor.



Beszélhetünk inhomogén (*b≠0*) és homogén (*b=0)* egyenletrendszerekről. A homogén esetben akkor van a triviális *x=0*-tól eltérő megoldás, ha az A mátrix determinánsa egyenlő nullával: **det**(A)=0. Erre láttunk példát az előző gyakorlaton a sajátérték problémánál. Inhomogén esetben a megoldások száma szerint a következő módon csoportosíthatjuk a megoldásokat:



MEGOLDÁS LÉTEZÉSE ÉS EGYÉRTELMŰSÉGE

LÉTEZIK ÉS EGYÉRTELMŰ A MEGOLDÁS

Létezik a megoldás, ha az A mátrix rangja (lineárisan független oszlopvektorainak száma, *r(A)*) megegyezik az A mátrixnak a b oszlopvektorral kibővített mátrix rangjával (*r(A|b)*), azaz r(A)=r(A|b). Matlab-ban a mátrix rangja a **rank** paranccsal számítható:

```
> rank(A), rank([A b])
```

Egyértelmű megoldás van, ha r(A)=r(A|b)=n illetve det(A)≠0, azaz az *A* mátrix rangja megegyezik a kibővített mátrix rangjával és ez a rang teljes (megegyezik az oszlopok számával is). A mátrix oszlopvektorai függetlenek. Ha a rang kisebb lenne, mint az oszlopok száma, akkor végtelen sok megoldás lenne. A megoldás:

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

Vizsgáljuk meg, Matlabot használva, hogy a fent megadott rácsos tartónál van-e megoldása a feladatnak és egyértelmű-e? Az A és a b mátrix kézzel is bevihető lenne az alábbi módon:

```
> A = [0.5 1 0 0 0 0 0; 0.866 0 0 0 0 0 0; -0.5 0 0.5 1 0 0 0;
>    0.866 0 0.866 0 0 0 0; 0 -1 -0.5 0 0.5 1 0; 0 0 0.866 0 0.866 0 0;
>    0 0 0 -1 -0.5 0 0.5]
>  f1 = 1000; f2 = 5000;
>  b = [f1; 0.433*f1-0.5*f2; -f1; 0; 0; f2; 0]
```

Azonban az A és b értékei el is vannak mentve a **racsos.txt** fájlban. Az elírások elkerülése végett most töltsük be ezt a fájlt, majd válasszuk szét az A mátrixot és a b vektort, ami az utolsó oszlopban van!

```
> Ab = load('racsos.txt')
> A = Ab(:,1:end-1)
> b = Ab(:,end)
```

Nézzük meg, hogy van-e egyértelmű megoldása a feladatnak!

```
> size(A,2) % n=7 oszlop van
> rank(A), rank([A,b]) % 7=7,  van megoldás, egyértelmű: n=7
```

Mivel az *A* mátrix rangja és a *b* vektorral kibővített mátrix rangja is 7, így van megoldása a feladatnak. A megoldás egyértelmű, mivel ez egy 7x7-es négyzetes mátrix és a rang megegyezik az oszlopok számával is. Ellenőrizzük, hogy a determináns nem nulla-e?

```
> det(A) % =-0.3247 – egyértelmű megoldás van
```

Írjunk egy programot, ami eldönti, hogy van-e egyértelmű megoldása a feladatnak, vagy nincs, és kiírja a választ a parancssorba!

```
> if rank(A)==rank([A,b]) && det(A)~=0
>     disp('Egyertelmű megoldás van')
> else disp('Nincs egyértelmű megoldás')
> end
```

GAUSS-ELIMINÁCIÓ, HÁROMSZÖG MÁTRIXOK

Azt már tudjuk, hogy a feladatnak van egyértelmű megoldása, már csak az a kérdés, ezt hogyan kapjuk meg? Mielőtt nekiállnánk a rácsos tartó rúderőinek számításához, nézzünk először egy egyszerűbb példát!

$$x_1 - 2x_2 + 3x_3 = 4$$
$$2x_1 - 5x_2 + 12x_3 = 15$$
$$2x_2 - 10x_3 = -10$$

Ezt felírhatjuk mátrixos alakban, vagy még egyszerűbben a kibővített mátrix alakjában:

$\begin{pmatrix} 1 & -2 & 3 \\ 2 & -5 & 12 \\ 0 & 2 & -10 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ 15 \\ -10 \end{pmatrix}$, kibővített mátrixként: $\left(\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 2 & -5 & 12 & 15 \\ 0 & 2 & -10 & -10 \end{array}\right)$

A kibővített mátrixos alak előnye, hogy csak számokat tartalmaz, változókat nem, ezekkel könnyebben boldogulnak a számítógépek. Ismételjük át a matematikában korábban már bizonyára megismert Gauss-eliminációt!

A Gauss-elimináció célja, hogy felső háromszög mátrixba alakítsuk az alakmátrixot, ami után már egyszerű visszahelyettesítéssel megoldható az egyenletrendszer. Nézzük a fenti példát! Ahhoz, hogy felső háromszög mátrix legyen, a főátló alatt csupa nulla elem lehet csak. Nullázzuk ki a második sor első elemét (2,1), úgy, hogy kivonjuk az első sor kétszeresét a második sorból:

$$\left(\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 2 & -5 & 12 & 15 \\ 0 & 2 & -10 & -10 \end{array}\right) \rightarrow \left(\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 0 & -1 & 6 & 7 \\ 0 & 2 & -10 & -10 \end{array}\right)$$

A harmadik sor első eleme (3,1) már nulla, itt nem kell semmit eliminálni, a 3. sor 2. eleme (3,2) azonban nem nulla. Ezt úgy tűntethetjük el, ha a harmadik sorból kivonjuk a második sor -2 szeresét.

$$\rightarrow \left(\begin{array}{ccc|c} 1 & -2 & 3 & 4 \\ 0 & -1 & 6 & 7 \\ 0 & 0 & 2 & 4 \end{array}\right) \rightarrow \begin{array}{rl} x_1 - 10 + 6 = 4 & \rightarrow x_1 = 8 \\ -x_2 + 12 = 7 & \rightarrow x_2 = 5 \\ 2x_3 = 4 & \rightarrow x_3 = 2 \end{array} \uparrow$$

A fenti mátrix már egy felső háromszögmátrix, ami könnyen megoldható. Ne felejtsük el, hogy a mátrix minden sora megfelel egy-egy egyenletnek. Az utolsó sor: $2x_3 = 4$, aminek nagyon egyszerű a megoldása: $x_3 = 2$. A második sor megfelel a $-x_2 + 6x_3 = 7$ egyenletnek, visszahelyettesítve x_3 már ismert értékét a $-x_2 + 12 = 7$ egyenletet kell megoldani, amiből $x_2 = 5$ adódik. Miután ismerjük x_2 és x_3 értékét, az első sor $x_1 - 10 + 6 = 4$ egyenletre egyszerűsödik, amiből $x_1 = 8$ adódik. Az tette lehetővé, hogy ilyen egyszerűen visszahelyettesítéssel megoldjuk az egyenletrendszert, hogy sikerült felső háromszög mátrixszá alakítani az alakmátrixot. Ellenőrizzük Matlab-ban:

```
> A = [1 -2 3; 2 -5 12; 0 2 -10], b = [4; 15; -10]
> x = inv(A)*b % x = [8; 5; 2]
```

A Gauss-elimináció hatékonyságának növeléséhez szükségünk lehet a sorok felcserélésére, permutációjára (pivoting), hogy csökkentsük a numerikus (kerekítési) hibák hatását. Ilyenkor úgy cseréljük fel a sorokat, hogy az adott oszlopban a legnagyobb abszolút értékű elemet használhassuk a többi elem eliminálására. Egy

nagyobb mátrixban minden oszlop elkészülte után célszerű megnézni, hogy a következő oszlopban melyik a legnagyobb abszolút értékű elem, és a szerint felcserélni a sorokat. Az előző példában, a numerikus pontosság növelése érdekében célszerű lett volna felcserélni először az első és a második sort, hiszen $|2| > |1|$, és a (2,1) elem kinullázása után a 2. és 3. sort is célszerű felcserélni.

LU FELBONTÁS

Sok esetben ugyanazt az **Ax = b** lineáris egyenletrendszert több **b** vektorra is meg kell oldanunk. Gondoljunk itt például egy híd próbaterhelésére, amit több különböző terheléssel is ellátnak, nem csak eggyel. Itt a különböző terhek jelentik a különböző **b** vektorokat. A fenti példából látható, hogy a munka nagyobb részét az **A** mátrixon végeztük. Ha több **b** vektorra is meg kell oldanunk ugyanazt az egyenletrendszert és **A** mátrix nagy, akkor szeretnénk elkerülni, hogy meg kelljen ismételni a Gauss-eliminációt minden lépését az **A** mátrixra minden **b** esetében. Ezt megtehetjük az LU felbontást használva, ami tulajdonképpen eltárolja a Gauss-eliminációt műveleteit is.

A fenti példát használva próbáljuk meg eltárolni, hogy az egyes lépésekben az adott sor hányszorosát kellett kivonni az aktuális sorból, hogy eliminálják a megfelelő elemet. Tegyük ezeket zárójelbe. Először a második sor első elemét (2,1) eliminálták, úgy, hogy kivontuk az első sor kétszeresét a második sorból (most csak az **A** mátrixot nézzük **b** nélkül):

$$\begin{pmatrix} 1 & -2 & 3 \\ \mathbf{2} & -5 & 12 \\ 0 & 2 & -10 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & -2 & 3 \\ \mathbf{0} & -1 & 6 \\ 0 & 2 & -10 \end{pmatrix}$$

(Előző sor hányszorosát vontuk ki, hogy kinullázzuk?)

(2)		

A harmadik sor első eleme (3,1) már eleve nulla volt, itt nem kellett semmit eliminálni, írjunk ide (0)-t, a 3. sor 2. elemét (3,2) úgy tűntettük el, hogy a harmadik sorból kivontuk a második sor -2 szeresét:

$$\begin{pmatrix} 1 & -2 & 3 \\ 0 & -1 & 6 \\ \mathbf{0} & \mathbf{2} & -10 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & -2 & 3 \\ 0 & -1 & 6 \\ \mathbf{0} & \mathbf{0} & 2 \end{pmatrix}$$

(Előző sor hányszorosát vontuk ki, hogy kinullázzuk?)

(2)		
(0)	**(-2)**	

Legyen U az előbb is megkapott felső háromszög mátrix (**U**pper trianglar matrix) és L egy alsó háromszög mátrix (**L**ower trianglar matrix), aminek az elemei legyenek az eltárolt műveletek és a főátlójában 1-esek:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix} \text{ és } U = \begin{pmatrix} 1 & -2 & 3 \\ 0 & -1 & 6 \\ 0 & 0 & 2 \end{pmatrix}$$

Ez az úgynevezett LU felbontás. Ennek a felbontásnak megvan az a tulajdonsága, hogy $L \cdot U = A$:

$$L \cdot U = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & -2 & 3 \\ 0 & -1 & 6 \\ 0 & 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & -2 & 3 \\ 2 & -5 & 12 \\ 0 & 2 & -10 \end{pmatrix} = A$$

Ellenőrizzük le Matlab-ban:

```
> L = [1 0 0; 2 1 0; 0 -2 1]
```

```
> U = [1 -2 3; 0 -1 6; 0 0 2]
> L*U-A
> norm(L*U-A) % 0 (ellenőrzés)
```

Láthatjuk, hogy az **A** mátrix előáll az **L** és az **U** mátrixok szorzataként. Itt **L** egy alsó, **U** egy felső háromszög mátrix. Amikor egy mátrix előállítható egyszerűbb mátrixok szorzataként, azt mátrix felbontásnak (decomposition) nevezzük. Ez éppen az LU felbontás, ahol U mátrixban a Gauss-elimináció eredménye van eltárolva, L mátrixban pedig a lépései. Ellenőrzésként általában az eltérésvektor/mátrix normáját ('hosszát') szokás megadni, ezt a **norm** paranccsal tehetjük.

MEGOLDÁS LU FELBONTÁSSAL

Ha a sorok felcserélését, permutációját is belevesszük, akkor **A** mátrix LU felbontása 3 mátrixból áll (**L**, **U**, **P** – permutáció mátrix):

$$P \cdot A = L \cdot U$$

A **P** permutáció mátrix egy egységmátrix felcserélt sorokkal, amelyekben ugyanazok a sorok vannak felcserélve, mint az **A** mátrixban. Pl. a következő **P** permutációs mátrix a második és a harmadik sor felcserélését jelenti: $P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$.

Matlab-ban LU felbontást permutációval együtt a következő paranccsal állíthatunk elő:

```
> [L U P] = lu(A)
```

Ha ezt szeretnénk az $A \cdot x = b$ lineáris egyenletrendszer megoldására használni, akkor először szorozzuk meg mind a két oldalt a permutációs mátrixszal:

$$P \cdot A \cdot x = P \cdot b \equiv d$$

Majd helyettesítsük be $P \cdot A$ helyére $L \cdot U$-t:

$$L \cdot U \cdot x = d$$

Legyen $y = U \cdot x$:

$$L \cdot y = d$$

Ezek után csupán két egyszerű visszahelyettesítéses feladatot kell megoldanunk, két háromszög mátrixszal:

- $L \cdot y = d$ (ahol L alsó háromszög mátrix és $d = P \cdot b$)
- $U \cdot x = y$ (ahol U felső háromszög mátrix)

PÉLDA LU FELBONTÁSRA

Oldjuk meg az előbbi rácsos tartó egyenletrendszerét is LU felbontással! A megoldás során használjunk olyan algoritmust, ahol figyelembe tudjuk venni az L és U mátrixok speciális voltát (alsó ill. felső háromszög mátrix)!

Először töltsük be újra a mátrixokat, majd állítsuk elő az LU felbontást!

```
> Ab = load('racsos.txt'); A = Ab(:,1:end-1); b = Ab(:,end);
> [L U P] = lu(A)
```

Most jön az $L \cdot y = d$ és az $U \cdot x = y$ megoldása. Ez tulajdonképpen két egyszerű visszahelyettesítést jelent, mivel tudjuk, hogy ezek a mátrixok alsó/felső háromszögmátrixok. Ehhez használhatjuk a lineáris egyenletrendszerek megoldásához használható **linsolve** parancsot. A **linsolve** parancsot használva az opcióknál megadható az *A* mátrix néhány tulajdonsága, ha előzetesen ismert (pl. alsó/felső háromszögmátrix, szimmetrikus, pozitív definit), ez lényegesen gyorsíthatja a megoldást. Itt az **LT** a lower triangle – alsó háromszögmátrix, az **UT** az upper triangle – felső háromszögmátrix rövidítése. Ezeket a tulajdonságokat egy struktúra típusban adhatjuk meg, ahol a változónév után ponttal hivatkozhatunk az adott tulajdonságra, hogy igaz-e vagy hamis.

```
> d = P*b;
> opt1.LT=true
> y = linsolve(L,d,opt1);
> opt2.UT=true
> x = linsolve(U,y,opt2)
> elteres = norm(A*x - b) % ellenőrzés
```

Az LU felbontás egy példa a mátrix felbontásokra, ahol az **A** mátrixot két egyszerűbb mátrixra bontottuk, egy alsó és egy felső háromszög mátrix szorzatára Gauss-eliminációval. Sok másféle mátrix felbontással is találkozhatunk, amelyek különböző esetekben lehetnek hasznosak. A legismertebbek az LU felbontáson kívül a Cholesky felbontás, QR felbontás és az SVD felbontás.

CHOLESKY FELBONTÁS

A Cholesky felbontás és használata lineáris egyenletrendszer megoldására nagyon hasonló az LU felbontásra. A különbség annyi, hogy itt nem egy alsó (**L**) és egy felső (**U**) háromszög mátrixra bontjuk **A** mátrixot, hanem egy darab **L** felső(!) háromszög mátrixunk lesz, és az **A** mátrix ennek és a transzponáltjának (ami már alsó háromszög mátrix) lesz a szorzata:

$$A = L^T \cdot L$$

A megoldás menete $A \cdot x = L^T \cdot L \cdot x = L^T \cdot y = b$ alapján:

- $L^T \cdot y = b$ (ahol L^T alsó háromszög mátrix)
- $L \cdot x = y$ (ahol L felső háromszög mátrix)

Itt csak egy háromszög mátrixunk van, ami lényegesen egyszerűbb, mint a két háromszög mátrix az LU felbontás esetében. Viszont ezt a felbontást nem tudjuk mindig elvégezni, csak, ha néhány fontos feltétel teljesül:

- Az **A** mátrix szimmetrikus, azaz $\mathbf{A^T=A}$ és
- pozitív definit, azaz minden sajátértéke pozitív: $\lambda_i > 0, i = 1, \cdots, n$

Matlab-ban a sajátértékeket az **eig** paranccsal állíthatjuk elő, amit kétféleképp hívhatunk:

```
> E = eig(A) % négyzetes X mátrix sajátértékei
> [V,D] = eig(A) % sajátvektorok V mátrixban (oszlopok), sajátértékek a
  D diagonálmátrixban
```

Vizsgáljuk meg, hogy a rácsos tartó A mátrixa szimmetrikus és pozitív definit-e?

```
> norm(A-A') %  1.5946
```

```
> eig(A) % [0.5000; -0.7136; -0.7136; -0.7136; 1.2136; 1.2136; 1.2136]
```

A fenti mátrix se nem szimmetrikus norm(A'-A)≠0, se nem pozitív definit (a sajátértékek nem mind pozitívak), tehát Cholesky felbontással nem oldható meg a feladat (**chol**(A) parancs hibaüzenetet adna). Nézzünk példát egy szimmetrikus, pozitív definit mátrixra! A **pascal** paranccsal előállíthatjuk a binomiális együtthatókat tartalmazó szimmetrikus Pascal mátrixot, a **diag** paranccsal pedig kivehetjük egy mátrix főátlójából az elemeket (vagy egy vektorból csinálhatunk diagonális mátrixot), a **min** paranccsal pedig lekérdezhetjük egy vektor legkisebb elemét (**max** paranccsal pedig a legnagyobbat).

```
> A = pascal(4) % a binomiális együtthatókat tartalmazó Pascal mátrix
> norm(A-A') % szimmetrikus, mivel A = A'
> [V D] = eig(A) % sajátvektorok és sajátértékek előállítása
> min(diag(D)) % pozitív definit, mivel a legkisebb sajátérték is>0
> L = chol(A) % elvégezhető a Cholesky felbontás
> norm(A-L'*L) % ellenőrzés
> b = rand(4,1) % tetszőleges b vektor
```

Az egyenletrendszer megoldása hasonló az LU felbontáshoz (különbség, hogy itt nincs permutáció, tehát d helyett b vektor lesz, L helyére L' és U helyére L kerül):

```
> opt1.LT=true
> y = linsolve(L',b,opt1);
> opt2.UT=true
> x = linsolve(L,y,opt2)
> elteres = norm(A*x - b) % ellenőrzés
```

MATLAB BEÉPÍTETT FÜGGVÉNYEK (INV, LINSOLVE, \ - MLDIVIDE)

Természetesen a Matlab-nak vannak beépített parancsai is az $A \cdot x = b$ egyenletrendszer közvetlen megoldására, anélkül, hogy nekünk kellene lépésenként elvégezni az LU vagy éppen a Cholesky felbontást. Egyértelmű megoldás esetén a beépített parancsok többsége ezeket a felbontásokat használja. Nézzünk rá példákat!

Háromféle lehetőséget fogunk megnézni:

- `x = inv(A)*b`
- `x = linsolve(A,b)`
- `x = A\b % vagy x=mldivide(A,b)`

Nézzük meg mi a különbség az egyes megoldások között a Matlab-ban! Határozzuk meg a beépített parancsokkal is a rácsos tartó rúderőit! Minden megoldásnál nézzük meg a megoldás időigényét is. Tegyük ezt úgy, hogy 10000-szer elvégezzük mindegyik számítást, hogy reálisan összevethetőek legyenek az idők. A Matlab-ban időmérést a **tic**-**toc** parancsokkal tudunk végrehajtani.

Nézzük először a matematikában hagyományosan felírható megoldást, az inverz számítás segítségével: $x = A^{-1} \cdot b$! Matlabban az **inv** parancs használható inverz számításra.

```
> %% Beépített függvények
> Ab = load('racsos.txt'); A = Ab(:,1:end-1); b = Ab(:,end);
> % Megoldás: x=inv(A)*b
> x1 = inv(A)*b % hagyományos inverz számítás
> tic
> for i=1:10000
>     x1 = inv(A)*b;
```

```
> end
> toc % Elapsed time is 0.118122 seconds.
> norm(A*x1-b) % 1.0904e-12
```

Természetesen a futás idő gépenként (sőt futtatásonként) is különbözni fog, de a nagyságrendjük összehasonlítható. A tesztgépen a futásidő 0.118 másodperc volt az inverz számításnál.

Használtuk korábban a **linsolve** parancsot alsó és felső háromszögmátrixokkal megadott egyenletrendszerek megoldására. Ez a parancs nem csak ezekben a speciális esetekben használható, hanem általános esetben is, ilyenkor nem adunk meg opciókat. Nézzük meg a megoldást ebben az esetben!

```
> % Megoldás: x = linsolve(A,b)
> x2 = linsolve(A,b) % nxn: Cholesky vagy LU felbontás
> tic
> for i=1:10000
>     x2 = linsolve(A,b);
> end
> toc % Elapsed time is 0.082469 seconds.
> norm(A*x2-b) % 6.0157e-13
```

A **linsolve** parancs LU felbontást használ a megoldáshoz, ha az A mátrix négyzetes. Itt a futásidő 0.082 másodperc lett, jóval gyorsabb, mint az előző esetben.

Nézzük meg a harmadik parancsot is, ez az x = A\b megoldás. A **** parancs megegyezik az **mldivide** paranccsal (x=mldivide(A,b)). Ez a függvény több megoldási módszer közül választja ki a legmegfelelőbbet a mátrix vizsgálata után (lásd az mldivide folyamatábráját a fejezet végén).

Négyzetes mátrix (nxn) esetében használhatja pl. a Cholesky vagy LU felbontást.

```
> % Megoldás: x = A\b
> x3 = A\b % nxn: pl. Cholesky vagy LU felbontás
> x3 = mldivide(A,b) % ua., mint az előző
> tic
> for i=1:10000
>     x3 = A\b;
> end
> toc % Elapsed time is 0.024091 seconds.
> norm(A*x3-b) % 5.5695e-13
```

A futásidő ebben az esetben lett a legjobb, 0.024 másodperc, ami egy nagyságrenddel jobb, mint a hagyományos inverz számítás volt.

Nézzük meg az eredményeket is:

```
1.0e+03 *

   -2.3868   -2.3868   -2.3868
    2.1934    2.1934    2.1934
    2.3868    2.3868    2.3868
   -3.3868   -3.3868   -3.3868
    3.3868    3.3868    3.3868
    1.6934    1.6934    1.6934
   -3.3868   -3.3868   -3.3868
```

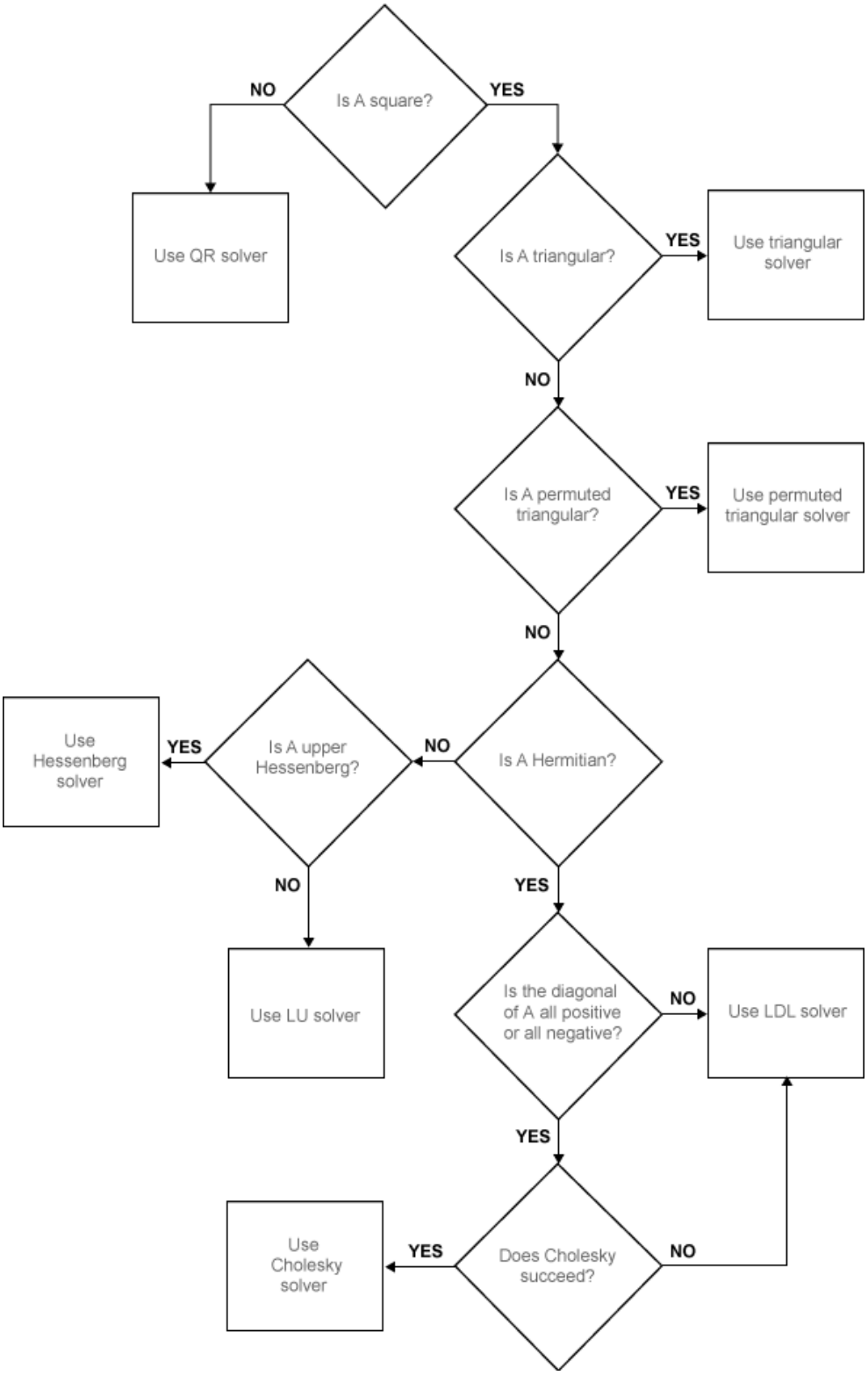
Lényegében mindenhol ugyanazt az eredményt kaptuk x-re, kis eltérések vannak, ha megnézzük az eltérésvektor normáját, a legpontosabb eredményt (legközelebb nullához) itt is az A\b parancs adta:

x=inv(A)*b esetén: $\|A \cdot x - b\| = 1.0904\text{e} - 12$

x=linsolve(A,b) esetén: $\|A \cdot x - b\| = 16.0157\text{e} - 13$

x=A\b esetén: $\|A \cdot x - b\| = 15.5695\text{e} - 13$

A fentiek alapján általános esetben, négyzetes mátrix és egyértelmű megoldás mellett célszerű az **x=A\b** parancsot használni. Amennyiben előre tudjuk, hogy a mátrix alsó/felső háromszögmátrix, szimmetrikus vagy pozitív definit mátrix, akkor célszerű az **x=linsolve(A,b)** parancsot alkalmazni és megadni az opcióknál a mátrix típusát.



MLDIVIDE (\) PARANCS ÁLTAL HASZNÁLT ALGORITMUSOK A MATLABBAN