Debreceni Egyetem

Informatikai Kar

BÁZIS MÁTRIX FRISSÍTÉSE LINEÁRIS ÉS HIPERBOLIKUS PROGRAMOZÁSI FELADATOKBAN

Konzulens:
Dr. Bajalinov Erik
Tudományos főmunkatárs

Készítette:
Fogarasi Edina
Programtervező matematikus

Debrecen
2007

Tartalomjegyzék

Bevezetés 3.

1 Lineáris és hiperbolikus programozási feladat 4.

2 Számítási problémák 5.

2.1 Hiperbolikus probléma skálázása 6.

2.1.1 Az RHS vektor skálázása: $b \rightarrow \rho b$ 8.

2.1.2 Oszlop helyettesítés: $A_j \rightarrow \rho A_j$ 10.

2.1.3 Sor helyettesítés: $a_i \rightarrow \rho a_i$ 13.

2.1.4 A számláló vektor skálázása: $p \rightarrow \rho p$ 14.

2.1.5 A nevező vektor skálázása: $d \rightarrow \rho d$ 15.

2.1.6 Skálázó faktorok 16.

2.1.6.1 Hall szabály 17.

2.1.6.2 Gondzio szabály 17.

2.1.6.3 Implementáció 18.

2.2 Bázis mátrix faktorizációja 19.

2.2.1 LU-faktorizáció 20.

2.2.2 LU-faktorizáció és Gauss-elimináció 24.

2.2.3 LU-faktorizáció frissítése 28.

2.2.3.1 Alapok 29.

2.2.3.2 Bartels-Golub frissítés 30.

2.2.3.3 Forest-Tomlin frissítés 36.

2.2.4 Egyéb faktorizáció típusok 38.

2.2.4.1 Cholesky felbontás 38.

2.2.4.2 QR-felbontás 39.

3 Bartels-Golub frissítés megvalósítása 43.

3.1 A DLL 43.

3.2 Az LU felbontás frissítésének megvalósítása 44.

3.2.1 Az EnDll állomány ismertetése 45.
Az eljárások 46.
Összefoglalás 53.
Irodalomjegyzék 54.

Bevezetés

Az operációkutatás elég új tudományág, mely szorosan összefügg a számítógépek kialakulásával, fejlődésével. Az operációkutatás feladata a gyakorlati élet különböző problémacsoportjaihoz az illető problémacsoportokat leíró optimumszámítási modellek konstruálása, továbbá a meglévő modellekhez az optimális megoldást meghatározó eljárások kidolgozása.

Az operációkutatásnak csak kis szeletével, a lineáris programozással foglalkozom a dolgozatomban. Általános lineáris programozási feladatok megoldására alkalmazható a szimplex módszer. A szimplex módszer lényege egy lehetséges bázismegoldás megkeresése, mely igen sok számítást igényel, és néha a kerekítések miatt nem megfelelő eredményre jutnak.

Dolgozatom első fejezetében röviden ismertetem a lineáris és hiperbolikus programozási feladatokat, majd a második fejezetben írok a számítási problémákról, és ezek elkerülésének, kiküszöbölésének két módjáról. Részletesebben tárgyalom a bázis mátrix LU felbontásán, és ennek frissítésén alapuló módszert.

Végül ismertetem az LU felbontás Bartels-Golub módszer alapján történő frissítés Delphi nyelven történő megvalósítását, környezetét, tapasztalatokat.

1. Lineáris és hiperbolikus programozási feladat

Lineáris programozási feladat: Lineáris programozási feladaton olyan feltételes szélsőérték feladatot értünk, amelyben a feltételek lineáris egyenlőség és egyenlőtlenség formájában adottak, és ezen feltételek mellett keressük egy lineáris függvénynek, a célfüggvénynek a minimumát, vagy maximumát.

Célfüggvénye:

$$P(x) = \sum_{j=1}^{n} p_j x_j \rightarrow max/min \tag{1.1}$$

Hiperbolikus programozási feladat (Linear-fractional programming): Hiperbolikus programozási feladat alatt olyan optimumszámítási feladatot értünk, amelyben a feltételek lineáris egyenlőség és egyenlőtlenség formájában adottak, és ezen feltételek mellett keressük két lineáris függvény hányadosának a maximumát vagy minimumát.

Célfüggvénye:

$$Q(x) = \frac{P(x)}{D(x)} = \frac{\sum_{j=1}^{n} p_j x_j + p_0}{\sum_{j=1}^{n} d_j x_j + d_0} \rightarrow max/min \tag{1.2}$$

Az előbbi két célfüggvényre a következő megszorításoknak kell teljesülnie:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i=1,\ldots,m; \tag{1.3}$$

$$x_j \geqslant 0, \quad j=1,\ldots,n, \tag{1.4}$$

ahol $\forall\, x=(x_1,\ldots,x_n)^T \in S$, S a lehetséges megoldások halmaza az (1.3) és (1.4) feladatra vonatkozóan, és hiperbolikus esetben a $D(x)>0$ teljesül.

2. Számítási problémák

A lineáris és hiperbolikus programozási feladatok megoldása rengeteg lebegőpontos aritmetikai számítást igényelnek. A számítógéppel végzett számítások esetén – a módszer pontossága miatt - csekély számú számítási hiba történik. Ezeknek a hibáknak kommulatív hatása van, ami numerikus stabilitási problémához vezet és a kapott eredményben is jelentős hibákat okozhat.

Ahhoz, hogy ezeket a hibákat elkerüljék, minden gyári LP megoldó rendelkezik egy speciális és kifinomult technikával, amely nagy mértékben csökkenti a kerekítés összegződő hatásait, és gyakran a megoldás nagymértékű javulásához vezet.

E technikák egyik legegyszerűbb, viszonylag hatékony és széles körben elterjedt típusa a skálázás[1]. Ez a technika azt jelenti, hogy az eredeti optimalizálási feladat $A=\|a_{ij}\|_{m\times n}$ mátrixának azon sorait és/vagy oszlopait, melyek gyengén (vagy rosszul) skálázottak, egyenként el kell osztani (vagy meg kell szorozni) a saját skálázási fokszámukkal: $\rho_i^r, i=1,\ldots,m$ és/vagy $\rho_j^c, j=1,\ldots,n$. A legtöbb valós LP és LFP alkalmazás esetén a modell eredetileg gyengén skálázott. Épp ezért mielőtt a szimplex módszert vagy más módszert alkalmaznánk, a program újraskálázza a sorokat, oszlopokat, vagy az RHS, azaz a $b=(b_1,\ldots,b_m)^T$ vektort[2].

Az ilyen skálázási algoritmusok a célfüggvény együtthatóit vagy tartalmazzák, vagy nem. Az LP probléma esetén az *A* skálázó mátrix, a jobb oldali *b* vektor és a *P(x)* célfüggvény nem okoz gondot a feltételek és a célfüggvény linearitása miatt. A legtöbb esetben a skálázás javítja a megoldott feladat eredményét, tehát célszerű alkalmazni. Sőt, néha nagymértékben csökkenti a szimplex módszer iterációinak a számát is.

Általában a következő skálázások közül lehet választani:

- Nincs skálázás
- Csak sorokat skáláz

1 A probléma adatainek megoldás előtti transzformálása, mely segítségével megpróbáljuk az adatok terjedelmét olyan szűkre szabni, amennyire csak lehet.

2 Gyakori skálázó algoritmus, hogy osztunk minden sort a legnagyobb elemével, majd minden eredményül kapott oszlopot a legnagyobb elemével. Ezzel elérjük, hogy a legnagyobb elem a mátrixban 1.0 legyen, és minden sorban és oszlopban legalább 1 elem egyenlő legyen 1.0-val.

- Csak oszlopokat skáláz
- Sorokat és oszlopokat is skáláz,

mindezt célfüggvénnyel vagy anélkül.

Az LFP probléma esetében nem szabad megfeledkeznünk az LFP és LP probléma fő különbségéről, hogy az LFP *Q(x)* célfüggvénye nem lineáris.

Egy másik széleskörben elterjedt módszer a kerekítés kommulatív hatásának csökkentésére, a bázismátrix refaktorizációja. Ez a technika azt jelenti, hogy lineáris algebrai módszereket használva újraszámoljuk a szimplex tábla x_{ij} együthatóit. A legtöbb program az LU-felbontást vagy más speciális metódusokat alkalmaz (pl. Cholesky-algoritmust, ami szimmetrikus mátrixot igényel) periódikusan frissítve a szimplex módszert a végrehajtás ideje alatt. Ez javítja a numerikus stabilitást, ugyanakkor a refaktorizáció költséges művelet.

2.1. Hiperbolikus probléma skálázása

Ezt a fejezet egy példával kezdem, mely megmutatja, hogy egy rosszul vagy egyáltalán nem skálázott egyenleten végzett véges pontosságú számítások, és kerekítések milyen komoly problémákat okozhatnak.

A következő lineáris egyenletünk van:

$$\begin{pmatrix} 0.003 & 59.140 \\ 5.291 & -6.130 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 59.17 \\ 46.78 \end{pmatrix}$$

A pontos megoldás:

$$x_1 = 10.000; \quad x_2 = 1.000$$

Most oldjuk meg az egyenletet Gauss eliminációval és 4 tizedesjegy pontossággal. Válasszuk tengelynek az $a_{11} = 0.003$ -as elemet és számoljuk ki a következő szorzót:

$$\lambda = \frac{a_{21}}{a_{11}} = \frac{5.291}{0.003} = 1763.6667 \approx 1763.(6)$$

Ezután végezzük el a következő elemi sorátalakítást:

$$(2.sor) - \lambda(1.sor) \rightarrow (2.sor)$$

Ekkor a következőt kapjuk eredményül, ha a $\lambda = 1763.6667$:

$$\begin{pmatrix} 0.0030 & 59.1400 \\ 0.0000 & -104309.3786 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 59.1700 \\ -104309.3786 \end{pmatrix}$$

és ezt kapjuk, ha a kerekített λ-val számolunk:

$$\begin{pmatrix} 0.0030 & 59.1400 \\ 0.0000 & -104309.37(6) \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 59.1700 \\ -104309.37(6) \end{pmatrix}$$

Az így kapott helyettesítési érték: $x_2 = 1.00000000001$, ami elég közel van a helyes $x_2 = 1.0000$ eredményhez. Habár egy közel helyes x_2 értékkel számolunk, az x_1 értékéül mégis a következőt kapjuk:

$$x_1 \approx \frac{59.17 - (59.140)(1.00000000001)}{0.003} = -9.71(3)$$

A helyes eredmény viszont $x_1 = 10.0000$ lenne. Igaz, hogy x_2-nél a hiba csak 0.00000000001 volt, de ez szorzódott 59.121 / 0.003-mal.

Természetesen sok számítógép sokkal pontosabban számol, de ők is véges pontossággal dolgoznak. A legtöbb az IEEE szabványt használja, ami 16 számjegyű pontosságot jelent.

$$\varepsilon = 2.23 \times 10^{-16}.$$

Ez azt jelenti, hogy:

$$1 + \varepsilon = 1 + \varepsilon$$

$$1 + \frac{1}{2}\varepsilon = 1.$$

Egy lineáris egyenletrendszer megoldása közben felmerülő pontossági problémák elkerülésének egyszerű módja az, hogy egy teljesen skálázott feladattal dolgozunk. Ezt a megközelítést eredményesen lehet alkalmazni az LFP problémák esetén is. Az LFP probléma skálázása hatással lehet a kiszámított eredmény pontosságára, és ahhoz vezethet, hogy a szimplex módszer közben meg kell változtatni a kiválasztott tengelyt.

Az LFP feladat skálázásakor a következő eseteket különböztetjük meg:

1. Megszorítások, feltételek skálázása:
   - Jobb oldali vektor skálázása: $b=(b_1,\ldots,b_m)^T$;
   - Az A mátrix j. oszlopának skálázása: $A_j,\ j = 1,\ldots,n;$
   - Az A mátrix sorainak skálázása.

2. A célfüggvény skálázása:
   - Csak a számlálóban lévő $P(x)$ függvény vektora: $p = (p_0, \ldots, p_n)$
   - Csak a nevezőben lévő $D(x)$ függvény vektora: $d = (d_0, \ldots, d_n)$
   - A $Q(x)$ célfüggvényhez tartozó vektorok (p és d).

2.1.1. Az RHS vektor skálázása: $b \rightarrow \rho b$

Tegyük fel, hogy az x^* az optimális megoldása az LFP feladatnak, tehát

$$\sum_{j=1}^{n} A_j x_j^* = b \quad és \quad x^* \geqslant 0$$

és a bázismátrixa: $B = (A_{S_1},\ldots,A_{S_m})$

Helyettesítsük a jobb oldali b vektort a $b' = \rho b$-vel, ahol $\rho > 0$. Ezt figyelembe véve az új vektor: $x' = \rho x^*$. Az x' kielégíti a következő feltételeket:

$$\sum_{j=1}^{n} A_j(\rho x_j^*) = \rho b \quad és \quad x' = \rho x^* \geqslant 0,$$

Tehát az x' lehetséges megoldása az alábbi LFP feladatnak:

$$Q(x) = \frac{P(x)}{D(x)} = \frac{\sum_{j=1}^{n} p_j x_j + p_0'}{\sum_{j=1}^{n} d_j x_j + d_0'} \rightarrow max \tag{2.1}$$

melyre a következő feltételeknek kell teljesülnie:

$$\sum_{j=1}^{n} a_{ij} x_j = \rho b_i, \quad i = 1,\ldots,m; \tag{2.2}$$

$$x_j \geqslant 0, \quad j = 1,\ldots,n. \tag{2.3}$$

Ellenőrizzük le, hogy x' optimális megoldása-e a (2.1)-(2.3)-mal jelölt feladatnak.
Mivel az x^* optimális megoldása az eredeti LFP feladatnak, felírhatjuk a következőt:

$$\Delta_j(x^*) \geqslant 0, \quad j=1,\ldots,n, \tag{2.4}$$

ahol

$$\Delta_j(x^*) = D(x^*)\Delta_j' - P(x^*)\Delta_j'', \quad j=1,\ldots,n,$$

$$\Delta_j' = \sum_{i=1}^{m} p_{s_i} x_{ij} - p_j, \quad j=1,\ldots,n,$$

$$\Delta_j'' = \sum_{i=1}^{m} d_{s_i} x_{ij} - d_j, \quad j=1,\ldots,n.$$

Az x_{ij} együtthatót a következő rendszerből határozzuk meg:

$$\sum_{i=1}^{m} A_{s_i} x_{ij} = A_j, \quad j=1,\ldots,n \tag{2.5}$$

és A_j jelölje az $A = \|a_{ij}\|_{m\times n}$ mátrix j. oszlopvektorát:

$$A_j = (a_{1j},\ldots,a_{mj})^T, \quad j=1,\ldots,n$$

A skálázott Δ_j' és Δ_j'' költségek függetlenek a b RHS vektortól, tehát a $b \to \rho b$ helyettesítés nincs hatással a Δ_j' és Δ_j'' értékeire. Viszont a *P(x)* és *D(x)* függvények értékei függnek a b RHS vektortól, ezért figyeljük meg az új, csökkentett $\Delta_j(x')$ költségeket, ahol $x' = \rho x^*$.

$$\boldsymbol{\Delta_j(\rho x^*)} = D(\rho x^*)\Delta_j' - P(\rho x^*)\Delta_j'' =$$

$$= (\sum_{j=1}^{n} d_j(\rho x_j^*) + d_0')\Delta_j' - (\sum_{j=1}^{n} p_j(\rho x_j^*) + p_0')\Delta_j'' =$$

$$= (\sum_{j=1}^{n} d_j(\rho x_j^*) + d_0' + \rho d_0 - \rho d_0)\Delta_j' -$$

$$-(\sum_{j=1}^{n} p_j(\rho x_j^*) + p_0' + \rho p_0 - \rho p_0)\Delta_j'' =$$

$$= \rho D(x^*)\Delta_j' + (d_0' - \rho d_0)\Delta_j' -$$

$$-\rho P(x^*)\Delta_j'' - (p_0' - \rho p_0)\Delta_j'' =$$

$$= \rho \Delta_j(x^*) + (d_0' - \rho d_0)\Delta_j' - (p_0' - \rho p_0)\Delta_j'' =$$

$$= \boldsymbol{\rho \Delta_j(x^*) - G_j} \tag{2.6}$$

ahol:

$$G_j = \begin{vmatrix} p_0' - \rho\, p_0 & \Delta_j' \\ d_0' - \rho\, d_0 & \Delta_j'' \end{vmatrix}$$

A (2.6) azt jelenti, ha p_0' és d_0' olyanok, hogy

$$\rho\, \Delta_j(x^*) - G_j \geqslant 0, \quad j = 1, \ldots, n,$$

vagy, ha

$$p_0' = \rho\, p_0 \quad \acute{e}s \quad d_0' = \rho\, d_{0,}$$

akkor

$$\Delta_j(\rho\, x^*) \overset{(2.6)}{=} \rho\, \Delta_j(x^*) \overset{(2.4)}{\geqslant} 0, \quad \forall\, j = 1, \ldots, n,$$

ezért az x' vektor egy optimális megoldása az LFP feladatnak.

Tehát ha a b vektort helyettesítjük valamilyen másik $b' = \rho\, b, \rho > 0$ vektorral, akkor csak ki kell cserélni a p_0 és d_0 együtthatókat az eredeti $Q(x)$ célfüggvényben $p_0' = \rho\, p_0$ és $d_0' = \rho\, d_{0,}$-ra. Ez a két helyettesítés garantálja az egyenlőséget az eredeti feladat és az új, skálázott LFP feladat között. Nyilvánvaló, ha az x' vektor optimális megoldása az újra skálázott LFP feladatnak, akkor az $x^* = x'/\rho$ optimális megoldása az eredeti LFP feladatnak.

2.1.2. Oszlop helyettesítés: $A_j \rightarrow \rho\, A_j$

Most az A_j oszlopvektort skálázzuk, ami az $A = \|a_{ij}\|_{m \times n}$ mátrix egyik oszlopa *(j=1,...,n)*. tegyük fel, hogy az x^* vektor optimális megoldás az eredeti LFP feladatban. Tehát:

$$\sum_{j=1}^{n} A_j x_j^* = b \quad \acute{e}s \quad x_j^* \geqslant 0, \quad j = 1, \ldots, n$$

és a $B = (A_{S_1}, \ldots, A_{S_m})$ a bázis mátrix. Cseréljünk ki néhány A_r vektort *(r ∈ J = {1,...,n})* néhány más $A_r' = \rho\, A_r$ vektorra, ahol $\rho > 0$.

Nyilvánvaló, hogy az új vektor : $x' = \left(x_1^*,\ x_2^*,\ \ldots,\ x_{r-1}^*,\ \frac{x_r^*}{\rho},\ x_{r+1}^*,\ \ldots,\ x_n^*\right)$

Ez ki fogja elégíteni a feltételeket:

$$\sum_{\substack{j=1\\ j\neq r}}^{n} A_j x_j^* + \rho A_r \frac{x_r^*}{\rho} = b,$$

$$x_j' \geqslant 0, \quad j=1,\ldots,n,$$

ezért az x' lehetséges megoldása az új skálázott LFP feladatnak:

$$Q(x) = \frac{P'(x)}{D'(x)} = \frac{\sum_{\substack{j=1\\ j\neq r}}^{n} p_j x_j + p_r' x_r + p_0}{\sum_{\substack{j=1\\ j\neq r}}^{n} d_j x_j + d_r' x_r + d_0} \rightarrow max \tag{2.7}$$

a következő feltételek mellett:

$$\sum_{\substack{j=1\\ j\neq r}}^{n} A_j x_j + A_r' x_r = b, \tag{2.8}$$

$$x_j \geqslant 0, \quad j=1,\ldots,n. \tag{2.9}$$

A célunk az, hogy megvizsgáljuk, hogy az x' vektor optimális megoldása-e a skálázott LFP feladatnak.

Ha x^* optimális megoldás az eredeti feladatban, akkor

$$\Delta_j(x^*) = D(x^*)\Delta_j' - P(x^*)\Delta_j'' \geqslant 0, \quad j=1,\ldots,n \tag{2.10}$$

Tegyük fel, hogy A_r egy bázisvektor, $r \in J_B = \{s_1,\ldots,s_m\}$. Ez esetben az új skálázott feladatnál:

$$\Delta_j(x') = D'(x')\Delta_j' - P'(x')\Delta_j'' =$$

$$= \Big(\sum_{\substack{j=1\\ j\neq r}}^{n} d_j x_j^* + d_r' \frac{x_r^*}{\rho} + d_0\Big)\Big(\sum_{\substack{i=1\\ s_i\neq r}}^{m} p_{s_i} x_{ij} + p_r' \frac{x_{rj}}{\rho} - p_j\Big) -$$

$$- \Big(\sum_{\substack{j=1\\ j\neq r}}^{n} p_j x_j^* + p_r' \frac{x_r^*}{\rho} + p_0\Big)\Big(\sum_{\substack{i=1\\ s_i\neq r}}^{m} d_{s_i} x_{ij} + d_r' \frac{x_{rj}}{\rho} - d_j\Big) =$$

$$= \Big(\sum_{\substack{j=1\\ j\neq r}}^{n} d_j x_j^* + d_r' \frac{x_r^*}{\rho} + d_0 + d_r x_r^* - d_r x_r^*\Big) \times$$

$$\times \left(\sum_{\substack{i=1 \\ s_i \neq r}}^{m} p_{s_i} x_{ij} + p_r' \frac{x_{rj}}{\rho} - p_j + p_r x_{rj} - p_r x_{rj}\right) -$$

$$-\left(\sum_{\substack{j=1 \\ j \neq r}}^{n} p_j x_j^* + p_r' \frac{x_r^*}{\rho} + p_0 + p_r x_r^* - p_r x_r^*\right) \times$$

$$\times \left(\sum_{\substack{i=1 \\ s_i \neq r}}^{m} d_{s_i} x_{ij} + d_r' \frac{x_{rj}}{\rho} - d_j + d_r x_{rj} - d_r x_{rj}\right) =$$

$$= \left(D(x^*) - d_r x_r^* + d_r' \frac{x_r^*}{\rho}\right)\left(\Delta_j' - p_r x_{rj} + p_r' \frac{x_{rj}}{\rho}\right) -$$

$$-\left(P(x^*) - p_r x_r^* + p_r' \frac{x_r^*}{\rho}\right)\left(\Delta_j'' - d_r x_{rj} + d_r' \frac{x_{rj}}{\rho}\right) \qquad (2.11)$$

Egyértelmű, ha $p_r' = p_r \rho$ *és* $d_r' = d_r \rho$, akkor:

$$\Delta_j(x') \overset{(2.11)}{=} \Delta_j(x^*) \overset{(2.10)}{\geqslant} 0, \quad j = 1, \ldots, n.$$

Tehát az x' vektor optimális megoldása a (2.7)-(2.9) feladatnak.

Ha egy lehetséges A_r vektort helyettesítünk $A_r' = \rho A_r$-rel ($\rho > 0$), akkor egyszerűen csak ki kell cserélni a p_r és d_r együtthatókat az eredeti feladat $Q(x)$ célfüggvényében $p_r' = p_r \rho$ $d_r' = d_r \rho$ -re. Ez a két helyettesítés garantálja az egyenlőséget az eredeti és az új, skálázott LFP feladat között.

Egyértelmű, ha az x' vektor az új LFP feladat optimális megoldása, akkor:

$$x^* = (x_1', \; x_2', \; \ldots, \; x_{r-1}', \; x_r' \rho, \; x_{r+1}', \; \ldots, \; x_n')$$

az eredeti LFP optimális megoldása lesz.

Meg kell vizsgálnunk azt az esetet, amikor a helyettesített A_r vektor nem bázis vektor, $r \in J_N = J \setminus J_B$.

Ahogy a korábbi esetben, most is egyszerűen kicseréljük az eredeti feladat p_r és d_r együtthatóit ρp_r és ρd_r -re. Ha az r index nem bázis index, akkor $x_r^* = 0$, ebből következik, hogy $x' = x^*$, $P'(x') = P(x^*)$, $D'(x') = D(x^*)$ és ezért $Q'(x') = Q(x^*)$.

Tehát az $A_r \rightarrow \rho A_r$, $r \in J_N$ helyettesítés csak a Δ_r', Δ_r'' és $\Delta_r(x')$ értékeire van hatással.

Valóban, ha az eredeti LFP feladatban a nem bázisbeli A_r vektort vesszük, akkor:

$$\sum_{i=1}^{m} A_{s_i} x_{ir} = A_r, \quad j=1,\ldots,n,$$

ezután az $A_r \rightarrow A_r'$, helyettesítést végrehajtjuk, ahol $A_r' = \rho A_r$, és kapjuk a következő új A_r' reprezentációt egy hasonló *B* bázisban:

$$\sum_{i=1}^{m} A_{s_i}(\rho x_{ir}) = \rho A_r, \quad j=1,\ldots,n.$$

Amikor $A_r \rightarrow \rho A_r$ helyettesítést alkalmazzuk, akkor egyszerűen helyettesítjük $p_r \rightarrow p_r'$-vel, ahol $p_r' = \rho p_r$, és $d_r \rightarrow d_r'$-vel, ahol $d_r' = \rho d_r$, ezután az új $\tilde{\Delta}_r'$, $\tilde{\Delta}_r''$ és $\tilde{\Delta}_r(x')$ a következő:

$$\tilde{\Delta}_r' = \sum_{i=1}^{m} p_{s_i}(\rho x_{ir}) - (\rho p_r) = \rho \Delta_r',$$

$$\tilde{\Delta}_r'' = \sum_{i=1}^{m} d_{s_i}(\rho x_{ir}) - (\rho d_r) = \rho \Delta_r'',$$

$$\tilde{\Delta}_r(x') = D(x^*)\tilde{\Delta}_r' - P(x^*)\tilde{\Delta}_r'' =$$

$$= D(x^*)(\rho \Delta_r') - P(x^*)(\rho \Delta_r'') = \rho \Delta_r(x^*) \overset{(2.10)}{\geqslant} 0.$$

Ez azt jelenti,hogy az *x** vektor optimális megoldása az új LFP feladatnak.

Ha helyettesítünk egy nem bázis A_r vektort $A_r' = \rho A_r$-vel ($\rho>0$), akkor egyszerűen ki kell cserélni a p_r és d_r együtthatókat az eredeti *Q(x)* célfüggvényben $p_r' = \rho p_r$, és $d_r' = \rho d_r$-re. Ez a két helyettesítés garantálja az eredeti és az LFP feladat egyenlőségét. Sőt azt is, hogy $x_r^* = x_r' = 0$.

2.1.3. Sor helyettesítés: $a_i \rightarrow \rho a_i$

Az $a_r = (a_{r1}, a_{r2}, \ldots, a_{rn})$ sort cseréljük az $A = \|a_{ij}\|_{m\times n}$ mátrixból az $a_r' = \rho a_r$ sorvektorra. A következő 2 esetet különböztetjük meg:

1. Egyszerűen az $a_r \rightarrow \rho a_i$ kicserélésekor a *b* vektor *r.* elemét is helyettesítjük $b_r \rightarrow b_r' = \rho b_r$-re.
2. Más elemet nem módosítunk a *b*-ben, tehát a skálázást csak az *A* mátrixban kell

végrehajtani.

Az 1. esetben:

Az *r.* sorbeli eredeti megszorítások:

$$\sum_{j=1}^{n} a_{rj} x_j = b_r,$$

azaz:

$$\sum_{j=1}^{n} (\rho a_{rj}) x_j = (\rho b_r),$$

Ismert, hogy az ilyen skálázás nincs hatással a lehetséges megoldások *S* halmazának struktúrájára. Tehát az új skálzási probléma teljesen megegyezik az eredetivel.

A 2. esetben:

Nem módosítjuk a *b* vektort. Az ilyen skálázás előre nem látható deformációhoz vezet az S megoldások halmazában, vagyis nem tudjuk biztosítani, hogy a skálázott feladat optimális bázisa az eredetihez hasonló lesz.

Így az egyetlen járható skálázás az *A* mátrix sorain az $\tilde{a}_r \rightarrow \tilde{a}_r'$, ahol

$$\tilde{a}_r = (a_{r1}, a_{r2}, \ldots, a_{rn}, b_r),$$

$$\tilde{a}_r' = (\rho a_{r1}, \rho a_{r2}, \ldots, \rho a_{rn}, \rho b_r)$$

Nyílvánvaló, hogy az optimális megoldása az eredeti és a skálázott feladatnak, azaz az x^* és x', pontosan ugyanaz. Tehát nincs szükségünk skálázásra.

Jegyezzük meg, hogy a szimplex módszernek csak a tengely oszlop elemei hasonlóak. Ezért a tengely sor választása függ a sor skálázástól. A rossz választás nagy számolási hibához vezethet, ez azt jelenti, hogy fontos a megfelelő sorskálázás.

2.1.4. A számláló vektor skálázása: $p \rightarrow \rho p$

Cseréljük ki a $p=(p_0, \ldots, p_n)$ vektort a *Q(x)* célfüggvény *P(x)* számlálójában $p' = (p_0', p_1', \ldots, p_n')$ -ra, ahol $p_j' = \rho p_j$, $j=0, \ldots, n$.

Az ilyen csere nincs hatással sem a *D(x)* optimális értékére, sem a csökkentett Δ_j'' költségre, de változtat a *P(x)* és *Q(x)* függvények csökkentett Δ_j' és Δ_j költségén.

Tehát az új értékek: $\tilde{\Delta}_j'$, $P'(x^*)$, $Q'(x^*)$ *és* $\tilde{\Delta}_j(x^*)$.

$$\tilde{\Delta}_j' = \sum_{i=1}^{m} p'_{s_i} x_{ij} - p'_j =$$

$$= \sum_{i=1}^{m} (\rho p_{s_i}) x_{ij} - (\rho p_j) = \rho \Delta'_j, \quad j=1,\dots,n,$$

$$P'(x^*) = \sum_{j=1}^{n} p'_j x^*_j + p'_0 = \sum_{j=1}^{n} (\rho p_j) x^*_j + (\rho p_0) = \rho P(x^*),$$

$$Q'(x^*) = \frac{P'(x^*)}{D(x^*)} = \frac{\rho P(x^*)}{D(x^*)} = \rho Q(x^*),$$

ezért:

$$\tilde{\Delta}_j(x^*) = D(x^*)\tilde{\Delta}_j' - P'(x^*)\Delta''_j =$$

$$= D(x^*)(\rho \Delta_j') - (\rho P(x^*))\Delta''_j = \rho \Delta_j(x^*), \quad j=1,\dots,n.$$

Az utóbbi egyenlőségből kapjuk:

$$\tilde{\Delta}_j(x^*) = \rho \Delta_j(x^*) \overset{(2.10)}{\geqslant} 0, \quad j=1,\dots,n.$$

Végül jegyezzük meg, hogy a $p \rightarrow \rho p$ nem vezet semmilyen változáshoz sem az optimális bázisban, sem x^* optimális megoldásban.

Tehát, ha mi skálázatlanul akarjuk megoldani az LFP feladatot, akkor az optimális megoldást úgy kapjuk, hogy $Q(x^*) = \frac{1}{\rho} Q'(x^*)$, mivel a skálázott feladat x' optimális megoldása pontosan ugyanaz, mint az eredeti feladat x^* optimális megoldása.

2.1.5. A nevező vektor skálázása: $d \rightarrow \rho d$

Cseréljük ki a d=*(d_0, ..., d_n)* vektort a *Q(x)* célfüggvény D*(x)* számlálójában $d' = (d'_0, d'_1, \dots, d'_n)$ -ra, ahol $d'_j = \rho d_j$, *j=0, ..., n*.

Az ilyen csere nincs hatással sem a *P(x)* optimális értékére, sem a csökkentett Δ'_j költségre, de változtat a *D(x)* és *Q(x)* függvények optimális értékében és a Δ''_j és $\Delta_j(x)$ értékében.

Tehát az új értékek: $\tilde{\Delta}_j''$, $D'(x^*)$, $Q'(x^*)$ *és* $\tilde{\Delta}_j(x^*)$.

$$\tilde{\Delta}_j'' = \sum_{i=1}^{m} d'_{s_i} x_{ij} - d'_j =$$

$$= \sum_{i=1}^{m} (\rho d_{s_i}) x_{ij} - (\rho d_j) = \rho \Delta''_j, \quad j=1,\dots,n,$$

$$D'(x^*) = \sum_{j=1}^{n} d'_j x^*_j + d'_0 = \sum_{j=1}^{n} (\rho d_j) x^*_j + (\rho d_0) = \rho D(x^*),$$

$$Q'(x^*) = \frac{P(x^*)}{D'(x^*)} = \frac{P(x^*)}{\rho D(x^*)} = \frac{Q(x^*)}{\rho},$$

ezért:

$$\tilde{\Delta}_j(x^*) = D'(x^*)\Delta_j' - P(x^*)\tilde{\Delta}_j'' =$$

$$= (\rho D(x^*))\Delta_j' - P(x^*)(\rho \Delta_j'') = \rho \Delta_j(x^*), \quad j = 1, \ldots, n.$$

Az utóbbi egyenlőségből kapjuk:

$$\tilde{\Delta}_j(x^*) = \rho \Delta_j(x^*) \overset{(2.10)}{\geqslant} 0, \quad j = 1, \ldots, n.$$

Végül jegyezzük meg, hogy a $d \rightarrow \rho d$ nem vezet semmilyen változáshoz sem a B optimális bázisban, sem x^* optimális megoldásban.

Tehát, ha mi skálázatlanul akarjuk megoldani az LFP feladatot, akkor az optimális megoldást úgy kapjuk, hogy $Q(x^*) = \rho Q'(x^*)$, mivel a skálázott feladat x' optimális megoldása pontosan az eredeti feladat x^* optimális megoldásának felel meg.

2.1.6. Skálázó faktorok

Figyeljük meg az $A = \|a_{ij}\|_{m \times n}$ mátrixot és a $b = (b_1, \ldots, b_m)^T$ vektort.

Az $Ax=b$ rosszul skálázásának a mértékét adja meg a következő:

$$\sigma(A) = \frac{\max\limits_{i,\, j \in J_+} (|a_{ij}|)}{\min\limits_{i,\, j \in J_+} (|a_{ij}|)}$$

ahol $J_+ = \{i, j |\ a_{ij} \neq 0\}$. Minél nagyobb a különbség a legkisebb és a legnagyobb nem nulla a_{ij} elemek között, annál rosszabb a rendszer skálázása.

Definíció: Azt mondjuk, hogy egy A mátrix gyengén, vagy rosszul skálázott, ha $\sigma(A) > 1E+5$.

A skálázás célja, hogy a $\sigma(A)$ mértékét minél kisebbre csökkentse. Ahhoz, hogy ezt elérjük, az oszlopokat és a sorokat skálázzuk, amennyiszer csak szükséges.

2.1.6.1. Hall szabály

A szabálynak megfelelően definiáljuk a következő sorskálázó ρ^r oszlopvektort:

$$\rho^r = (\rho_1^r, \ \rho_2^r, \ \ldots, \ \rho_m^r)^T, \tag{2.12}$$

ahol:

$$\rho_i^r = \left(\prod_{j \in J_i^+} a_{ij} \right)^{\frac{1}{K_i^r}}, \quad i=1,\ldots,m;$$

$J_i^+ = \{ j : a_{ij} \neq 0 \}$, $i=1, \ \ldots,m$, sort meghatározó index halmaz, mely azokat a j indexeket tartalmazza, ahol az $a_{ij} \neq 0$ az i. sorban,és a K_i^r az i. sor nem nulla a_{ij} elemeinek a számát jelenti.

Hasonló módon az oszlop skálázáshoz a következő ρ^c sorvektor kell:

$$\rho^c = (\rho_1^c, \ \rho_2^c, \ \ldots, \ \rho_n^c)^T, \tag{2.13}$$

ahol:

$$\rho_j^c = \left(\prod_{j \in J_j^+} a_{ij} \right)^{\frac{1}{K_j^c}}, \quad j=1,\ldots,n;$$

$J_j^+ = \{ i : a_{ij} \neq 0 \}$, j=$1, \ \ldots,n$, sort meghatározó index halmaz, mely azokat a i indexeket tartalmazza, ahol az $a_{ij} \neq 0$ a j. sorban, és a K_j^c a j. oszlop nem nulla a_{ij} elemeinek a számát jelenti.

2.1.6.2. Gondzio szabály

Ez a faktor számolás a Hall szabálynak megfelelően működik, definiálni tudjuk a sorok skálázó faktorának ρ^r oszlopvektorát.

$$\rho^r = (\rho_1^r, \ \rho_2^r, \ \ldots, \ \rho_m^r)^T, \tag{2.14}$$

ahol:

$$\rho_i^r = \sqrt{r_i' r_i''}, \quad i=1,\ldots,m;$$

és:

$$r_i' = \max_{j : a_{ij} \neq 0} |a_{ij}|, \quad r_i'' = \min_{j : a_{ij} \neq 0} |a_{ij}|, \quad i=1, \ \ ,m.$$

Hasonlóan a sorskálázó faktorhoz, az oszlopokhoz is definiálhatunk egy ρ^c skálázó sorvektort.

$$\rho^c = (\rho_1^c, \ \rho_2^c, \ \ldots, \ \rho_n^c)^T, \tag{2.15}$$

ahol:

$$\rho_j^c = \sqrt{c_j' c_j''}, \quad j=1,\ldots,n;$$

és:

$$c_j' = \max_{i:a_{ij}\neq 0} |a_{ij}|, \quad c_j'' = \min_{i:a_{ij}\neq 0} |a_{ij}|, \quad j=1, \ , n.$$

2.1.6.3. Implementáció

Egy LFP probléma skálázásakor ki tudjuk számolni és tárolni tudjuk a sorok, oszlopok és a célfüggvény skálázó faktorát (külön a számlálóját és külön a nevezőjét).

A faktorok tárolásának egyik lehetséges módja, hogy kiterjesztjük a feladat mátrixát a következő módon:

$$\tilde{A} = \left(\begin{array}{c|cccc|c} \rho_1^r & a_{11} & a_{12} & \ldots & a_{1n} & b_1 \\ \rho_2^r & a_{21} & a_{22} & \ldots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \ldots & \vdots & \vdots \\ \rho_m^r & a_{m1} & a_{m2} & \ldots & a_{mn} & b_n \\ \hline \rho_{m+1}^r & p_1 & p_2 & \ldots & p_n & p_0 \\ \rho_{m+2}^r & d_1 & d_2 & \ldots & d_n & d_0 \\ \hline & \rho_1^c & \rho_2^c & \ldots & \rho_n^c & \rho_{n+1}^c \end{array}\right)$$

Jegyezzük meg, hogy sok LP kód ahelyett, hogy pontosan kiszámolná a ρ skálázó faktort, ezen értékek bináris közelítéseiből a legközelebbit használják. Ennek az az oka, hogy a számítógépek bináris rendszerek. Ez sokat javíthat a skálázás hatékonyságán, mivel ebben az esetben a meglehetősen drága szorzás művelete helyettesíthető a sokkal gyorsabb eltolással.

2.2. Bázis mátrix faktorizációja

A legtöbb LP és LFP alkalmazás több ezernyi ismeretlen változót és feltételt tartalmazhat. Az erősen skálázott problémák rendszerint több ezer iterációt, és több millió lebegőpontos számolást igényelnek a szimplex metódusban. Mivel a számítógépek számolási pontossága véges, ezért ezekben apró számítási hibák léphetnek fel. A szimplex módszer iteratív jellege miatt ez a hiba, pontatlanság kommulatív hatást eredményez, végül lényegesen nagyobb hibához vezet.

Ez a típusú hiba az egész szimplex metódust végigköveti a transzformáció során, amikor a következő elemeket számoljuk:

$$x'_{ij} = \begin{cases} x_{ij} - \dfrac{x_{rj}x_{ik}}{x_{rk}}, & i=1,\ldots,m, \quad i \neq r, \\ \dfrac{x_{rj}}{x_{rk}}, & i = r, \qquad j \in J_N = J \setminus J_B \end{cases}$$

ahol $J=\{1,...,n\}$ minden A_j vektor indexeinek a halmaza, ahol $J_B=\{s_1,...,s_m\}$ bázisvektor indexhalmaza, r annak a vektornak az indexe, amelyiket kivisszük a bázisból, és k a bejövő új vektor indexe.

Ez a formula a szimplex metódus végrehajtása közben kapcsolatot biztosít a régi és az új x_{ij} együtthatók között amikor vektort cserélünk a táblában, és újraszámoljuk az új x'_{ij} elemet a régi x_{ij}-ből. Az iterációkban végrehajtott véges pontosságú számolásokból fakadó kicsi kerekítési hibák helytelen megoldást eredményezhetnek. Ezért számoljuk újra a fő x_{ij} elemeket a szimplex metódus végrehajtásakor, közvetlenül az eredeti A mátrixból és az aktuális $B = (A_{S_1},\ldots,A_{S_m})$ bázisból.

Oldjuk meg a következő lineáris egyenletrendszert:

$$\sum_{i=1}^{m} A_{s_i} x_{ij} = A_j \quad , ahol\ j \in J_N = J \setminus J_B.$$

azaz, más szóval $Ax=b$ alakú lineáris egyenletrendszerek halmaza, melynek bal oldalán Ax, jobb oldalán pedig b vektor.

$$\underbrace{\underbrace{Ax}}_{LHS} = \underbrace{b}_{RHS}$$

Nem a legjobb választás Gauss eliminációt vagy Gauss – Jordan metódust alkalmazni a hasonló feladatok megoldására, mert mindkét metódus esetében ismerni kell az RHS vektort mielőtt végrehajtjuk a számolást. Másik ok, hogy mindkét metódus drága és kb. $m^2/2+m^2$ lebegőpontos számolást igényel az eredeti négyzetes mátrix csökkentéséhez és a megmaradt helyettesítések végrehajtásához. Tehát ezek a metódusok $O(m^3)$ mértékben drágák.
Az LU faktorizációnál nem kell előre ismerni az RHS vektort, és sokkal hatékonyabb is.

2.2.1. LU-faktorizáció

Az $Ax=b$ alakú lineáris egyenletrendszer megoldásának egyik módjával foglalkozom ebben a fejezetben.

$$Ax = b \tag{2.16}$$

Az A mátrix egy invertálható, ($m \times m$)-es négyzetes mátrix, és a b vektor egy tetszőleges m elemű $b=(b_1,...b_m)^T$ oszlopvektor. Az A mátrix A^{-1} inverzét könnyű kiszámolni, tehát a (2.16)-tal jelölt rendszer eredményét $A^{-1}b$ adja meg. Sajnos ritka az az eset, hogy a mátrix alkalmas legyen az invertálásra a lineáris egyenlet megoldása céljából. A legtöbb közvetlen módszer az együttható mátrix egy faktorizációját alkalmazza a megoldáshoz. Az egyik legismertebb, és széles körben használt faktorizáció a nem szimmetrikus rendszerekhez alkalmazható. Ez az LU-faktorizáció, ahol az A mátrixot úgy határozzák meg, hogy van egy L alsó háromszög mátrix, és egy U felső háromszög mátrix.

$$P_r A\, P_C = LU, \tag{2.17}$$

ahol:

$$L = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ l_{m1} & l_{m2} & \dots & l_{mm} \end{pmatrix}, \qquad U = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1m} \\ 0 & u_{22} & \dots & u_{2m} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & u_{mm} \end{pmatrix}$$

és a P_r, P_c permutációs mátrixok, melyekkel a sorokat és oszlopokat egyenként meg lehet cserélni.

Azután, hogy az A mátrixot LU alakba bontottuk, a következő 2 lépést kell végrehajtani a megoldáshoz:

$$Ly = P_r b \tag{2.18}$$

$$Uz = y \tag{2.19}$$

így az x megoldás a z vektorból kiszámítható egy permutációval:

$$x = P_c z. \tag{2.20}$$

Az LU felbontás nagyon hasznos, mert háromszög mátrix alakú megoldás esetén a megfelelő lineáris egyenletrendszer helyettesítéseit könnyű meghatározni.

Az $Ax=b$ egyenletrendszer megoldásának algoritmusát az LU-dekompozícióval a következőképpen lehetne megfogalmazni:

1. A mátrix felbontása: $A = LU$.
2. $LUx = b$ egyenletrendszerben jelölje az Ux-et y, és számoljuk ki az $Ly = b$ rendszerből y-t: $y = L^{-1}b$.
3. A már ismert y segítségével határozzuk meg az $Ux = y$ egyenletbeli x értékét: $x = U^{-1}(L^{-1}b)$.

Nem minden mátrixot lehet LU alakban felbontani.

Példa 1.:

$$A = \begin{pmatrix} 0 & 4 \\ 4 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_{21} & 1 \end{pmatrix}\begin{pmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{pmatrix}, \qquad u_{11} = 0 \quad \text{és} \quad l_{21}\,u_{11} = 4.$$

De ez nem megfelelő, ezért létrehozzuk a B mátrixot, amit az A mátrixból kapunk a sorok felcserélésével.

$$B = \begin{pmatrix} 4 & 2 \\ 0 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 4 & 2 \\ 0 & 4 \end{pmatrix}.$$

Ez már felbontható lesz, mivel az átlójában nincs 0 elem.

A mátrix sorainak ez a fajta újrarendezése mindig lehetséges, ha a mátrix nem szinguláris, azaz determinánsa nem 0, ezért $Ax = b$ egyenletrendszernek egy megoldása van. A nem szingularitás nem szükséges feltétele az LU felbontás létezésének.

Példa 2.: Szinguláris mátrix:

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Ennek LU felbontása:

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} = LU.$$

Az első példában látható probléma miatt alkalmazzuk a (2.17)-es formulában a két permutációs mátrixot az LU-felbontáshoz.

Tétel: (Az LU-felbontás létezése) Ha az A négyzetes mátrix nem szinguláris, akkor létezik olyan P_r és P_c permutációk, egy L alsó háromszög mátrix, és egy nem szinguláris U felső háromszög mátrix, hogy $P_r A P_C = LU$.

Ez azt jelenti, hogy mielőtt végrehajtjuk az LU felbontást, ellenőriznünk kell, hogy minden átlóbeli elem nem nulla-e, különben sorcseréket kell végezni úgy, hogy az átlóban ne legyen 0. Egy mátrixhoz nem csak egy LU-felbontás létezik.

LU-felbontás végrehajtása:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ l_{m1} & l_{m2} & \dots & l_{mm} \end{pmatrix}\begin{pmatrix} u_{11} & u_{12} & \dots & u_{1m} \\ 0 & u_{22} & \dots & u_{2m} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & u_{mm} \end{pmatrix} = LU.$$

A felbontás műveletei:

$$a_{ij} = l_{i1}u_{1j} + \dots, \quad i = 1, \dots, m.$$

Ha i = 1, akkor

$$a_{1j} = l_{i1}u_{1j}, \quad j = 1, \dots, m.$$

Ha i = 2, akkor:

$$a_{21} = l_{21}u_{11};$$
$$a_{22} = l_{21}u_{12} + l_{22}u_{22.}$$

Ha i = 3, akkor:

$$a_{31} = l_{31}\,u_{11};$$
$$a_{32} = l_{31}\,u_{12} + l_{32}\,u_{22};$$
$$a_{33} = l_{31}\,u_{13} + l_{32}\,u_{23} + l_{33}\,u_{33.}$$

A tagok száma az öszegben függ attól, hogy az *i* vagy a *j* index a kisebb.

$$l_{i1}u_{1j} + l_{i2}u_{2j} + \ldots + l_{ii}u_{ij} = a_{ij}, \qquad ha \quad i < j; \tag{2.21}$$

$$l_{i1}u_{1j} + l_{i2}u_{2j} + \ldots + l_{ii}u_{jj} = a_{ij}, \qquad ha \quad i = j; \tag{2.22}$$

$$l_{i1}u_{1j} + l_{i2}u_{2j} + \ldots + l_{ij}u_{jj} = a_{ij}, \qquad ha \quad i > j; \tag{2.23}$$

Láthatjuk, hogy ez az egyenletrendszer m^2 egyenletet tartalmaz, és m^2+m l_{ij} és u_{ij} ismeretlent (a főátlóbeli elemek kétszer szerepelnek). Mivel az ismeretlenek száma nagyobb, mint az egyenleteké, rögzíthetjük bármely m db l_{ij} és u_{ij} ismeretlenek értékét, tetszőleges értéket adva nekik, és megoldjuk a rendszert a többi, nem rögzített imseretlenre. A következő értéket mindig adhatjuk:

$$l_{ii} = 1, \quad i = 1,\ldots,m.$$

A következő eljárást, ami az előbbi egyenletrendszert oldja meg, Crout algoritmusnak hívjuk:

1. Legyen $l_{ii} = 1$, minden $i = 1,...,m$ esetén.
2. Minden egyes $j = 1,...,m$ esetén végrehajtjuk a következő 2 lépést:
   1. Először használjuk a (2.21)-(2.23) rendszert a következő érték meghatározására *(i= 1,..j):*

$$u_{ij} = \begin{cases} a_{ij}, & ha \quad i = 1; \\ a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj}, & ha \quad i > 1; \end{cases} \tag{2.24}$$

   2. Ezután a (2.23)-as egyenlet segítségével, $i=j+1, \ldots,m$ esetekben számoljuk ki a következő értéket:

$$l_{ij} = \frac{(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj})}{u_{jj}}. \tag{2.25}$$

A Crout algoritmus iterációit végrehajtva megfigyelhetjük, hogy az ismeretlen l_{ij} és u_{ij} elemek, melyek a képlet jobb oldalán szerepelnek, mindig meghatározódnak, mire szükség

van rájuk. Láthatjuk, hogy az eredeti A mátrix minden a_{ij} elemét pontosan egyszer használjuk fel. Ez azt jelenti, hogy a megfelelő l_{ij} és u_{ij} elemeket ugyanott tudjuk tárolni a számítógép memóriájában, ahol az eredeti a_{ij} elem foglalt helyet. Tehát az LU-dekompozíciót helyben végre lehet hajtani. Figyeljük meg azt is, hogy a főátlóban lévő l_{ij} egységelemeket sem kell letárolni (hiszen mindegyik értéke 1). Más szóval, a Crout algoritmus megengedi, hogy az eredeti A mátrix átalakítását LU alakba, helyben végezzük, anélkül, hogy extra memóriát igényelnénk, épp ezért a memória felhasználása hatékony.

Megfigyelhetjük, hogy a (2.25)-ös egyenlőség tartalmaz osztást, ami lényeges és kihagyhatatlan része a Crout algoritmusnak. Az egyszerűbb, és hatékonyabb megvalósítás érdekében lehetőség van arra, hogy csak rész tengelykiválasztást végezzünk, azaz csak sorcseréket alkalmazunk.

Az LU felbontás elkészítése a Crout algoritmussal kb. $m^3/3$ szorzást és ugyanennyi összeadást igényel. Az $Ly=b$ és $Ux=y$ helyettesítések pedig m^2-et. Tehát az $Ax=b$ egyenletrendszer, melynek csak egyetlen b RHS vektora van (vagy csak kevés különböző), sokkal kevesebb művelet elvégzését igényli, mint a Gauss-elimináció vagy a Gauss-Jordan metódus. Sőt, ha $Ax=b$ egyenletrendszerek halmazát kell megoldanunk összetett RHS vektorokkal, akkor előnyösebb az LU-felbontást alkalmazni, mivel ebben az esetben a felbontást csak egyszer végezzük el annyiszor használhatjuk, ahány RHS vektor van. Az LU-felbontás $O(m^3)$, a visszahelyettesítés pedig $O(m^2)$ műveletet igényel, ezért nyílvánvaló, hogy a helyettesítések gyorsabbá teszik az LU-felbontást. Minél nagyobb a megoldadnó egyenletrendszerek mérete, annál nagyobb részt tesznek ki ki az LU-faktorizáció lépései az összes műveletből. Ezért fontos, hogy olyan kevés LU-felbontást hajtsunk végre, amennyit csak lehetséges.

Fontos, hogy egy mátrix LU felbontása megegyezik a Gauss-eliminációval, abból a szempontból, hogy az A matrixot balról szorozva L^{-1}-gyel, ugyanazt a hatást érjük el, mint az elemi sorátalakításokkal, azaz megkapjuk az U felső háromszög mátrixot.

2.2.2. LU-faktorizáció és Gauss-elimináció

Egy A mátrix LU felbontása nagyon közel áll a Gauss eliminációhoz. A két eljárás

kombinálható egymással.

Először is hogyan alakítunk át egy adott A mátrixot felső háromszög mátrixszá Gauss eliminációval? Első lépésben helyettesítjük az i. sort az $(A \mid b)$ bővített mátrixban a következő kifejezéssel:

$$(i.\ sor) - (1.\ sor)\mu_{i1}, \text{ ahol} \quad \mu_{i1} = \frac{a_{i1}}{a_{11}}$$

Mátrix jelölésekkel leírva:

$$A^{(2)} = M^{(1)} A \tag{2.26}$$

ha, az adott A mátrix a következő:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1\mathrm{m}} \\ a_{21} & a_{22} & \dots & a_{2\mathrm{m}} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{pmatrix},$$

és a (2.26)-ös formulában szereplő többi mátrix pedig:

$$A^{(2)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1\mathrm{m}} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \dots & a_{2\mathrm{m}}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & \dots & a_{3\mathrm{m}}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{m2}^{(2)} & a_{m3}^{(2)} & \dots & a_{mm}^{(2)} \end{pmatrix}, \qquad M^{(1)} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ -\mu_{21} & 1 & 0 & \dots & 0 \\ -\mu_{31} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\mu_{m1} & 0 & 0 & \dots & 1 \end{pmatrix}$$

A második lépésben létrehozzuk az $M^{(2)}$ mátrixot:

$$M^{(2)} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & -\mu_{32} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -\mu_{m2} & 0 & \dots & 1 \end{pmatrix},$$

ahol:

$$\mu_{i2} = \frac{a_{i2}^{(2)}}{a_{22}^{(2)}}, \qquad i = 3,4,\dots,m.$$

Ezután kifejezzük $A^{(3)}$ mátrixot:

$$A^{(3)} = M^{(2)} M^{(1)} A \tag{2.27}$$

Ha az a_{11} vagy a_{22} nullák, akkor még a végrehajtás előtt kénytelenek vagyunk a sorokat megfelelő módon felcserélni, azaz sor permutációs mátrixot kell alkalmazni. Az előzőeket ennek megfelelően felírva:

$$A^{(2)} = M^{(1)} P^{(1)} A, \tag{2.28}$$

$$A^{(3)} = M^{(2)} P^{(2)} M^{(1)} P^{(1)} A, \tag{2.29}$$

ahol $P^{(1)}$ megfelelő m-ed rendű permutációs mátrix, melyet az A mátrixbeli sorok cseréjére alkalmazunk, és $P^{(2)}$ szintén ilyen permutációs mátrix, melyet az $A^{(2)}$ mátrixbeli sorok cseréjére alkalmazunk.

Általánosítva:

$$A^{(k+1)} = M^{(k)} P^{(k)} M^{(k-1)} P^{(k-1)} \ldots M^{(1)} P^{(1)} A.$$

A Gauss elimináció végeredménye:

$$A^{(m)} = M^{(m-1)} P^{(m-1)} M^{(m-2)} P^{(m-2)} \ldots M^{(1)} P^{(1)} A, \tag{2.30}$$

ahol:

$$A^{(m)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \ldots & a_{1m}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \ldots & a_{2m}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \ldots & a_{3m}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & a_{mm}^{(m)} \end{pmatrix} = U, \tag{2.31}$$

és $a_{1j}^{(1)} = a_{1j}, \quad j = 1, 2, \ldots, m.$

Szorozzuk be balról a (2.30)-es képletet $(M^{(m-1)})^{-1}$ inverzzel, majd $(P^{(m-1)})^{-1}$-gyel:

$$(P^{(m-1)})^{-1} (M^{(m-1)})^{-1} A^{(m)} = (P^{(m-1)})^{-1} (M^{(m-1)})^{-1} M^{(m-1)} P^{(m-1)} M^{(m-2)} P^{(m-2)} \ldots M^{(1)} P^{(1)} A.$$

Mivel $(M^{(m-1)})^{-1} M^{(m-1)} = I$ és $(P^{(m-1)})^{-1} P^{(m-1)} = I$, ezért:

$$(P^{(m-1)})^{-1} (M^{(m-1)})^{-1} A^{(m)} = M^{(m-2)} P^{(m-2)} \ldots M^{(1)} P^{(1)} A.$$

Ezután az előző lépéseket ismételjük, míg végül a következőt akpjuk:

$$(P^{(1)})^{-1}\,(M^{(1)})^{-1}\,\ldots\,(P^{(m-1)})^{-1}\,(M^{(m-1)})^{-1}A^{(m)} = A,$$

azaz:

$$(P^{(1)})^{-1}\,(M^{(1)})^{-1}\,\ldots\,(P^{(m-1)})^{-1}\,(M^{(m-1)})^{-1}U = A. \tag{2.32}$$

Vegyük a következő mátrixot:

$$M^{(k)} = \begin{pmatrix} 1 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \iddots & \vdots & \vdots \\ 0 & \ldots & 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & \ldots & 0 & -\mu_{k+1,k} & 1 & \ldots & 0 & 0 \\ 0 & \ldots & 0 & -\mu_{k+2,k} & 0 & \ldots & 0 & 0 \\ \vdots & \iddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & -\mu_{m,k} & 0 & \ldots & 0 & 1 \end{pmatrix}, \tag{2.33}$$

amely megfelel a Gauss elimináció *k.* lépésének, és a következő kivonást jelenti:

$$(i.\ sor) - (k.\ sor - 1)\mu_{ki},\ i = k, k+1, \ldots, m.$$

Láthatjuk, hogy az inverz műveletet úgy kapjuk, hogy az *i.* sorhoz μ_{ki}-szer hozzáadjuk a *(k.sor–1)*-et, ahol $i = k, k+1, \ldots, m$. Ezért az $M^{(k)}$ inverze:

$$(M^{(k)})^{-1} = \begin{pmatrix} 1 & \ldots & 0 & 0 & 0 & \ldots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \iddots & \vdots & \vdots \\ 0 & \ldots & 1 & 0 & 0 & \ldots & 0 & 0 \\ 0 & \ldots & 0 & 1 & 0 & \ldots & 0 & 0 \\ 0 & \ldots & 0 & \mu_{k+1,k} & 1 & \ldots & 0 & 0 \\ 0 & \ldots & 0 & \mu_{k+2,k} & 0 & \ldots & 0 & 0 \\ \vdots & \iddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & \mu_{m,k} & 0 & \ldots & 0 & 1 \end{pmatrix}, \tag{2.34}$$

Könnyen ellenőrizhetjük, hogy $(M^{(k)})^{-1}\,M^{(k)} = M^{(k)}\,(M^{(k)})^{-1} = I$. Tehát minden egyes $M^{(k)}$ mátrix inverzét előállíthatjuk úgy, hogy az átlótól különböző elemek előjelét megváltoztatom, és minden $(M^{(k)})^{-1}$ is alsó háromszög mátrix lesz.

Tudjuk, hogy két alsó háromszög mátrix szorzatának eredménye is alsó háromszög mátrix, tehát:

$$(M^{(1)})^{-1}\,(M^{(2)})^{-1}\,\ldots\,(M^{(m-1)})^{-1} = L. \tag{2.35}$$

Vegyük a következő mátrixot:

$$\tilde{L} = (P^{(1)})^{-1}(M^{(1)})^{-1}\ldots(P^{(m-1)})^{-1}(M^{(m-1)})^{-1} \tag{2.36}$$

a (2.32)-es egyenlőségből. Mivel P permutációs mátrixnál $P = P^{-1}$, ezért írhatjuk az előbbi formulát a következő módon:

$$\tilde{L} = P^{(1)}(M^{(1)})^{-1}\ldots P^{(m-1)}(M^{(m-1)})^{-1} \tag{2.37}$$

Ennek ellenére sajnos nem mondhatjuk, hogy $\tilde{L}$ alsó háromszög mátrix.

Ha követjük a $P^{(1)}, P^{(2)}, \ldots, P^{(m-1)}$ permutációs mátrixokkal végrehajtott sorcseréket, akkor a $\tilde{A}$ mátrixot kapjuk, ami ugyanaz, mint az A mátrix, csak a sorai fel vannak cserélve. Ennek a permutált $\tilde{A}$ mátrixnak a következő felbontása van:

$$\tilde{A} = LU,$$

ahol az L alsó háromszög mátrixot a (2.35)-es kifejezés adja, az U felső háromszög mátrixot, pedig a (2.31).

A permutációs mátrix nem csak arra való, hogy elkerüljük vele az átlóbeli nulla elemeket, hanem a számítás potnosságát is javítja. A számítógépek véges pontossága miatt gyakran helytelen választás lehet a k. lépésben, ha olyan sort választunk, melyben az átlóbeli elemének az abszolút értéke a legnagyobb, azaz:

$$|a_{kk}^{(k)}| = \max_i \{|a_{ik}^{(k)}|: \; i = k, k+1, \ldots, m\}.$$

2.2.3. LU-faktorizáció frissítése

Mivel a szimplex módszer bázis mátrixa nem változik egyik iterációról a másikra nagy mértékben , ezért egyértelmű, hogy a számítások végrehajtását javíthatjuk, ha elkerüljük az új bázis LU alakra bontását, amikor csak lehetséges, és helyette újrahasználnák a már létező, A mátrixhoz tartozó LU felbontást (amit korábbi lépésben kaptunk meg) valahol a későbbi lépések folyamán. Számos frissítésre szolgáló eljárás van. Ezek akkor alkalmazhatóak, ha az A mátrix alig módosul a további lépéseknél.

2.2.3.1. Alapok

Szükséges jelölések:

B: Aktuális bázis (melyhez kiszámoltuk az LU felbontást).

$\tilde{B}$: A következő iteráció bázisa.

Az új $\tilde{B}$ bázis csak 1 oszlopban különbözik B-től, ez a j. oszlop, amely a B bázis következő oszlopvektorát tartalmazza:

$$A_r = (a_{1r}, a_{2r}, \ldots, a_{mr})^T,$$

amely a kilépő x_r változóhoz tartozik, és az új $\tilde{B}$ bázisban az A_r oszlop kicserélődik a következőre:

$$A_k = (a_{1k}, a_{2k}, \ldots, a_{mk})^T,$$

az új x_k bázis változóhoz tartozik. A mátrixok jelölését alkalmazva a következőképpen fejezhetjük ki az előbbieket:

$$\tilde{B} = B + (A_k - A_r)e_j^T, \tag{2.38}$$

ahol e_j a j. oszlopát jelenti az m-edrendű I egységmátrixnak.

Példa 3.:

$$B = \begin{pmatrix} a_{12} & a_{11} & a_{13} \\ a_{22} & a_{21} & a_{23} \\ a_{32} & a_{31} & a_{33} \end{pmatrix}$$

Cseréljük fel $a_2 = (a_{12}, a_{22}, a_{32})^T$ oszlopot egy másik oszloppal, c-vel: $c = (c_1, c_2, c_3)^T$

$$\tilde{B} = \begin{pmatrix} a_{12} & a_{11} & a_{13} \\ a_{22} & a_{21} & a_{23} \\ a_{32} & a_{31} & a_{33} \end{pmatrix} + \left(\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} - \begin{pmatrix} a_{12} \\ a_{22} \\ a_{32} \end{pmatrix} \right) (1, \quad 0, \quad 0) =$$

$$\tilde{B} = \begin{pmatrix} a_{12} & a_{11} & a_{13} \\ a_{22} & a_{21} & a_{23} \\ a_{32} & a_{31} & a_{33} \end{pmatrix} + \begin{pmatrix} c_1 - a_{12} & 0 & 0 \\ c_2 - a_{22} & 0 & 0 \\ c_3 - a_{32} & 0 & 0 \end{pmatrix} = \begin{pmatrix} c_1 & a_{11} & a_{13} \\ c_2 & a_{21} & a_{23} \\ c_3 & a_{31} & a_{33} \end{pmatrix}.$$

Két nem nulla $u = (u_1, u_2, \ldots, u_m)^T$ és $v = (v_1, v_2, \ldots, v_m)^T$ vektorok uv^T külső szorzatának hozzáadása vagy kivonása a $A = \|a_{ij}\|_{m \times m}$ mátrixhoz, a következő módon írható le mátrix jelöléseket használva:

$$\tilde{A} = A \pm uv^T.$$

Ezt elsőrendű módosításnak nevezzük, mivel a mátrix külső szorzatának csak egy lineárisan független oszlopa vagy sora van.

$$uv^T = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix} \begin{pmatrix} v_1, & v_2, & \ldots, & v_m \end{pmatrix} = \begin{pmatrix} u_1 v_1 & u_1 v_2 & \ldots & u_1 v_m \\ u_2 v_1 & u_2 v_2 & \ldots & u_2 v_m \\ \vdots & \vdots & \ddots & \vdots \\ u_m v_1 & u_m v_2 & \ldots & u_m v_m \end{pmatrix}$$

Ha egy A mátrix valamely a_{ij} elemét változtatjuk, akkor $a_{ij} \rightarrow \tilde{a}_{ij} = a_{ij} + \alpha$, így az új mátrix:

$$\tilde{A} = A + \alpha e_i e_j^T, \tag{2.39}$$

ahol e_i és e_j az i. és j. oszlopa az egységmátrixnak.

Tegyük fel, hogy az aktuális B bázis most már megfelel a (2.38)-tel jelölt képletnek, és alkalmas az LU felbontás frissítéséhez.

2.2.3.2. Bartels-Golub frissítés

Az ötlet az, hogy újrarendezünk egy, már létező LU felbontást, és kihasználjuk a struktúráját. Vegyük az aktuális B bázis megfelelő LU-felbontását, amely $A_1, A_2, \ldots, A_m$ oszlopvektorokból áll, ahol:

$$A_j = (a_{1j}, a_{2j}, \ldots, a_{mj})^T, \quad j = 1, 2, \ldots, m,$$

és:

$$M^{(m-1)} P^{(m-1)} M^{(m-2)} P^{(m-2)} \ldots M^{(1)} P^{(1)} B = U,$$

ahol $M^{(i)}$, $i = 1, \ldots, m-1$, a Gauss transzformáció mátrixa, $P^{(i)}$, $i = 1, \ldots, m-1$, egy permutációs mátrix, amely biztosítja az $M^{(i)}$ Gauss transzformáció végrehajtása előtt a maximális tengely értéket, és az U az eredményül kapott, felső háromszög mátrix:

$$U = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \ldots & a_{1m}^{(1)} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \ldots & a_{2m}^{(2)} \\ 0 & 0 & a_{33}^{(3)} & \ldots & a_{3m}^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & a_{mm}^{(m)} \end{pmatrix}. \tag{2.40}$$

Ahhoz, hogy az U felső háromszög mátrixban elkerüljük a szükségtelen indexeléseket, írjuk fel a következő módon:

$$U = (U_1, U_2, \ldots, U_m) = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \ldots & u_{1m} \\ 0 & u_{22} & u_{23} & \ldots & u_{2m} \\ 0 & 0 & u_{33} & \ldots & u_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & u_{mm} \end{pmatrix}. \tag{2.41}$$

Tegyük fel, hogy a B egyik oszlopát le kell cserélni. Ez az oszlop legyen az A_r, ami a B mátrix r. oszlopa. A helyette bejövő új vektor pedig legyen az A_k. Tehát a bázis:

$$B = (A_1, A_2, \ldots, A_{r-1}, A_r, A_{r+1}, \ldots, A_m)$$

A transzformáció után:

$$\tilde{B} = (A_1, A_2, \ldots, A_{r-1}, A_k, A_{r+1}, \ldots, A_m)$$

Hogy megőrizzük a már létező LU felbontás háromszög alakját, újrarendezzük a $\tilde{B}$ bázist úgy, hogy az A_k jobb oldalán lévő összes vektort eggyel balra toljuk, és az A_k oszlopvektor lesz a mátrix legjobboldalibb vektora. Az újrarendezett $\tilde{B}$ bázis:

$$\tilde{B} P^{(R)} = (A_1, A_2, \ldots, A_{r-1}, A_{r+1}, A_{r+2}, \ldots, A_m, A_k), \tag{2.42}$$

ahol a $P^{(R)}$ egy megfelelő permutációs mátrix.

Jelölje f a következő szorzatot:

$$f = M^{(m-1)} P^{(m-1)} M^{(m-2)} P^{(m-2)} \ldots M^{(1)} P^{(1)} A_k. \tag{2.43}$$

Ha az új, permutált $\tilde{B}$ bázist megszorozzuk balról a következővel:

$$M^{(m-1)} P^{(m-1)} M^{(m-2)} P^{(m-2)} \ldots M^{(1)} P^{(1)},$$

akkor kapjuk:

$$M^{(m-1)} P^{(m-1)} M^{(m-2)} P^{(m-2)} \ldots M^{(1)} P^{(1)} \tilde{B} P^{(R)} = U', \tag{2.44}$$

ahol:

$$U' = (U_1, U_r, \ldots, U_{r-1}, U_{r+1}, \ldots, U_m, f) \tag{2.45}$$

vagy:

$$U' = \begin{pmatrix} U_{1,1} & \dots & U_{1,r-1} & U_{1,r+1} & U_{1,r+2} & \dots & U_{1,m} & f_1 \\ 0 & \dots & U_{2,r-1} & U_{2,r+1} & U_{2,r+2} & \dots & U_{2,m} & f_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & U_{r-1,r-1} & U_{r-1,r+1} & U_{r-1,r+2} & \dots & U_{r-1,m} & f_{r-1} \\ 0 & \dots & 0 & U_{r,r+1} & U_{r,r+2} & \dots & U_{r,m} & f_r \\ 0 & \dots & 0 & U_{r+1,r+1} & U_{r+1,r+2} & \dots & U_{r+1,m} & f_{r+1} \\ 0 & \dots & 0 & 0 & U_{r+2,r+2} & \dots & U_{r+2,m} & f_{r+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & 0 & 0 & \dots & U_{m,m} & f_m \end{pmatrix}.$$

Ennek az U' mátrixnak speciális struktúrája van (még mindig felső háromszög mátrix, csak van benne a főátló mellett egy részátló: $u_{r+1,r+1}$, $u_{r+2,r+2}$, ..., $u_{m,m}$), viszonylag könnyen visszaalakítható tisztán felső háromszög alakú mátrixszá a megfelelő Gauss transzformációkkal. Ehhez a következő műveletek ismétlésére van szükség, méghozzá $j = r, r+1, \dots, m-1$ értékekre:

1. **lépés:** Alkalmazzuk a $\tilde{P}_j$ permutációs mátrixot a j. és j+1. Sorokra úgy, hogy:

$$\tilde{u}_{j,j} \geqslant \tilde{u}_{j+1,j}. \tag{2.46}$$

   Jegyezzük meg, hogy $\tilde{P}_j$ egy m-edrendű egységmátrix, ha nem szükséges a csere, akkor:

$$u_{j,j} \geqslant u_{j+1,j}.$$

2. **lépés:** Hajtsuk végre a $\tilde{M}^{(j)}$ Gauss transzformációt, hogy lenullázzuk a $\tilde{u}_{j+1,j}$ elemet.

$$\tilde{M}^{(j)} = I - (\underbrace{0,0,\dots,0}_{j}, \frac{\tilde{u}_{j+1,j}}{\tilde{u}_{j,j}}, 0, 0, \dots, 0)^T e_j^T =$$

$$= \begin{pmatrix} 1 & 0 & \dots & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \iddots & \vdots \\ 0 & 0 & \dots & 1 & \dots & 0 \\ 0 & 0 & \dots & -\frac{\tilde{u}_{j+1,j}}{\tilde{u}_{j,j}} & \dots & 0 \\ \vdots & \vdots & \iddots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 1 \end{pmatrix}.$$

Az $m-r$ darab $\tilde{P}_j$ permutáció és $\tilde{M}^{(j)}$ Gauss transzformáció után az új U' mátrixot

visszaalakítottuk felső háromszög mátrixszá:

$$U'' = \tilde{M}^{(m-1)} \tilde{P}^{(m-1)} \tilde{M}^{(m-2)} \tilde{P}^{(m-2)} \ldots \tilde{M}^{(r)} \tilde{P}^{(r)} U'$$

Vagy a (2.44)-at alkalmazva kapjuk a következőt:

$$U'' = \tilde{M}^{(m-1)} \tilde{P}^{(m-1)} \tilde{M}^{(m-2)} \tilde{P}^{(m-2)} \ldots \tilde{M}^{(r)} \tilde{P}^{(r)} M^{(m-1)} P^{(m-1)} M^{(m-2)} P^{(m-2)} \ldots M^{(1)} P^{(1)} \tilde{B} P^{(R)}.$$

Jegyezzük meg, hogy minden egyes $\tilde{P}_j$ permutációs mátrix csak két sort változtat meg. Tehát minden Gauss transzformációs $\tilde{M}^{(j)}$ mátrixnak csak 1 nem nulla főátlón kívüli eleme van. Tehát ezek alapján az LU frissítése nagyon gyors. De ennek az eljárásnak is van hátránya. A fő probléma az, hogy az LU sora:

$$\tilde{M}^{(m-1)} \tilde{P}^{(m-1)} \tilde{M}^{(m-2)} \tilde{P}^{(m-2)} \ldots \tilde{M}^{(r)} \tilde{P}^{(r)} M^{(m-1)} P^{(m-1)} M^{(m-2)} P^{(m-2)} \ldots M^{(1)} P^{(1)}$$

a B bázis minden frissítésénél egyre hosszabb lesz. Nyílvánvaló, hogy minél nagyobb a probléma mérete, annál hosszabb lesz az előbbi LU sor. A pontosabb és hatékonyabb memória felhasználás érdekében az LU felbontást a kezdetektől fogva periódikusan újraszámoljuk.

Figyeljük meg a következő általános sémát az LU-felbontás frissítésére, amikor egy lineáris egyenletrendszert oldunk meg: $B\,x = b$.

A B bázis LU felbontása: $B = LU$, vagy $L^{-1}B = U$.

Tehát a B bázis módosítása után a következőt kapjuk: $L^{-1}\,\tilde{B} = \tilde{U}$, ahol az $\tilde{U}$ mátrixnak van egy úgynevezett hegye, ami az f_k vektort tartalmazza:

$$\tilde{U} = \begin{pmatrix} * & * & * & * & * & * & * \\ & * & * & * & * & * & * \\ & & * & * & * & * & * \\ & & * & * & * & * & * \\ & & * & & * & * & * \\ & & * & & & * & * \\ & & * & & & & * \end{pmatrix}$$

Miután jobbról alkalmazzuk rá a $P^{(R)}$ permutációs mátrixot, a következő U' mátrixot kapjuk:

$$L^{-1}\,\tilde{B}\,P^{(R)} = \tilde{U}\,P^{(R)} = \begin{pmatrix} * & * & * & * & * & * & * \\ & * & * & * & * & * & * \\ & & * & * & * & * & * \\ & & * & * & * & * & * \\ & & & * & * & * & * \\ & & & & * & * & * \\ & & & & & * & * \end{pmatrix} = U'.$$

Az U' mátrixot felső háromszög alakúvá konvertáljuk $\tilde{P}_j$ és $\tilde{M}^{(j)}$, j = r, r+1, …,m-1 segítségével. Bal oldalról szorozva az egyenlőséget :

$$L^{-1}\,\tilde{B}\,P^{(R)} \;=\; \tilde{U}\,P^{(R)}$$

a következővel:

$$\tilde{M}^{(m-1)}\tilde{P}^{(m-1)}\tilde{M}^{(m-2)}\tilde{P}^{(m-2)}\ldots\tilde{M}^{(r)}\tilde{P}^{(r)}$$

Ezután kapjuk:

$$Q\,L^{-1}\,\tilde{B}\,P^{(R)} \;=\; Q\,\tilde{U}\,P^{(R)}, \tag{2.47}$$

ahol a Q akövetkező szorzást jelenti:

$$Q \;=\; \tilde{M}^{(m-1)}\tilde{P}^{(m-1)}\tilde{M}^{(m-2)}\tilde{P}^{(m-2)}\ldots\tilde{M}^{(r)}\tilde{P}^{(r)} \tag{2.48}$$

A $Q\,\tilde{U}\,P^{(R)}$ a frissített U'' felső háromszög mátrix. Tehát a (2.47)-ot a következő módon is felírhatjuk:

$$Q\,L^{-1}\,\tilde{B}\,P^{(R)} \;=\; U'', \tag{2.49}$$

Végül a (2.49)-as egyenlőségből kapjuk az LU felbontás új $\tilde{B}$ bázisát:

$$\tilde{B} \;=\; L\,Q^{-1}\,U''\,(P^{(R)})^{-1}. \tag{2.50}$$

A (2.50)-as formulával már meg tudjuk oldani a $\tilde{B}\,x \;=\; b$ egyenletrendszert, a kövekező lépéseket alkalazva:

1. **lépés:** A (2.50)-as formula segítségével átfogalmazzuk a $\tilde{B}\,x \;=\; b$ egyenletrendszert:

$$L\,Q^{-1}\,U''\,(P^{(R)})^{-1}\,x \;=\; b, \tag{2.51}$$

2. **lépés:** Ezután bevezetünk egy új változót:

$$y \;=\; Q^{-1}\,U''\,(P^{(R)})^{-1}\,x$$

   Meg kell oldanunk az $L\,y = b$ egyenletrendszert, és ekkor kapjuk az $y = L^{-1}\,b$ vektort. Lehet, hogy ennek az egyenletrendszernek a megoldása már az előző lépésben meg lesz, amikor az eredeti $B\,x = b$ egyenletrendszert megoldottuk. Ekkor természetesen kihagyjuk ezt a lépést.

3. **lépés:** Q-val $y \;=\; Q^{-1}\,U''\,(P^{(R)})^{-1}\,x$ mindkét oldalát szorozzuk meg balról:

$$Qy \;=\; Q\,Q^{-1}\,U''\,(P^{(R)})^{-1}\,x$$

   azaz:

$$U''\,(P^{(R)})^{-1}\,x \;=\; Q\,y$$

   Ezután vezessük be a $z \;=\; (P^{(R)})^{-1}\,x$ új változót és helyettesítsük be a rendszerbe:

$$U'' z = Q y$$

4. **lépés:** Végül megkapjuk az eredményt:

$$x = P^{(R)} z.$$

Egy másik módja a (2.50)-es formula alkalmazásának a $\tilde{B} x = b$ egyenletrendszer megoldásához, a következő lépésekből áll:

1. **lépés:** Ahogy az előbbi első lépés esetében, itt is átírjuk a $\tilde{B} x = b$ egyenletrendszert (2.51)-nek megfelelő alakba.
2. **lépés:** Majd ennek mindkét oldalát beszorzom balról L^{-1}-gyel:

$$L^{-1} L Q^{-1} U''(P^{(R)})^{-1} x = L^{-1} b, \tag{2.52}$$

3. **lépés:** Ezután szorozzuk be balról Q-val (2.51)-es formulát:

$$Q Q^{-1} U''(P^{(R)})^{-1} x = Q L^{-1} b. \tag{2.53}$$

4. **lépés:** Számoljuk ki U'' mátrix inverzét, majd szorozzuk be vele balról a (2.53)-es formulát:

$$(U'')^{-1} U''(P^{(R)})^{-1} x = (U'')^{-1} Q L^{-1} b, \tag{2.54}$$

5. **lépés:** Végül szorozzuk be (2.53)-as formulát $P^{(R)}$-rel:

$$P^{(R)}(P^{(R)})^{-1} x = P^{(R)}(U'')^{-1} Q L^{-1} b,$$

és ebből megkapjuk az x-et:

$$x = P^{(R)}(U'')^{-1} Q L^{-1} b. \tag{2.55}$$

Ez a megoldás extra számításokat igényel, az L^{-1}, Q, $(U'')^{-1}$ és $P^{(R)}$ mátrixok inverzét is ki kell számolni (a Q és $P^{(R)}$ inverzét U'' kiszámolásánál meghatározzuk). Amikor vektort cserélünk a B bázisban, akkor minden szükséges frissítést végre kell hajtani a megfelelő LU felbontás U felső háromszög mátrixában, viszont az L alsó háromszög mátrixnak változatlannak kell maradnia. Ez azt jelenit, hogy az L inverzét (L^{-1}) csak egyszer kell kiszámolnunk, ezután változás nélkül használhatjuk, ahányszor csak szükségünk van rá. Ezek alapján, amikor ezt a módszert választjuk, akkor egyedül a 4. lépésnél kell plusz számolást végrehajtani, mégpedig az $(U'')^{-1}$ meghatározásánál. Mivel az U'' felső háromszög mátrix, ezért inverzének kiszámítása viszonylag olcsó művelet.

Az eljárás legnagyobb előnye, amikor a B bázisban báziscsere történik, akkor az LU felbontás frissítése az L mátrix változása nélkül megy végbe.

2.2.3.3. A Forest-Tomlin frissítés

Ahogy a Bartels-Golub módszernél, itt is a következő lépések történnek báziscsere esetén. A B bázis A_r oszlopvektorát egy új vektorra cseréljük, B bázisban az összes oszlop balra tolódik és jobbról bejön az új A_k oszlopvektor, és a (2.42)-nek megfelelő új bázisvektort kapunk, (2.45)-nek megfelelő U' felső háromszög mátrixszal, melynek főátlója mellett van egy részátló: $u_{r+1,r+1}$, $u_{r+2,r+2}$, ..., $u_{m,m}$. A Forest-Tomlin módszer szerint az L és az U mátrixok nem szingulárisok, és a részátlós elemek nem 0-ák az i. sorban ($i = r+1, r+2, ..., m$). Ezért az $r+1, r+2, ..., m$ sorokat használhatjuk az r. sor $r, r+1, r+2, ..., m-1$. oszlopelemének eliminálására, megadva a következő mátrixot:

$$U'' = \begin{pmatrix} U_{1,1} & \cdots & U_{1,r-1} & U_{1,r+1} & U_{1,r+2} & \cdots & U_{1,m} & f_1 \\ 0 & \cdots & U_{2,r-1} & U_{2,r+1} & U_{2,r+2} & \cdots & U_{2,m} & f_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & U_{r-1,r-1} & U_{r-1,r+1} & U_{r-1,r+2} & \cdots & U_{r-1,m} & f_{r-1} \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & f'_r \\ 0 & \cdots & 0 & U_{r+1,r+1} & U_{r+1,r+2} & \cdots & U_{r+1,m} & f_{r+1} \\ 0 & \cdots & 0 & 0 & U_{r+2,r+2} & \cdots & U_{r+2,m} & f_{r+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & U_{m,m} & f_m \end{pmatrix}.$$

Eggyel felfelé mozgatva az összes lenti $r+1, r+2, ..., m$ sort és az r. sort a végére téve, így megkapjuk a kívánt felső háromszög mátrixot:

$$U''' = \begin{pmatrix} U_{1,1} & \cdots & U_{1,r-1} & U_{1,r+1} & U_{1,r+2} & \cdots & U_{1,m} & f_1 \\ 0 & \cdots & U_{2,r-1} & U_{2,r+1} & U_{2,r+2} & \cdots & U_{2,m} & f_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & U_{r-1,r-1} & U_{r-1,r+1} & U_{r-1,r+2} & \cdots & U_{r-1,m} & f_{r-1} \\ 0 & \cdots & 0 & U_{r+1,r+1} & U_{r+1,r+2} & \cdots & U_{r+1,m} & f_{r+1} \\ 0 & \cdots & 0 & 0 & U_{r+2,r+2} & \cdots & U_{r+2,m} & f_{r+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & U_{m,m} & f_m \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & f'_r \end{pmatrix}.$$

Ez a permutáció inverze annak, amelyikben az $r+1, r+2, ..., m$ oszlopokat toltuk eggyel balra és az r. oszlopot vittük a jobb oldalra.

Használjuk a mátrix jelöléseket és jelöljük ezt a permutációt *Q*-val, ekkor azt kapjuk, hogy:

$$Q^{-1}\,\tilde{M}\,L^{-1}\,\tilde{B} \;=\; U'''. \tag{2.56}$$

ahol az *L* inverze:

$$L^{-1} \;=\; M^{(m-1)}\ldots M^{(2)}M^{(1)}$$

az a Gauss elimináció, amely az eredeti *B* mátrixot transzformálja *U* felső háromszög mátrixszá. Az $\tilde{M} \;=\; \tilde{M}_{m-1}\ldots\tilde{M}_{r+1}\tilde{M}_{r}$ eredmény mátrix az a transzformációs mátrix, amely eliminálja az *r, r+1, r+2, ..., m-1*. elemeket az *r*. sorban (ez eredményezi *U''*-t). Nyilvánvaló, hogy az M_i mátrixok transzformációs mátrixok, melyek minden *i = r, r+1, r+2, ..., m-1*. elemet lenulláznak az *r*. sorban:

$$\tilde{M}_i \;=\; I \;-\; e_r g_i^T,$$

ahol az e_r az *r*. oszlopvektora az m-edrendű egységmátrixnak, és:

$$g_i^T \;=\; (\underbrace{0,0,\ldots,0}_{i},\mu_i,0,0,\ldots,0)^T$$

és

$$\mu_i \;=\; \frac{u_{r,i+1}}{u_{i+1,i+1}}, \qquad i \;=\; r,r+1,\ldots,m-1.$$

Tehát a μ_i szorzó az *i+1*. Eleme az *i*. sornak, azaz:

$$\tilde{M}_i \;=\; \begin{pmatrix} 1 & \ldots & 0 & 0 & 0 & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \iddots & \vdots \\ 0 & \ldots & 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 1 & -\mu_i & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & 1 & 0 & \ldots & 0 \\ 0 & \ldots & 0 & 0 & 0 & 1 & \ldots & 0 \\ \vdots & \iddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & 0 & 0 & 0 & \ldots & 1 \end{pmatrix}, \qquad i \;=\; r,r+1,\ldots,m-1.$$

Végül *Q* a permutációs mátrix, és legyen $Q^{-1} = Q^T$. Írjuk át a (2.56)-et a következő alakban:

$$Q^T\tilde{M}_{m-1}\ldots\tilde{M}_{r+1}\tilde{M}_r M^{(m-1)}\ldots M^{(2)}M^{(1)}\tilde{B} \;=\; U'''. \tag{2.57}$$

Ez az algoritmus csak az *U* felső háromszög mátrixszal foglalkozik, az *L* alsó háromszög mátrixszot nem változtatja.

2.2.4. Egyéb faktorizáció típusok

A lineáris algebra, az LP és az LFP programozás legfontosabb feladata, hogy a számítások végrehajtását fejlesszük. Az *A* mátrix minden speciális jellemzőjét megpróbáljuk kihasználni, pl.:szimmetrikusság, ortogonalitás, stb. A következő algoritmust gyakran használják az LP feladatoknál.

2.2.4.1. Cholesky felbontás

Ha az adott $A=\|a_{ij}\|_{m\times n}$ mátrix szimmetrikus és pozitív definit, akkor az általános LU felbontás helyett egy speciálisabb és hatékonyabb háromszög felbontást alkalmazunk:

$$A = L\,L^T, \tag{2.58}$$

ahol az *L* jelöl egy alsó háromszög mátrixot és L^T egy felső háromszög mátrixot. Ezt hívjuk Cholesky felbontásnak.

A szimmetrikusság azt jelenti, hogy: $a_{ij} = a_{ji}$, minden $i = 1, \ldots, m$, és $j = 1, \ldots, m$, azaz:

$$A = A^T.$$

A mátrix pozitív definit, ha $v^TAv > 0$ minden $v \neq 0$ vektor esetén.

Az LL^T felbontást hasonlóan számoljuk ki, mint az LU felbontást az (2.24)-(2.25)-ös formulában:

$$l_{ii} = \begin{cases} \sqrt{a_{ii}}, & ha \quad i = 1; \\ \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}, & ha \quad i > 1; \end{cases} \tag{2.59}$$

$$l_{ji} = \frac{(a_{ij} - \sum_{k=1}^{i-1} l_{ik}\,l_{jk})}{l_{ii}}, \quad j = i+1, i+2, \ldots, m; \tag{2.60}$$

minden i = 1, …, m esetén. Mint ahogy a Crout algoritmus szerinti LU felbontásnál, itt is sikeresen tudjuk alkalmazni a (2.59)-as és (2.60)-es egyenlőségeket. Ha végrehajtjuk ezeket a műveleteket, láthatjuk, hogy a jobb oldali l_{ij}-k mindig meghatározódnak, mire szükség van rájuk.

A Cholesky algoritmus teljes műveletigénye kb. kétszer kevesebb, mint egy *A* mátrix LU

felbontása, melyben a szimmetriát figyelmen kívül hagyjuk. A másik előnye ennek a metódusnak az A mátrix szimmetriájából fakad, mert az alsó háromszög L mátrix (az átlóbeli elemeket kivéve) tárolható az A mátrix alsó háromszög részében. Az egyetlen plusz tárhelyet csak egy m hosszú vektor tárolása igényli, amit hozzáilleszt az L átlójához.

Általánosabb szimmetrikus mátrixok esetén sokkal jobb a következő:

$$A = LDL^T, \tag{2.61}$$

ahol a D egy diagonális mátrix, és L jelöl egy alsó háromszög mátrixot.

Figyeljük meg, hogy a (2.59)-as és (2.60)-as formulák csak azokra az a_{ij} komponensekre hivatkoznak, ahol $j \geq i$. Ha az A szimmetrikus, akkor van elég információ, hogy végrehajtsuk a felbontást. A (2.59)-as és (2.60)-as formulák egy hatékony módját adják annak, hogy ellenőrizzük egy mátrix szimmetrikusságát és pozitív definitségét.

2.2.4.2. QR felbontás

Egy másik hasznos mátrix faktorizáció a QR felbontás:

$$A = QR,$$

ahol R felső háromszög mátrix, Q pedig ortogonális mátrix, azaz $Q^TQ = I$ és $Q^T = Q^{-1}$. Mint ahogy az eddigi metódusok is, ez is használható az $Ax = b$ lineáris egyenletrendszer megoldására.

Helyettesítsük be a Q-t és R-t az egyenletrendszerbe a következő módon:

$$QRx = b.$$

Először is oldjuk meg a $Qy = b$ egyenletrendszert, ahol $y = Rx$, hogy az y ismeretlen értékét megkapjuk. Ezután oldjuk meg az $Rx = y$ egyenletrendszert, és megkapjuk x értékét.

Mivel $Q^T = Q^{-1}$, ezért:

$$Q\,y \;=\; b\,, \;\rightarrow\; Q^{-1}\,Q\,y \;=\; Q^{-1}\,b\,, \;\rightarrow\; y \;=\; Q^T\,b.$$

Ez a faktorizáció nem csak négyzetes mátrixokra való, de itt csak az $A = \|a_{ij}\|_{m\times n}$ alakú mátrixokkal foglalkozunk. Ez az algoritmus sokszor fontos szerepet játszik a sajátérték kiszámításában vagy a legkisebb négyzetek módszerének megoldásában. Mivel a QR kétszer annyi műveletet igényel, mint az LU felbontás, és másfélszer több memóriát is használ, ezért

kevésbé használják a négyzetes mátrixokra. Mindazonáltal számos oka is van annak, hogy miért használjunk mégis ortogonális metódust. Először is, a metódus rendszerint nem igényel főelem kiválasztást, és számításilag is nagyon stabil, ami nem mondható el a Gauss eliminációról. Egy másik előnye, hogy a QR felbontás megfelel egy A mátrix első rendű módosításának $O(m^2)$ műveleten. A QR frissítés implementációja könnyebb és egyszerűbb, mint az LU felbontásé.

Van két standard algoritmus a QR faktorizációra. Az egyik magábafoglalja a Householder transzformációt vagy más néven Householder tükrözést, a másik pedig a Givens forgatást veszi alapul. Ez egy másik metódus a QR faktorizáció alkalmazására. Ezt az algoritmust szokták Gram-Schmidt ortogonalizációnak vagy Gram-Schmidt eljárásnak hívni. Mivel a klasszikus Gram-Schmidt eljárás numerikusan nem stabil, ezért nem vesszük figyelembe. A Householder transzformáció egy kevesebb számítást igénylő algoritmushoz vezet, mint a Givens forgatás, ezért ezt nézzük a következőkben.

Definíció: Ha $v \neq 0$, akkor a:

$$H = I - 2\frac{v v^T}{v^T v}$$

mátrixot hívjuk Housholder mátrixnak vagy Housholder tükrözésnek, és a v vektort Housholder vektornak.

A Housholder mátrix szimmetrius és ortogonális. A megfelelő v vektort alkalmazva létrehozhatunk egy olyan Housholder mátrixot, amivel az adott A mátrixot balról megszorozva, el tudjuk tüntetni az összes elemet az A mátrix j. oszlopában a kiválasztott a_{ij} elem alatt.
Tehát legyen a kiválasztott elemünk a_{11}, ekkor a következő módon számoljuk ki a Q_1 Housholder mátrixot, ha az A mátrix a következő:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{pmatrix}$$

és:

$$Q_1 A = Q_1 \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mm} \end{pmatrix} = \begin{pmatrix} \tilde{a}_{11} & \tilde{a}_{12} & \dots & \tilde{a}_{1m} \\ 0 & \tilde{a}_{22} & \dots & \tilde{a}_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \tilde{a}_{m2} & \dots & \tilde{a}_{mm} \end{pmatrix} = A^1 .$$

Hasonlóan tudjuk létrehozni a Q_2 mátrixot, kinullázva az elemeket a $\tilde{a}_{22}$ alatt, és így tovább Q_{m-1} Householder mátrixig. Tehát

$$Q_{m-1} \dots Q_2 Q_1 A = R.$$

Kihazsnálva a $Q_1, \dots, Q_{m-1}$ mátrixok ortogonalitását, könnyen megállapíthatjuk, hogy

$$Q = (Q_{m-1} \dots Q_2 Q_1)^{-1} = Q_1^T Q_2^T \dots Q_{m-1}^T .$$

Mivel a Householder mátrixok szimmetrikusak, ezt felírhatjuk úgy, hogy:

$$Q = Q_1 Q_2 \dots Q_{m-1} .$$

A Q_1 mátrix kiszámításához, szükségünk van a következő vektorra:

$$v_1 = A_1 + |A_1| e_{1,}$$

ahol az $A_1 = (a_{11}, a_{21}, \dots, a_{m1})^T$ oszlopvektor az A mátrix első oszlopa, és $|A_1|$ jeleni az A_1 vektor hosszát, azaz:

$$|A_1| = \sqrt{a_{11}^2 + a_{21}^2 + \dots + a_{m1}^2},$$

és az e_1 az m-ed rendű egységmátrix első oszlopa. Így:

$$Q_1 = I - 2 \frac{v_1 v_1^T}{v_1^T v_1}$$

és

$$Q_1 A = A^1 .$$

Az eljárást hasonlóan folytatjuk az $\tilde{A}^1 = \|\tilde{a}_{ij}\|_{m-1 \times m-1}$ mátrixon, mely úgy állt elő, hogy A^1-ből töröltük az első sort és oszlopot, azaz:

$$\tilde{A}^1 = \begin{pmatrix} \tilde{a}_{22} & \tilde{a}_{23} & \dots & \tilde{a}_{2m} \\ \tilde{a}_{32} & \tilde{a}_{33} & \dots & \tilde{a}_{3m} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{a}_{m2} & \tilde{a}_{m3} & \dots & \tilde{a}_{mm} \end{pmatrix}.$$

Így most már a megfelelő Householder vektor csak (m-1) dimenziós, tehát kiegészítjük nullával: $v_2 = (0, \tilde{v}_2)^T$. Használjuk a $\tilde{v}_2$ vektort a következő Householder mátrixhoz:

$$\tilde{Q}_2 = I - 2\frac{\tilde{v}_2 \tilde{v}_2^T}{\tilde{v}_2^T \tilde{v}_2}.$$

Ezután használjuk ezt a mátrixot a következőhöz:

$$Q_2 = \begin{pmatrix} 1 & 0 \\ 0 & \tilde{Q}_2 \end{pmatrix}.$$

Majd (m-1) lépés után megkapjuk a kívánt $A = QR$ felbontást, ahol:

$$R = Q_{m-1} \ldots Q_2 Q_1 A \qquad \text{és} \qquad Q = Q_1 Q_2 \ldots Q_{m-1}.$$

3. Bartels-Golub frissítés megvalósítása

A szimplex módszert megvalósító Wingulf program néhány feladatnál a kerekítésből származó hibák miatt nem megfelelő eredményt ad. Ennek kiküszöbölését segíti az aktuális bázis LU felbontása, frissítése, és a szimplex módszer együtthatóinak megfelelő újraszámolása. A szükséges eljárásokat Delphi nyelven DLL fájlként valósítottam meg.

A következőkben röviden ismertetem, hogy mi is az a DLL fájl, és hogyan valósítottam meg az LU felbontást, és frissítését.

3.1 A DLL

A DLL-ek dinamikus csatolású könyvtárak (Dynamic Link Libraries). A DLL fájlok közvetlenül nem futtathatók, de tartalmaznak programkódot (függvényeket, eljárásokat), továbbá tartalmazhatnak erőforrásokat (képeket, hangokat, szövegtáblákat), akárcsak a programok. Amikor egy Delphi alkalmazást írunk, akkor egy programfájlt készítünk. Ez a programfájl felhasználhatja a DLL-ekben található függvényeket, eljárásokat, erőforrásokat.

DLL-ek felhasználhatósága és azok főbb előnyei:

- Több program használhatja ugynazt a DLL-t. Vagyis, ha készítünk egy DLL-t olyan függvényekkel, eljárásokkal, erőforrásokkal, melyeket gyakran használunk a programjainkban, akkor később bármelyik alkalmazásban felhasználhatjuk ezt.
- Ha több program fut egyszerre, melyek ugyanazt a DLL-t használják, akkor a DLL csak egyszer töltődik be a memóriába. Így tehát memóriát is takaríthatunk meg.
- A Delphi-ben elkészített DLL-eket más programozási nyelven készült alkalmazásokban is felhasználhatjuk.

Az egyes eljárások és függvények megírása után fontos, hogy azokat a függvényeket és eljárásokat, melyeket kívülről (a programokból, melyek a DLL-t használni fogják) is meg kívánunk hívni, felsoroljuk az ***exports*** záradék után. Ha más fejlesztői rendszerből is el szeretnénk érni a DLL exportált eljárásait és függvényeit, akkor a Delphi-ben alapértelmezett register paraméterátadási mód helyett a ***stdcall*** szabványos Win32 paraméterátadási módot kell használnunk.

Ahhoz, hogy a kész DLL könyvtár függvényeit és eljárásait fel tudjuk használni alkalmazásainkban, a lefordított DLL állományt másoljuk oda, ahová alkalmazásunkat mentjük.

A DLL-ben definiált alprogramokat elérhetővé tehetjük, ha importáljuk őket az ***external*** kulcsszóval.

```
function pelda(a,b: integer):integer; stdcall; external 'dll_pelda.dll';
```

3.2 Az LU felbontás frissítésének megvalósítása.

A diplomamunkám nagy részét tette ki a Bartels-Golub frissítés vizsgálata, és megértése. Az általam készített DLL állomány neve ***EnDll***.

Az *EnDll* a *Wingulf* nevű, operációkutatással kapcsolatos feladatokat megoldó programhoz készült, annak számításait hivatott pontosabbá tenni. Ahhoz, hogy a *Wingulf* számára felhasználható legyen a DLL, szükség volt néhány változtatásra. A DLL-ben csak olyan fájlokra hivatkozhatunk, melyeknek a mérete nem túl nagy. Ez okozta az első problémát, mivel az *EnDll*-nek hozzá kell férnie a *Wingulf* néhány változójához, melyek eredetileg az *UInit.pas* fájlban voltak. Ez a fájl viszont túl nagy méretű, ezért a szükséges változókat egy *alap.pas* nevű fájlban adtam meg, mely az *UInit.pas*-ból is elérhető.

Az *EnDll*-ben az *alap.pas* állományban található változók közül is csak az ***OriginalProblem*** nevű rekordot használom. Az *OriginalProblem* rekord tartalmazza a feladat eredeti oszlopvektorait, és ennek segítségével töltöm fel a bázisoszlopokat tartalmazó mátrixot.

3.2.1 Az EnDll állomány ismertetése

Az *EnDll* állományban három fontos eljárás található. Az ***LU_Init***, ***LU_Frissit*** és a ***SimTabColJav***. Ezeken kívül van még a sorcserét megvalósító ***ChangeRows*** eljárás. Az első három eljárást használhatja majd a *Wingulf*, csak ezeket adtam meg az export kulcsszó után. A *ChangeRows* eljárást csak a DLL-en belül használom.

A program során alkalmazott változók:

```
frissito=Record
        LU: array of array of RealType;
        BG: array of array of RealType;
        OBL:array of integer;
        MBL:array of integer;
        LUP:array of integer;
        End;
```

Ábra 1: A frissito rekord

Egy rekordba kerültek a fontos változók, melynek neve ***frissito***. Az ***LU*** egy $m \times m$ -es mátrix, elemeinek típusa ***RealType***, melyet a *Wingulf* használ, és az *alap.pas* állományban definiáltam. Az ***LU*** fogja tartalmazni a bázis mátrix felbontását. ***BG*** az *LU*-hoz hasonló mátrix, csak ez a Bartels-Golub frissítéskor szükséges sorcseréket és módosításokat tartalmazza, melyekkel elérjük, hogy az U újra felső háromszög mátrix legyen. Lényegében a *BG* mátrix a (2.48)-as szorzás eredményét tartalmazza:

$$BG = \tilde{M}^{(m-1)}\tilde{P}^{(m-1)}\tilde{M}^{(m-2)}\tilde{P}^{(m-2)}\ldots\tilde{M}^{(r)}\tilde{P}^{(r)}$$

Ez azért előnyös, mert az újabb és újabb frissítésekkel az LU felbontás sora egyre csak nőne.

$$\tilde{M}^{(m-1)}\tilde{P}^{(m-1)}\tilde{M}^{(m-2)}\tilde{P}^{(m-2)}\ldots\tilde{M}^{(r)}\tilde{P}^{(r)}M^{(m-1)}P^{(m-1)}M^{(m-2)}P^{(m-2)}\ldots M^{(1)}P^{(1)}$$

Így viszont ezt a hosszú szorzatot csak egy mátrixban elég tárolni, és nem igényel egyre több helyet. Később a frissítés folyamán ezt a szorzatot kell használnunk.

Az ***OBL*** és ***MBL*** két 1 dimenziós tömb, melyek a bázisoszlopok eredeti feladat szerinti indexeit tartalmazzák. Az *OBL* a bázisoszlopok eredeti sorrendjét tárolja. Ezt használjuk majd fel a szimplex tábla együtthatóinak az újraszámolásánál, hogy a megfelelő sorrendben adja vissza a *SimTabColJav* eljárás az együtthatókat. Az *MBL* tömb elemeinek a sorrendjét viszont mindig aszerint változtatom, ahogy az U felső háromszög mátrixban változik a bázisoszlopok

sorrendje. Tehát az *MBL* tömb segítségével találom meg, hogy az U mátrixban melyik oszlop, melyik eredeti indexű bázisoszlophoz tartozik, és innen tudom, hogy az U melyik oszlopát kell kicserélni az új bázisoszlopra. Lényegében a (2.42)-es képletben található $P^{(R)}$ permutációs mátrixok (melyekkel a bejövő A_k bázisoszlopokat az U mátrix utolsó oszlopába mozgatom) által okozott oszlopsorrend változásokat dokumentálja.

$$\tilde{B}\,P^{(R)} = (A_1, A_2, \ldots, A_{r-1}, A_{r+1}, A_{r+2}, \ldots, A_m, A_k),$$

Az rekord utolsó eleme az ***LUP***, mely *m-1* elemű 1 dimenziós tömb. Ez írja le az LU felbontás folyamán történt sorcseréket. Mivel az *i.* lépésben a sorcseréhez alkalmazható sorokat csak az *i.* sor alatti sorokból választom ki, ezért az utolsó sornál nem lehet sorcsere, ezért elég, hogy az *LUP m-1* elemű legyen. Az *LUP i.* elemének értéke *-1*, ha nem történt az *i.* lépésben sorcsere, azaz az *i.* sort nem kellett felcserélni egy másikkal, ellenkező esetben az *i.* elem értéke annak a sornak az indexe, amellyel az *i.* sort kicserélem.

Az előbbieken kívül még két globális változója van az *EnDll*-nek. A frissito rekord típusú ***valt***, amivel a frissito rekord elemeire hivatkozhatunk. A ***size***, amely az LU mérete lesz.

Az Eljárások:

ChangeRows

Külső fájlból nem elérhető. Az LU felbontás során szükséges sorcseréket valósítja meg. Szükséges megjegyezni, hogy nem teljes sorokat cserél fel, mivel az L és az U mátrixokat egy LU mátrixban tárolom, és sorcserét csak az U mátrixban kell alkalmazni. Paraméterül két egész számot kap, melyek a két felcserélendő sor indexét határozzák meg.

LU_Init

Külső fájlból elérhető eljárás. Paraméterei: ***m***:integer; ***bl***:array of integer. Az *m* egy egész szám, mely meghatározza a bázismátrix méretét. A *size* globális változó kapja értékül az *m*-et, hogy az *EnDll* összes eljárásában hozzáférhető legyen. A *bl* oszlopindexeket tartalmazó 1 dimenziós tömb. Az aktuális bázis oszlopainak indexét tartalmazza. Ennek segítségével töltöm fel az *LU* mátrixot a bázisoszlopokkal, melyekre az LU felbontást kiszámolom.

Az *LU_Init* eljárás elején történik az *LU*, *BG*, *LUP*, *OBL*, *MBL* elemek helyfoglalása, inicializálása. Az *LU*-t feltöltöm a *bl* paraméter által meghatározott oszlopvektorokkal, az *LUP* összes elemének *-1*-et adok értékül. Az *OBL*, és *MBL* tömbök értékei a *bl* tömb értékeit veszik fel, azaz kezdetben még megegyeznek. A *BG* mátrix az inicializáláskor megegyezik az egységmátrixszal.

Az LU felbontás algoritmusa részben megegyezik a Crout algoritmussal. Annyival bővebb, hogy ez az algoritmus megvalósítja a sorcserét, ha a főátlóban nulla szerepel. Amennyiben az *i.* sorban a főátlóbeli elem nulla, akkor az *i.* oszlopban az i-től nagyobb indexű elemek közül megkeresi a legnagyobb abszolút értékű elemet, és az *i.* sort ennek a sorára cseréli, de csak az U-ban.

```
IMax:=0;
for i:=1 to m-1 do
        begin
                if valt.LU[i,i]=0 then
                        begin
                                Big:=0.0;
                                for j:=i+1 to m do
//legnagyobb abszolút értékű elem megkeresése az adott oszlopban
                                        if Abs(valt.LU[j,i])>Big then
                                                begin
                                                        Big:= Abs(valt.LU[j,i]);
                                                        IMax:=j;
                                                end;
                                        if Big<>0.0 then
                                                begin
                                                        ChangeRows(i,IMax);
                                                        valt.LUP[i]:=IMax;
                                                end;
                        end;

                If valt.LU[i,i]=0.0 Then
                        valt.LU[i,i]:=Tiny;

                for j:= i+1 to m do
                        begin
                                if valt.LU[i,i]<>0 then
                                        valt.LU[j,i]:=valt.LU[j,i]/valt.LU[i,i];
                                for k:=i+1 to m do
                                        valt.LU[j,k]:=valt.LU[j,k]-(valt.LU[j,i]*valt.LU[i,k]);
                        end;
        end;
```

Ábra 2: A módosított Crout

LU_Frissit

Külső fájlból elérhető eljárás. Paraméterei: ***mode***:integer;var ***be***:array of integer;var ***ki***:array of integer; ***n***:integer. A *mode* paraméter segítségével lehet megadni, hogy melyik módszerrel

történjen az LU felbontás frissítése. Ha 1, akkor Bartels-Golub, ha 2, akkor Forest-Tomlin frissítéssel. Ez majd csak a további fejlesztéseknél lesz érdekes, mivel jelenleg a Bartels-Golub frissítés valósult meg.

A *Wingulf* programban be lehet állítani, hogy hány iterációnként legyen az LU felbontás frissítése. Ennek megfelelően, amikor a *Wingulf* meghívja az *EnDll* állomány *LU_frissit* eljárását, akkor meg kell adnia az eljárásnak, hogy az iterációk során, mely oszlopok kerültek ki a bázisból, és melyek kerültek be a bázisba. Tehát a *Wingulf*nak dokumentálnia kell a bázis változásait. Az *n* egész szám adja meg, hogy hány iterációnként van frissítés. A ***be*** 1 dimenziós tömb tartalmazza az *n* iteráció alatt a bázisba bejövő oszlopok indexét, a ***ki*** 1 dimenziós tömb pedig a kimenő oszlopok indexét. Tehát az utolsó LU felbontás utáni *i.* iterációnál a bejövő oszlop indexét határozza meg a ***be*** tömb *i.* eleme, és az *i*. itercáiónál kimenő oszlop indexét adja meg a ***ki*** tömb *i.* eleme.

Az eljárás első lépéseként másolatot készítek az U felső háromszög mátrixról, mivel együtt tárolom az L alsó háromszög mátrixszal az *LU* változóban. A másolat segítségével érem el, hogy az U mátrix változásai ne módosítsák az L mátrixot.

Az *LU_frissit* eljárás folyamán az LU frissítése n-szer történik meg. A frissítés ciklusa a következő lépésekből áll:

1. *A bejövő oszlopvektor megfelelő átalakítása:* az új oszlopon ugyanazokat a lépéseket kell végrehajtani, mint amiket az eredeti bázison hajtottunk végre, hogy LU alakra hozzuk. Ehhez az L alsó háromszög mátrixot használjuk fel. Az L mátrix főátlóbeli elemeit kivéve, minden elemének vegyük a *-1*-szeresét, és *m-1*-szer szorozzuk meg balról az új oszlopot úgy, hogy például az *i.* szorzásnál a módisított L mátrixnak csak az *i.* oszlopában lévő elemeket vegyük figyelembe, a többi elem legyen nulla, illetve a főátlóbeli elemek 1-ek. A szorzás közben figyelni kell, hogy az *LUP* tömb *i.* eleme milyen érték, amennyiben nem *-1,* akkor az *i.* lépésben végre kell hajtani a megfelelő sorcserét is. Ezzel pont azokat az elemi sorátalakításokat hajtjuk végre az új oszlopon, melyeket a bázison kell végrehajtani, hogy megkapjuk az U felső háromszög mátrixot. Ehhez a lépéshez kihasználtam, hogy az LU felbontás nagyon közel áll a Gauss-eliminációhoz, csak kevesebb a műveletigénye.

$$f = M^{(m-1)} P^{(m-1)} M^{(m-2)} P^{(m-2)} \dots M^{(1)} P^{(1)} A_k.$$

ahol az $M^{(i)}$ alsó háromszög mátrixok olyanok, hogy a főátlóban csupa 1-es szerepel, és ezen kívül csak egy, az *i.* oszlopukban van nullától különböző elem. Pont emiatt tárolhatók ezek az *M* mátrixok egy mátrixban, és ha a főátló alatti elemeit -1-gyel megszorozzuk, pont az L mátrixot kapjuk. Ez az oka annak, hogy az L mátrix megfelelő oszlopait figyelembe véve, és -1-gyel való szorzás helyes alkalmazásával a bejövő, új bázisoszlopot az előbbi képlet alapján át tudom alakítani, az *M* mátrixok helyett az L-et használva.

```
for k:=1 to size-1 do  // az f uj oszlopvektor szorzasa L-lel
        begin
         if valt.LUP[k]<>-1 then
                 Begin
                 seged:=f[k];
                 f[k]:=f[valt.LUP[k]];
                 f[valt.LUP[k]]:=seged;
                 End;
         for l:=1 to size do
                 begin
                 count[l]:=0;
                 for j:= 1 to size do
                         begin
                         if l=j then
                                 count[l]:=count[l] + f[l];
                         if l>j then
                                 begin
                                 if j=k then
                                         count[l]:=(-1*valt.LU[l,j])*f[j];
                                 end;
                         end;
                 end;
         for l:=1 to size do
                 begin
                         f[l]:= count[l];
                 end;
         end;
```

Ábra 3: Az új oszlop átalakítása LU-nak megfelelően

2. *BG mátrixszal való szorzás*: A megfelelően átalakított új oszlopot balról be kell szorozni az előző ciklusokban létrejövő *BG* mátrixszal, ami az eddigi Bartels-Golub frissítés sorcseréit és U mátrix megfelelő módosításait tartalmazza. Mivel a *BG* mátrixban sok nulla elem lehet, ezért a szorzások száma csökkenthető egy ellenőrzéssel.

3. *Az OBL és MBL bázisleírások aktualizálása:* Az *OBL* tömbben meg kell keresni a ***ki*** tömbben megadott kimenő oszlop indexét, és kicserélni a ***be*** tömbben megadott megfelelő bejövő oszlop indexére. Az *MBL* tömbben pedig meg kell keresni a ***ki*** tömbben megadott kimenő oszlop indexét, és *PR* lokális változó értéke felveszi ennek a kimenő oszlop indexének az *MBL* tömbbeli pozícióját .
4. *Az új, módosított bázisoszlop beszúrása a PR által jelölt helyre:* Mivel az új oszlopot úgyis az U mátrix utolsó oszlopába kell forgatni, az U oszlopait a *PR*. oszlop után, eggyel előrébb toljuk, majd az új oszlopot az U utolsó oszlopába szúrjuk. Amennyiben a *PR* értéke megegyezik az U mátrix méretével (azaz *m* vagy *size*), akkor nincs forgatás. Ebben a lépésben aktualizáljuk az *MBL* tömböt a forgatásnak megfelelően.

```
if PR<>size then
        Begin
        for j:=PR+1 to size do
                for k:=1 to size do
                        Begin
                        U[k,j-1]:=U[k,j];
                        valt.MBL[j-1]:=valt.MBL[j];
                        End;
        End;
for j:=1 to size do
        Begin
        U[j,size]:=Count[j];
        valt.MBL[size]:=be[i];
        End;
```

Ábra 4: Az új bázisoszlop beszúrása az U mátrixba

5. *Az U mátrix átalakítása és a Bartels-Golub frissítése:* Az új bázisoszloppal frissített U mátrixban a *PR*. oszloptól kezdve a főátló alatt van egy plusz sor. Ezt a sort kell eltüntetni, hogy az U újra felső háromszög mátrix legyen. Először megvizsgálom, hogy a *j*. sor főátlóbeli eleme nem nulla-e, ha igen, akkor a *j*. sort fel kell cserélni a *j+1*.-el, mivel az U mátrix adott oszlopában a főátlóbeli *j*. elem alatt csak a *j+1*. elem nem nulla. A *BG* mátrixot is frissíteni kell a sorcserének megfelelően. Majd a következő Gauss transzformációs lépéseknek megfelelően nullázzuk le a főátló alatti elemeket:

$$\tilde{M}^{(j)} = I - (\underbrace{0,0,\ldots,0}_{j}, \frac{\tilde{u}_{j+1,j}}{\tilde{u}_{j,j}}, 0, 0, \ldots, 0)^{T} e_j^T =$$

$$= \begin{pmatrix} 1 & 0 & \dots & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \iddots & \vdots \\ 0 & 0 & \dots & 1 & \dots & 0 \\ 0 & 0 & \dots & -\frac{\tilde{u}_{j+1,j}}{\tilde{u}_{j,j}} & \dots & 0 \\ \vdots & \vdots & \iddots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 1 \end{pmatrix}.$$

Ez lényegében azt jelenti, hogy az U mátrixot balról szorozzuk egy olyan mátrixszal, amelynek a főátlóbeli 1 értékű elemektől eltekintve, egy nullától különböző eleme van. Az ilyen mátrixszal való szorzás az U mátrixnak csak egy sorát változtatja meg, a fenti képlet jelöléseit figyelembe véve, ez a megváltozó sor a *j+1*. sor. Természetesen a *BG* mátrixot is meg kell szorozni balról ezzel a mátrixszal, az U felső háromszög mátrixhoz hasonlóan.

```
for j:=1 to size-1 do
        Begin
        if U[j,j]=0 then    //sorcsere
                Begin
                //seged:=0.0;
                for k:=j to size do
                        Begin
                        seged:=U[j,k];
                        U[j,k]:=U[j+1,k];
                        U[j+1,k]:=seged;
                        //seged:=0.0;
                        seged:=valt.BG[j,k];
                        valt.BG[j,k]:=valt.BG[j+1,k];
                        valt.BG[j+1,k]:=seged;
                        End;
                End;
        if U[j+1,j]<>0 then    //atlo alatti elem eltuntetese
                Begin
               // seged:=0.0;
                seged:=-1*U[j+1,j]/U[j,j];
                for k:=1 to size do
                        Begin
                        U[j+1,k]:=seged*U[j,k]+U[j+1,k];
                        valt.BG[j+1,k]:=seged*valt.BG[j,k]+valt.BG[j+1,k];
                        End;
                End;
        End;
```

Ábra 5: Az U mátrix főátló alatti elemeinek lenullázása

Miután a frissítés ciklusa végére ért, a frissített U felső háromszög mátrixot visszamásoljuk a frissito rekord *LU* mátrixába.

SimTabColJav

Külső fájlból elérhető eljárás. Paraméterei: var ***tomb***:array of RealType; ***k***:Integer. A *tomb* egy üres 1 dimenziós tömb, melyben majd a szimplex tábla adott oszlopának az újraszámolt együtthatóit kell visszaadni. A *k* azt az oszlopot adja meg, melynek együtthatóit újra kell számolni. Ennek az eljárásnak 4 fő lépése van:

1. Az $Ly = A_k$ egyenletrendszerből számoljuk ki az *y* értékét.
2. Az *y* vektort balról szorozzuk be a *BG* mátrixszal. Erre azért van szükség, mivel az 1. lépsben alkalmazott L mátrix az eredeti LU felbontáshoz tartozik, viszont az *y* megfelelő értékéhez szükségünk van arra, hogy az U minden változásával is frisítsük az *y*-t. Amennyiben még nem változott az U mátrix értéke, akkor a *BG* mátrix egységmátrix, tehát az *y* nem változik. Mivel a *BG* mátrixban sok nulla elem lehet, ezért a szorzások száma csökkenthető egy ellenőrzéssel.
3. Az $Ux = y$ egyenletrendszerből számoljuk ki az *x* értékét. A programban a *seged* változó tartalmazza a 2. lépés utáni *y* vektort.

```
seged[size]:=seged[size]/valt.LU[size,size];
for i:=size-1 downto 1 do
        Begin
        for j:=size downto i+1 do
                seged[i]:=seged[i]-seged[j]*valt.LU[i,j];
        seged[i]:=seged[i]/valt.LU[i,i];
        End;
```

Ábra 6: Az Ux = y kiszámítása

4. Mivel a frissítéskor sok oszlopcsere történt, a *tomb*ben nem megfelelő sorrendben vannak az együtthatók. Így a *tomb* elemeit az *OBL* és *MBL* segítségével megfelelő sorrendbe rendezem, és így adom vissza.

Összefoglalás

A diplomamunkám készítése közben sikerült elmélyülnöm a bázismátrix frissítésével kapcsolatos módszerekben, különös tekintettel a Bartels-Golub metódusra. Feladatom volt ennek a módszernek a megvalósítása, a Wingulf pontosságának, hatékonyságának növelése céljából. A sok kutakodás közben, az LU felbontás elkészítésére leltem más algoritmusokat is, melyek megvalósították a sorcserét, ha a főátlóbeli elem nulla. Mégis a több módszer közül, az ismertetett algoritmust választottam, amely a Crout algoritmuson alapszik, ugyanis ez a feladat szempontjából sokkal kezelhetőbb felbontást eredményezett.

Úgy vélem, hogy a programban még számos lehetőség rejlik, amivel csökkenteni lehet az elvégzett műveletek számát. Ez abból adódik, hogy a Bartels-Golub frissítés közben sokszor számolunk ritkamátrixokkal.

Az LU_frissit eljárás paraméterében szerepel a *mode* változó, melynek jelenleg még nincs funkciója, de a további fejlesztés reményében szerepel az eljárásban. Ez biztosít lehetőséget majd választani a módszerek közül, hogy melyikkel szeretnénk az LU felbontás frissítését elvégezni. Több olyan módszer is van, mellyel tovább lehet fejleszteni ezt a DLL-t, de ezek közül is a legfontosabb a Forest-Tomlin módszer.

Visszatekintve azt kell mondanom, hogy hasznos ismeretekkel és tapasztalatokkal gazdagodtam, miközben megtanultam egy általam eddig nem alkalmazott programozási nyelvet, a Delphit.

Irodalomjegyzék

- Bajalinov,E.B.: ”Linear-Fractional Programming: Theory, Methods, Applications and Software”,
  Kluwer Academic Publishers, 2003.

- Bajalinov Erik – Imreh Balázs: Operációkutatás
  Polygon, Szeged, 2001

- Stoyan Gisbert, Takó Galina, ”Numerikus módszerek I.”, Typotex 2005

- http://www.cise.ufl.edu/~davis/Morgan/index.htm