Debreceni Egyetem
Informatika Kar

A megbízhatóság és stabilitás növelése az operációkutatási szoftverekben

Témavezető:
Dr. Bajalinov Erik

Készítette:
Tóth János
PTM

Debrecen
2007

Tartalomjegyzék

Bevezetés....3
1.A módszerek elméleti háttere....5
1.1A bázismátrix faktorizációja....5
1.1.1Az LU felbontás....6
1.1.2Numerikus példa az LU felbontásra....11
1.1.3Az LU felbontás és a Gauss elimináció kapcsolata....12
1.2Az LU faktorizáció frissítése....17
1.2.1Bartels-Golub frissítés....17
1.2.2Numerikus példa a Bartels-Golub frissítésre....21
1.2.3A Bartels-Golub frissítés megvalósítása....22
1.2.4Forrest-Tomlin frissítés....25
2. A példaprogram bemutatása....28
2.1Az MPS formátum....29
2.2A program használata....33
2.2.1Új feladat létrehozása....33
2.2.2Feladat betöltése....34
2.2.3Bázis kiválasztása....34
2.2.4Beállítások....34
2.2.5A feladat megoldása....35
2.2.6A bázis frissítése....35
2.2.7A Szimplex táblázat kiszámolása....35
3. A függvénykönyvtár bemutatása....37
3.1A felhasznált adatszerkezetek....37
3.1.1A linsys struktúra....37
3.1.2A linmx struktúra....38
3.2A függvénykönyvtár használata....40
3.2.1Főbb funkciók....40
3.3Segédfüggvények....42
3.3.1A linsys struktúrához....42
3.3.2A linmx struktúrához....43
Összefoglalás....45
Irodalomjegyzék....46

Bevezetés

A valós életben előforduló operációkutatási feladatok, ezen belül is a lineáris és hiperbolikus programozási feladatok akár több százezer változót és megszorítást is tartalmazhatnak, amikben az egyik feltételben millió Forintok szerepelnek a másikban tizedszázalékok. Ez a gyakorlatban gyakran előfordul. Ez nagyban megnehezíti a megoldás pontos meghatározását, mivel a számítógépek véges pontossággal dolgoznak, a kerekítési hibák elkerülhetetlenek.

Az ezen problémák megoldására alkalmazott Szimplex módszer egy iteratív módszer, így rendkívül fontos, hogy az egyes iterációkban minimális legyen a hiba mértéke, hiszen ezek továbbadódnak a következő iterációkba, míg végül a megoldás már teljesen használhatatlan lesz. Ahogy nő a feladat mérete, annál fontosabbak lesznek a stabilitást növelő módszerek és eljárások.

A kereskedelmi forgalomban kapható solverek számos ilyen módszert tartalmaznak, ezek közül szeretnék áttekinteni néhányat ebben a munkában.

Az első a bázismátrix faktorizációja, ami egy direkt módszer arra, hogy újraszámoljuk a Szimplex tábla elemeit. Erre általában az LU felbontást használják. Így a Szimplex táblázatot néhány iterációnként újraszámolják közvetlenül az eredeti feladatból, ami csökkenti a továbbvitt hibát.

Az LU felbontás rendkívül számításigényes feladat, viszont egy iteráció során csak egy oszlop cserélődik a bázisban. Így nagyszámú változó esetén pazarlás az egész LU felbontást előről kezdeni, hiszen a bázis nagy része nem változik. Ezért fontosak az LU felbontás frissítését végző algoritmusok, amivel ez a számítási igény jelentősen csökkenthető.

Dolgozatom középpontjában az LU felbontás frissítése áll majd, ezen belül két módszert vizsgálok meg, a Bartels-Golub és a Forrest-Tomlin frissítést.

Másodlagos célja a dolgozatomnak, hogy a fenti módszerek elméleti áttekintése mellett meg is valósítsam azokat ANSI C nyelven és létrehozzak egy újrahasznosítható függvénykönyvtárat ezekhez, valamint egy példaprogram írása Borland C++ Builder 5 környezetben, ami a fenti függvénykönyvtárra épül, és bemutatja annak használatát.

Így a diplomamunka második részében ezt a példaprogramot szeretném bemutatni. A Linsys alkalmazás lineáris egyenletrendszerek megoldását végzi el, hiszen Szimplex módszer során nagyon hasonló problémákat oldunk meg, ezekre ugyanúgy alkalmazhatóak a tárgyalt módszerek.

A harmadik részében pedig a módszereket megvalósító könyvtárhoz szeretnék egy kisebb referenciát írni, valamint magyarázatot fűzni a működéséhez.

1. A módszerek elméleti háttere

Ebben a fejezetben a bázismátrix faktorizációjának és a faktorizáció frissítésének az elméleti hátterét szeretném áttekinteni, valamint numerikus példákkal illusztrálni azok működését.

1.1 A bázismátrix faktorizációja

Nagyszámú változó és megszorítás esetén - amiknek száma a gyakorlatban előforduló problémák esetén akár több százezer is lehet - elkerülhetetlen, hogy a Szimplex módszerben az egyes iterációkban kis hibák jelenjenek meg, mivel a számítógép pontossága véges és a probléma megoldása akár több millió lebegőpontos számítást is igényelhet.

Konkrétabban a Szimplex tábla együtthatóinak meghatározásakor jelennek meg ezek, az új bázisba való áttérés során:

$$x'_{ij} = \begin{cases} x_{ij} - \dfrac{x_{rj}\, x_{ik}}{x_{rk}}, & i = 1,2,\ldots,m \quad i \neq r, \\ & j \in J_N = J \setminus J_B \\ \dfrac{x_{rj}}{x_{rk}}, & i = r, \end{cases}$$

, ahol $J = \{1,2,\ldots,n\}$ egy indexhalmaz, $J_B = \{s_1, s_2, \ldots, s_m\}$ pedig a bázisvektorok indexeit tartalmazó halmaz, *r* a jelenlegi bázist elhagyó vektor indexe, *k* a bázisba bekerülő vektor indexe.

A Szimplex módszer végrehajtása közben ez a formula teremt kapcsolatot a régi x_{ij} és az új x'_{ij} együtthatók között. Az iterációk során az előző számításokat felhasználva térünk át az új bázisra. Ezért szükséges időnként újraszámolni az x_{ij} együtthatókat közvetlenül az eredeti feladatból, valamilyen direkt módszer segítségével. Tehát az eredeti *A* mátrix és a jelenlegi bázis $B = (A_{s_1}, A_{s_2}, \ldots, A_{s_m})$ -ből. Ez a következő lineáris egyenletrendszerek megoldását jelenti:

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

Ez *n – m* egyenletrendszert jelent, ha a változók száma *n* és a megszorítások száma *m* . Ha magas *n / m* arány, akkor ez rendkívül számításigényes. Azonban vegyük észre, hogy csak az

egyenletrendszerek jobb oldala változik, $A\underline{x}=\underline{b}$ típusú egyenletrendszerekről van szó, ahol csak a *b* vektor különbözik. Ezek megoldására nem lenne célszerű a Gauss eliminációt vagy a Gauss-Jordan módszert használni, mivel ezekhez szükség van a *b* vektor ismeretére. Számításigényük magas, először az *A* mátrixot felső háromszög alakra kell hoznunk, majd előrehelyettesítéssel vagy visszahelyettesítéssel meghatároznunk a megoldást. Ez $m^3/2 + m^2$ lebegőpontos számítást igényel minden egyes egyenletrendszer megoldásakor. A következő részben tárgyalt módszer ezekre a problémákra nyújt megoldást, hatékonyan használható olyan egyenletrendszerek megoldására, amik csak a *b* vektorban különböznek.

1.1.1 Az LU felbontás

Ebben a részben az alábbi formában megadott egyenletrendszerek megoldásáról lesz szó :

$$Ax = b,$$

ahol *A* egy $m \times m$ -es invertálható négyzetes mátrix, *b* egy *m* elemű tetszőleges vektor $b_1, b_2, \ldots, b_m$ elemekkel.

Az egyenletrendszer megoldásának látszólag legegyszerűbb módja, ha kiszámoljuk az A^{-1} mátrixot (ez *A* inverzét jelenti) és az egyenlet mindkét oldalát megszorozzuk vele, tehát $x = A^{-1}b$. De ez a legritkább esetben célravezető, az inverz számolása nagyon számításigényes.

A legelterjedtebb direkt módszerek az együtthatómátrix faktorizációját használják fel a megoldás meghatározásához. Az egyik legismertebb ilyen eljárás az LU felbontás vagy LU faktorizáció, ami az eredeti *A* mátrixot egy alsó trianguláris *L* és egy felső trianguláris *U* mátrix szorzataként fejezi ki. Tehát

$$P_S A P_O = LU$$

,ahol

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

és P_S és P_O egy-egy permutációs mátrix, ami az *A* mátrix sorait és oszlopait cseréli fel.

Ezt a faktorizációt aztán két lépésben használhatjuk fel az egyenletrendszer megoldására. Először

$$Ly = P_S b \quad ,$$

aztán

$$Uz = y$$

Végül x -et úgy kapjuk meg, hogy z -t megszorozzuk P_O -val :

$$x = P_O z$$

Az eredeti lineáris egyenletrendszer helyett így két megoldandó egyenletrendszert kapunk, azonban ezek speciálisak. Alsó és felső háromszög alakúak az együtthatómátrixok, amiket egy előre- és egy visszahelyettesítéssel megoldhatunk. Ez $O(n^2)$ idő alatt valósítható meg. Tehát ha ismerjük az A mátrix LU felbontását, akkor tetszőleges b vektor esetén meg tudjuk oldani az egyenletrendszert ennyi idő alatt.

Formálisan az $Ax = b$ egyenletrendszer megoldása LU felbontással a következő:

1. Határozzuk meg az L és U mátrixokat ($A=LU$) !
2. Az $LUx=b$ egyenletrendszerben helyettesítsük Ux-et y-nal és oldjuk meg az $Ly=b$ rendszert előrehelyettesítéssel, így megkapjuk y-t.
3. Oldjuk meg az $Ux=y$ rendszert visszahelyettesítéssel, így megkapjuk x-et.

Azonban a felbontás csak olyan mátrixokra alkalmazható, amik nem tartalmaznak 0 elemet a főátlójukban, ezért szerepelnek $P_S A P_O = LU$ -ban a sorokat és oszlopokat cserélő permutációs mátrixok. Például

$$A = \begin{pmatrix} 0 & 2 \\ 2 & 4 \end{pmatrix} = LU = \begin{pmatrix} 1 & 0 \\ l_{21} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{pmatrix}$$

Ez alapján $u_{11} = 0$ és $l_{21}u_{11} = 4$, ami nem lehetséges. De ha a következőt tekintjük,

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

ennek már létezik LU felbontása és B-t az A mátrixból kaptuk sorcserével. Ha A mátrix

nem szinguláris, akkor mindig van a soroknak és oszlopoknak olyan permutációja, hogy a főátlóba nem 0 elemek álljanak.

Tehát ha egy A négyzetes mátrix nem szinguláris, mindig léteznek P_S, P_O permutációs mátrixok és L alsó- és U felső trianguláris mátrixok, hogy

$$P_S A P_O = LU$$

Valójában P_S és P_O közül az egyik is elég.

De hogyan határozzuk meg az L és U mátrixokat? Amit keresünk:

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

A fenti mátrixszorzást kibontva az alábbi egyenleteket kapjuk:

$$\begin{aligned} (i<j)\quad & a_{ij} = l_{i1}u_{1,j} + l_{i2}u_{2j} + \ldots + l_{ii}u_{ij} \\ (i=j)\quad & a_{ij} = l_{i1}u_{1,j} + l_{i2}u_{2j} + \ldots + l_{ii}u_{jj} \\ (i>j)\quad & a_{ij} = l_{i1}u_{1,j} + l_{i2}u_{2j} + \ldots + l_{ij}u_{jj} \end{aligned}$$

Adott m^2 darab egyenlet és $m^2 + m$ darab ismeretlen. Ez azt jelenti, hogy az LU felbontás nem egyértelmű, m darab változónak tetszőleges értéket adhatunk. Crout módszere a fenti egyenletrendszer megoldására a következő :

1. Legyen $l_{ii} = 1$ $i = 1,2,...,$ m -re.
2. Minden $j = 1, 2, ..., m$ -re végezzük el az alábbi két lépést
   - $i = 1, 2, ..., j$ -re $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}$
   - $i = j + 1, ..., m$ -re $l_{ij} = (a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}) / u_{ij}$

Az algoritmus végrehajtása során az u_{ij} és l_{ij} értékek meghatározásához csak a már kiszámolt elemeket kell felhasználnunk, így az algoritmus egyértelműen működik. Az a_{ij} értékeket csak egyszer kell felhasználnunk, ezért azok helyén tárulhatjuk a megfelelő u_{ij} és l_{ij} elemeket, hiszen tudjuk , hogy $u_{ij} = 0$, ha $(i > j)$ és $l_{ij} = 0$, ha $(i<j)$, valamint az L mátrix

főátlójában 1-esek állnak. Ez azt jelenti, hogy nincs szükség extra memória lefoglalására, a számítógép memóriáját hatékonyan tudjuk kihasználni.

A módszer végrehajtása során egy helyen kerül sor osztásra. Meg kell vizsgálnunk, hogy mi történik abban az esetben , ha $u_{jj} = 0$. Ez történt a fentebbi példában is. Ez azt jelenti, hogy az eredeti *A* mátrixnak nem létezik LU felbontása, azonban tudjuk, hogy van olyan az *A* mátrix sorainak permutációjából alkotott mátrix, aminek van LU dekompozíciója. Hogyan kezeljük ezt az esetet az algoritmusban?

Tegyük fel, hogy a mátrix feldolgozása során *a j.* oszlopban járunk, és azt látjuk, hogy $u_{jj} = 0$. Ekkor az eljárás nem folytatható, hiszen az l_{ij}-ket u_{jj}-vel kell osztani. Folytassuk tovább az l_{ij} -k számolását, de az osztást hagyjuk el! Megtehetjük azt, hogy az osztást elhalasztjuk és csak akkor végezzük el, ha már megvan az összes l_{ij} az adott oszlopban, hiszen ezek nem függnek egymástól. Legyen a *j*-edik oszlopban abszolút értékben legnagyobb l_{ij} sorindexe *k.* Cseréljük ki a *k*-adik és *j*-edik sort a mátrixban! Ha az abszolút értékben legnagyobb elem 0, akkor ez azt jelenti, hogy a mátrix szinguláris. Milyen mátrixot kaptunk így?

Ez nem más, mint amit úgy kapunk, hogy az eredeti *A* mátrix *j*-edik és *k*-adik sorát megcseréljük és alkalmazzuk rá ugyanazt az eljárást, mint az eredetire. A *k*-adik és *j*-edik sor csak L megfelelő elemeit (a helyben elvégzett módszer miatt) és még feldolgozatlan elemeket tartalmaz, hiszen a mátrix a *j*-edik oszlopig van feldolgozva és $k > j$, tehát az U mátrix nem változik. Az l_{ij} elemek pedig csak a sorban őket megelőző l_{ij} elemektől függenek, ezeket pedig velük együtt átmásoltuk.

Ekkor a főátlóban már az abszolút értékben legnagyobb elem áll, elvégezhetjük az l_{ij} elemek osztását. Ez a részleges főelem-kiválasztás. Ezzel a módszerrel mindig elvégezhető az LU felbontás abban az esetben, ha az *A* mátrix nem szinguláris. A sorok cseréjét természetesen fel kell jegyezni, hiszen ezek szorzata adja meg a P_S mátrixot. A főelem-kiválasztást akkor is elvégezhetjük, ha a főátlóban nem 0 áll, ezzel is növelve az algoritmus stabilitását.

Összefoglalva tehát Crout módszere részleges főelem-kiválasztással kibővítve minden invertálható $A = \|a_{ij}\|_{m\times m}$ mátrixra kicseréli azt a *PA* mátrix LU felbontásával, ahol *P* egy permutációs mátrix, ami a sorcseréket hajtja végre, *A*-ban nem tárolja az *L* mátrix főátlójában

szereplő 1-eseket.

A Crout módszerével végrehajtott LU dekompozíció nagyjából m^3 / 3 lebegőpontos szorzást és összeadást végez, míg az $Ly = b$ és $Ux=y$ rendszerek megoldására használt előre- és visszahelyettesítések műveletigény m^2 egyenként. Abban az esetben, ha több $Ax=b$ egyenletrendszert oldunk meg, amik csak a b vektorban különböznek, nagyságrendekkel gyorsabban kapunk eredményt, mint a Gauss elimináció segítségével, hiszen a faktorizációt csak egyszer kell elvégeznünk, ezután tetszőleges b vektorra $O(m^2)$ időben kapjuk meg a megoldást.

A Crout algoritmus részleges főelem-kiválasztással C kódban:

```
/*m az A mátrix, n a sorok és oszlopok száma,
perm vektor rögzíti a    sorcseréket, eredetileg 1..n -t
          tartalmazza,
eps a legkisebb pozitív lebegőpontos szám*/

int ls_lucrout(double m[], int n,int perm[],double eps)
{
       int i,j,k,swaps=1,imax;
       double max,tmpmax;

       if(m==NULL) return 0;


/* Minden oszlopra*/
   for(j=0;j<n;++j){
            int i=0;
            while(i<=j){
                     for(k=0;k<i;++k)
                              m[i*n+j]-=m[i*n+k]*m[k*n+j];
                     ++i;
            }
            while(i<n){
                     for(k=0;k<j;++k)
                              m[i*n+j]-=m[i*n+k]*m[k*n+j];
                     ++i;
            }
            /* főelem kiválasztása*/
            max=fabs(m[j*n+j]);
            imax=j;

            for(k=j+1;k<n;++k){
                     tmpmax=fabs(m[k*n+j]);
                     if(tmpmax>max){
```

```
                                max=tmpmax;
                                imax=k;
                        }
                }
                if(max<eps){/*a legnagyobb elem 0*/
                    return 0;/*a mátrix szinguláris*/
                }

                /*Sorcsere*/
                if(imax!=j){
                        int temp=perm[imax];
                        swaps*=-1;
                        perm[imax]=perm[j];
                        perm[j]=temp;
                        for(k=0;k<n;++k){
                                double temp=m[imax*n+k];
                                m[imax*n+k]=m[j*n+k];
                                m[j*n+k]=temp;
                        }

                }
                /*Osztás a főelemmel*/
                for(k=j+1;k<n;++k)
                 m[k*n+j]/=m[j*n+j];
        }
        return swaps;
}
/* visszatéréskor az m tömb tartalmazza a PA mátrix LU
felbontását, perm[i] pedig azt, hogy az i. sor hányadik sornak
felel meg az eredeti mátrixban. A determináns számolásához
szükség lehet a sorcserék számára, így a függvény  1-gyel tér
vissza, ha páros számú sorcsere történt, -1-gyel ha páratlan*/
```

1.1.2 Numerikus példa az LU felbontásra

Legyen $A = \begin{pmatrix} 1 & 1 & 1 \\ 3 & 1 & 2 \\ 4 & 2 & 1 \end{pmatrix}$ ekkor az algoritmus szerint:

- $j = 1$ -re
  - $u_{11} = a_{11} = 1$
  - $l_{21} = a_{21} / u_{11} = 3$
  - $l_{31} = a_{31} / u_{11} = 4$
- $j = 2$-re

- $u_{12} = a_{12} = 1$
- $u_{22} = a_{22} - l_{21}u_{12} = -2$
- $l_{32} = (a_{32} - l_{31}u_{12}) / u_{22} = 1$

- j = 3-ra
  - $u_{13} = a_{13} = 1$
  - $u_{23} = a_{23} - l_{21}u_{13} = -1$
  - $u_{33} = a_{33} - l_{31}u_{13} - l_{32}u_{23} = -2$

Tehát $A = \begin{pmatrix} 1 & 1 & 1 \\ 3 & 1 & 2 \\ 4 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 4 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & -2 & -1 \\ 0 & 0 & -2 \end{pmatrix} = LU$

1.1.3 Az LU felbontás és a Gauss elimináció kapcsolata

Analitikusan egy mátrix LU felbontása szoros kapcsolatban áll annak Gauss eliminációjával, ebben a részben ezt a kapcsolatot szeretném megvizsgálni.

Adott egy $A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mm} \end{pmatrix}$ mátrix. Ennek Gauss eliminációja során a b vektorral kibővített A mátrixot felső háromszög alakúra hozzuk. Ezt úgy érjük el, hogy az első oszloptól kiindulva kinullázzuk a főátló alatti elemeket elemi mátrixtranszformációkkal. Mátrixszorzással felírva ez nem más, mint $A^{(2)} = M^{(1)} A$, ahol

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

, ahol $\mu_{i1} = \frac{a_{i1}}{a_{11}} \quad i = 2,3,\ldots,m$. A következő lépésben kinullázzuk a második oszlopot. Ezt az $M^{(2)}$ mátrixszal való szorzással érjük el. Ekkor $A^{(3)} = M^{(2)} M^{(1)} A$ az A aktuális

állapota. Ha az a_{11} vagy a_{22} elem 0, akkor sorcserére van szükség, tehát valójában $A^{(2)}=M^{(1)}P^{(1)}A$ és $A^{(3)}=M^{(2)}P^{(2)}M^{(1)}P^{(1)}A$, ahol $P^{(1)}$ egy permutációs mátrix ami A sorait, $P^{(2)}$ pedig $M^{(1)}P^{(1)}A$ egy-egy sorát cseréli meg.

Az eljárást folytatva a következőket kapjuk:

$$A^{(k+1)}=M^{(k)}P^{(k)}\dots M^{(1)}P^{(1)}A$$

Ezt addig folytatjuk, míg a mátrixot felső háromszög alakúra nem alakítottuk, ez az m. lépésben következik be:

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

Ekkor $A^{(m)}$ felső trianguláris:

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

, ahol $a_{1\mathrm{j}}^{(1)}=a_{1\mathrm{j}}\quad j=1,2\dots,m$.

Rendezzük át a fenti összefüggést oly módon, hogy az egyenlet mindkét oldalát megszorozzuk sorban $(M^{(m-1)})^{-1},(P^{(m-1)})^{-1},\dots,(M^{(1)})^{-1},(P^{(1)})^{-1}$ -gyel. Ekkor azt kapjuk, hogy

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

, hiszen $M^{(m-1)}(M^{(m-1)})^{-1}=I$, $(P^{(m-1)})^{-1}P^{(m-1)}=I$ és hasonlóan a többire is.

Most vizsgáljuk meg a

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

mátrixot. Szavakkal leírva ez azt teszi, hogy a *j. s*orból kivonja a *k-1.* sor $\mu_{j,k}$ szeresét *j=k,k+1...,m* . Ebből logikusan adódik, hogy ennek az inverz művelete az lesz, ami a *j. s*orhoz hozzáadja a *k-1.* sor $\mu_{j,k}$ szeresét *j=k,k+1...,m* -re. Tehát

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

Könnyen ellenőrizhető, hogy $M^{(k)}(M^{(k)})^{-1}$ valóban az egységmátrixot adja. Ezek alapján nagyon könnyű az $M^{(k)}$ inverzének meghatározása, hiszen csak a főátló alatti elemek előjelét kell megváltoztatnunk. $P^{(k)}$ inverze önmaga, hiszen egyetlen sorcserét végrehajtó permutációs mátrixról van szó.

A fentiek ismeretében és mivel tudjuk, hogy alsó trianguláris mátrixok szorzata is alsó trianguláris, úgyhogy

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

Tehát legyen

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

$$\text{ekkor} \quad A=\tilde{L}\,U$$

Sajnos nem lehetünk benne biztosak, hogy $\tilde{L}$ alsó trianguláris a végrehajtott sorcserék miatt, de az biztos, hogy az $\tilde{L}$ mátrix egy alsó trianguláris mátrix sorainak permutációjából áll elő. Megtehetjük, hogy a permutációs mátrixokat kiemeljük, hiszen a Gauss elimináció végrehajtása során jól látszik, hogy mindegy az, hogy előbb végezzük el a kinullázást és aztán cseréljük meg a főelem alatti sorokat, vagy fordítva, viszont ha felcseréljük a műveleteket, akkor a soroknak megfelelő tényezőket is meg kell cserélnünk az összes ezt megelőző $\tilde{M}$ mátrixban is. Tehát felírható

$\tilde{L}=P^{(1)}\ldots P^{(m-2)}P^{(m-1)}M'^{(1)}..M'^{(m-1)}$, ahol $M'^{(i)}$ az $(M^{(i)})^{-1}$, aminek az i. oszlopvektorára alkalmazzuk az összes őt követő sorcserét, tehát szorozzuk balról

$P^{(m-1)} \dots P^{(i+1)}$ -gyel, így ha $P^{-1} = P^{(1)} \dots P^{(m-2)} P^{(m-1)}$ és $L' = M'^{(1)} \dots M'^{(m-1)}$

, akkor ez alapján $A = P^{-1} L' U$, tehát $PA = L' U$. Ha „megjegyezzük” $P^{(k)}$ sorcseréket, akkor meg tudjuk határozni azt az $\tilde{A}$ mátrixot, amit A sorainak felcserélésével kapunk, és LU a felbontása a fentiek szerint.

$$\tilde{A} = PA = L' U$$

Hasonlóan a Crout módszerhez részleges főelem-kiválasztással, megkapjuk a *P* permutációs, *L′* alsó és *U* felső trianguláris mátrixokat tetszőleges nem szinguláris *A* négyzetes mátrix esetén.

Tekintsünk egy rövid példát a módszer működésére! Oldjuk meg az *Ax=b* lineáris egyenletrendszert, ahol

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} \quad \text{és b} = (1, 2, 2)^{\mathrm{T}}$$

Az első lépés:

$$A^{(2)} = M^{(1)} A = \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & -1 \\ 0 & 1 & 2 \end{pmatrix}$$

A második lépés:

$$A^{(3)} = M^{(2)} M^{(1)} A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & -1 \\ 0 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & -1 \\ 0 & 0 & 1 \end{pmatrix} = U$$

A mátrix felső háromszög alakban van, meghatározhatjuk *L*-t !

$$L = (M^{(1)})^{-1} (M^{(2)})^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix}$$

Ellenőrzésképpen:

$$LU = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & -1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix} = A$$

Először előrehelyettesítéssel megoldjuk $Ly = b$ -t

$$\begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}$$

Ebből adódik, hogy $y_1 = 1 \quad y_2=0 \quad y_3=1$. Ezután az Ux=y -t oldjuk meg visszahelyettesítéssel.

$$\begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Ezt megoldva azt kapjuk, hogy $x_1 = 1$ $x_2=-1$ $x_3=1$. Az egyszerűség kedvéért a feladatban nem szerepeltek sorcserék. Azonban érdemes megjegyezni, hogy a számítás pontosságának növelése érdekében megtehetjük azt, hogy a Gauss elimináció egyes lépéseiben olyan sorcseréket végzünk, hogy az abszolút értékben legnagyobb elem kerüljön a főátlóba.

1.2 Az LU faktorizáció frissítése

Az előző részben megvizsgáltuk, hogyan lehet egy $A = \|a_{ij}\|_{m\times m}$ nem szinguláris mátrix LU felbontását előállítani, aminek segítségével meg tudunk oldani egy $Ax = b$ alakú lineáris egyenletrendszert.

Ez a művelet nagyjából $O(m^3)$ idő alatt valósítható meg, ami elég magas. A szimplex módszer végrehajtása során, viszont egy-egy iteráció között a bázisnak csak egy oszlopa változik. Ha lehetséges lenne egy előző bázis LU felbontását felhasználni az aktuális bázis LU felbontásához, azzal rengeteg számítást meg lehetne spórolni.

Számos módszert dolgoztak ki, amely ilyen update -et végez el, ezek közül a Bartels-Golub és Forrest-Tomlin frissítést tekintem át.

1.2.1 Bartels-Golub frissítés

Ez a módszer a LU felbontás oszlopvektorainak átrendezését és struktúrájának sajátosságait használja fel.

Jelölje $B = (A_1, A_2, ..., A_m)$ azt az eredeti bázist, amelyre már rendelkezésünkre áll az LU faktorizáció! $A_j = (a_{1j}, a_{2j}, \ldots, a_{mj})^T \quad j = 1,2,\ldots,m$. Legyen $\tilde{B}$ az új bázis. B és $\tilde{B}$ egymástól csak egy oszlopvektorban különböznek! Legyen

$B = (A_1, A_2, \ldots, A_{r-1}, A_r, A_{r+1}, \ldots, A_m)$, A_r hagyja el a bázist, valamint

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

Tudjuk B LU felbontását. Az $M^{(i)}$ Gauss mátrixokkal és $P^{(i)}$ sorokat cserélő permutációs mátrixokkal felírva:

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

, ahol $P^{(i)}$ -k olyan permutációs mátrixok, amik biztosítják, hogy a Gauss transzformáció során a legnagyobb abszolút értékű elem kerüljön a főátlóba, U a Gauss elimináció eredményeként létrejött felső trianguláris mátrix.

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

Az egyszerűbb jelölés végett nevezzük át az elemeit:

$$U=\begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1\mathrm{m}} \\ 0 & u_{22} & \cdots & u_{2\mathrm{m}} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{mm} \end{pmatrix}=(U_{1,}U_{2,}\ldots,U_{m})$$

Rendezzük át az új bázis oszlopvektorait úgy, hogy az új oszlopvektor kerüljön jobb szélre és a tőle eredetileg jobbra lévőket eggyel balra csúsztatjuk. Ezt a permutációt jelölje $P^{O.}$ Ekkor $\tilde{B}\,P^{O}=(A_{1,}A_{2,}\ldots,A_{r-1},A_{r+1},\ldots,A_{m},A_{k})$.

Jelölje *f* a következő szorzatot!

$$f=M^{(m-1)}P^{(m-1)}\ldots M^{(1)}P^{(1)}A_{k}$$

Szorozzuk meg az átrendezett új bázist balról $M^{(m-1)}P^{(m-1)}\ldots M^{(1)}P^{(1)}$ -vel. Ez nem lesz más, mint

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

ahol $U'=(U_{1,}U_{2,}\ldots,U_{r-1},U\,r+1,\ldots,U_{m},f)$. Részletesebben felírva U'-t :

$$U'=\begin{pmatrix} u_{11} & \cdots & u_{1,r-1} & u_{1,r+1} & u_{1,r+2} & \cdots & u_{1\mathrm{m}} & f1 \\ 0 & \cdots & u_{2,r-1} & u_{2,r+1} & u_{2,r+2} & \cdots & u_{2\mathrm{m}} & f2 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & u_{r-1,r-1} & u_{r-1,r+1} & u_{r-1,r+2} & \cdots & u_{r-1,m} & f_{r-1} \\ 0 & \cdots & 0 & u_{r,r+1} & u_{r,r+2} & \cdots & u_{r,m} & f_{r} \\ 0 & \cdots & 0 & u_{r+1,r+1} & u_{r+1,r+2} & \cdots & u_{r+1,m} & f_{r+1} \\ 0 & \cdots & 0 & 0 & u_{r+2,r+2} & \cdots & u_{r+1,m} & f_{r+1} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & u_{mm} & f_{m} \end{pmatrix}$$

U' speciális szerkezetű, majdnem felső háromszög alakú, csak az $u_{r+1,r+1},u_{r+2,r+2},\ldots u_{mm}$ vannak a főátló alatt, ezért ezt a mátrixot könnyedén újra felső

triangulárissá alakíthatjuk a megfelelő Gauss transzformációkkal. Nullázzuk ki a főátló alatti elemeket!

Végezzük el a következő lépéseket $j = r,r+1, ..,m-1$ -re:

1. Cseréljük meg a *j.* és *j+1.* sort a $\tilde{P}^{(j)}$ permutációs mátrixszal úgy hogy $\tilde{u}_{j,j} \geq \tilde{u}_{j+1,j}$ legyen. Ha már eredetileg is $u_{j,j} \geq u_{j+1,j}$, akkor $\tilde{P}^{(j)}$ nem más mint az egységmátrix, így elhagyható.

2. Végezzük el az $\tilde{M}^{(j)}$ Gauss transzformációt, amivel kinullázzuk a *j.* oszlopban a főátló alatti elemet.

$$\tilde{M}^{(j)} = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & \cdots & 0 \\ 0 & \cdots & \frac{-\tilde{u}_{j+1,j}}{\tilde{u}_{j,j}} & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 1 \end{pmatrix}$$

m – r iteráció után megkapjuk az $\hat{U}$ frissített mátrixot, valamint az $\tilde{M}^{(j)}$ Gauss mátrixokat és a $\tilde{P}^{(j)}$ permutációs mátrixokat.

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

A $\tilde{P}^{(j)}$ permutációs mátrixok csak egy-egy sort cserélnek meg, a $\tilde{M}^{(j)}$ Gauss mátrixok pedig csak egy elemet tartalmaznak a főátló alatt, így az *U* mátrix frissítése nagyon gyorsan elvégezhető. Azonban a módszernek meg van az a hátránya, hogy a mátrixok listája minden frissítéssel egyre hosszabb lesz, így egy idő után jobban megéri az eredeti mátrix alapján újra elvégezni az LU felbontást.

Hogyan használhatjuk a frissített felbontást egy $\tilde{B}x = b$ egyenletrendszer megoldására? Ismerjük az $B=LU$ mátrix felbontását. Ezt felírhatjuk $L^{-1}B = U$ -ként is. Ha *B*-ben megváltoztatjuk az egyik oszlopot, akkor $L^{-1}\tilde{B} = \tilde{U}$ -t kapunk. $\tilde{U}$ Szerkezete speciális, tartalmaz egy úgynevezett „tüskét”, ami az előzőekben megadott *f* vektor.

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

Erre alkalmazva a P^O permutációt jobbról, megkapjuk az U' mátrixot, ami a főátló alatti elemeket tartalmazza. $L^{-1}\tilde{B}P^O = \tilde{U}P^O = U'$

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

Ezek után U' -t felső triangulárissá alakítjuk úgy, hogy szorozzuk $\tilde{M}^{(j)}$ és $\tilde{P^{(j)}}$ mátrixokkal, j = r, r+1, ..., m-1. Legyen $Q = \tilde{M}^{(m-1)}\tilde{P}^{(m-1)}\ldots\tilde{M}^{(r)}\tilde{P}^{(r)}$.

Ekkor $QL^{-1}\tilde{B}P^O = \hat{U}$ felső trianguláris. Fejezzük ki ebből az összefüggésből $\tilde{B}$ -t.

$$\tilde{B} = LQ^{-1}\hat{U}(P^O)^{-1}$$

Ezt helyettesítsük vissza a megoldandó egyenletrendszerbe! Ekkor azt kapjuk , hogy

$$LQ^{-1}\hat{U}(P^O)^{-1}x = b$$

Ezt az egyenletrendszert az alábbi lépésekben oldhatjuk meg:

- Vezessünk be egy új változót, legyen $y = Q^{-1}\hat{U}(P^O)^{-1}x$!
- Oldjuk meg az *Ly=b* rendszert előrehelyettesítéssel! Ezt már valószínűleg megoldottuk az előző *Bx=b* rendszer során.
- Szorozzuk meg mindkét oldalt *Q*-val, így $Qy = \hat{U}(P^O)^{-1}x$ -t kapunk.
- Vezessünk be még egy új változót, legyen $z = (P^O)^{-1}x$
- Oldjuk meg a $\hat{U}z = Qy$ rendszert előrehelyettesítéssel !
- Végül megkapjuk az egyenlet megoldását: $x = P^O z$

1.2.2 Numerikus példa a Bartels-Golub frissítésre

Tekintsük az 1.1.3 részben megadott példát, tehát

$$B=(A_1, A_2, A_3)=LU=\begin{pmatrix}1&0&0\\2&1&0\\1&-1&1\end{pmatrix}\begin{pmatrix}1&1&1\\0&-1&-1\\0&0&1\end{pmatrix}=\begin{pmatrix}1&1&1\\2&1&1\\1&2&3\end{pmatrix}$$ és b = (1, 2, 2)^T

Cseréljük ki az első oszlopot az A_4 = (5, 1, 3)^T vektorral! Ekkor a Bartels-Golub frissítés lépései:

1. Az első oszlopvektort vigyük a mátrix végére, a többit toljuk eggyel balra. Ezt a következő permutációs mátrix valósítja meg.

$$P^O=\begin{pmatrix}0&0&1\\1&0&0\\0&1&0\end{pmatrix}$$. Ezek után $$\tilde{B}P^O=\begin{pmatrix}1&1&5\\1&1&1\\2&3&3\end{pmatrix}$$

2. Számoljuk ki az *f* vektort!

$$f=L^{-1}A_4=M_2M_1A_4=\begin{pmatrix}1&0&0\\0&1&0\\0&1&1\end{pmatrix}\begin{pmatrix}1&0&0\\-2&1&0\\-1&0&1\end{pmatrix}\begin{pmatrix}5\\1\\3\end{pmatrix}=\begin{pmatrix}5\\-9\\-11\end{pmatrix}$$

3. Ekkor

$$L^{-1}\tilde{B}P^O=U'=(U_2, U_3, f)=\begin{pmatrix}1&1&5\\-1&-1&-9\\0&1&-11\end{pmatrix}$$

4. Most határozzuk meg azokat a Gauss mátrixokat amikkel felső háromszög alakra hozhatjuk Az *U'*-t . Először:

$$\tilde{M}^{(1)}U'=\begin{pmatrix}1&0&0\\1&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&1&5\\-1&-1&-9\\0&1&-11\end{pmatrix}=\begin{pmatrix}1&1&5\\0&0&-4\\0&1&-11\end{pmatrix}$$

5. A következő lépésben sorcserére van szükség, hiszen a főátlóba 0 került:

$$\tilde{P}^{(2)}\tilde{M}^{(1)}U'=\begin{pmatrix}1&0&0\\0&0&1\\0&1&0\end{pmatrix}\begin{pmatrix}1&0&0\\1&1&0\\0&0&1\end{pmatrix}\begin{pmatrix}1&1&5\\-1&-1&-9\\0&1&-11\end{pmatrix}=\begin{pmatrix}1&1&5\\0&1&-11\\0&0&-4\end{pmatrix}$$

6. További átalakításra nincs szükség, a mátrix felső háromszög alakú. Tehát

$\hat{U} = \tilde{P}^{(2)} \tilde{M}^{(1)} U'$. $\tilde{P}^{(2)} \tilde{M}^{(1)} = Q$ -t meg kell jegyeznünk, erre a megoldás során szükség lesz.

7. A megoldandó egyenletrendszer tehát $L^{-1} Q^{-1} \hat{U} (P^O)^{-1} x = b$. Az algoritmus lépései szerint:

1. $Ly = b = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}$ ebből adódik, hogy y = (1, 0, 1)^T

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

3. $\hat{U} z = Qy$,tehát

$\begin{pmatrix} 1 & 1 & 5 \\ 0 & 1 & -11 \\ 0 & 0 & -4 \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$, ebből adódik, hogy z = (4,-7/4,-1/4)^T

4. Végül ezt megszorozva a permutációs mátrixszal megkapjuk a végeredményt.

$$x = P^O z = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 4 \\ -7/4 \\ -1/4 \end{pmatrix} = \begin{pmatrix} -1/4 \\ 4 \\ -7/4 \end{pmatrix}$$

1.2.3 A Bartels-Golub frissítés megvalósítása

A fenti módszer megvalósítása ANSI C nyelven a következőképp néz ki:

```
/*   ls a frissítendő lineáris egyenletrendszer
     entering  a belépő bázis indexe
     leaving a bázist elhagyó vektor indexe*/


int ls_updatelu(linsys ls,int entering, int leaving){
    int i,j,k;
    double* newcol;
    double val;

     /*i -be kerül, hogy a kilépő vektor hányadik a bázisban
       (r)*/
    for(i=0;i<ls->extmx->rows;++i){
        if(ls->bases[i]==leaving) break;
    }
```

```
    /*a frissítést helyben végezzük*/
    /*Az utolsó oszlop kiszámolása (f): Q*L^-1*entering*/
    newcol=linmx_getcol(ls->extmx,entering);
    ls_applypermut(ls,newcol);
     /*L^-1*/
    ls_forwardsub(ls,newcol);

    /*az eddigi frissítések alkalmazása*/
    for(j=0;j<ls->num_eta;++j){
            if(ls->Q[j].swap==_TRUE){
                double temp=newcol[ls->Q[j].col];
                newcol[ls->Q[j].col]=newcol[ls->Q[j].col+1];
                newcol[ls->Q[j].col+1]=temp;
            }
            newcol[ls->Q[j].col+1]+=
               newcol[ls->Q[j].col]*ls->Q[j].val;
    }

    /*a nem változó részek másaolása*/
    for(j=0;j<i;++j){
        for(k=i;k<ls->extmx->rows-1;++k){
             ls->lu[j*ls->extmx->rows+k]=
                    ls->lu[j*ls->extmx->rows+k+1];
        }
    }
     /*minden további oszlopra */
    for(;i<ls->extmx->rows-1;++i){
        /*bázis index másolása*/
        ls->bases[i]=ls->bases[i+1];
        ls->Q[ls->num_eta].col=i;

        if((fabs(ls->lu[i*ls->extmx->rows+i+1])<
            ls->extmx->eps)&&
           (fabs(ls->lu[(i+1)*ls->extmx->rows+i+1])<
            ls->extmx->eps)){
        /* az új bázis szinguláris*/
            ls->luvalid=_FALSE;
            return 0;
        }

        /*sorcsere, ha u_{j+1,j} nagyobb*/
        if(fabs(ls->lu[i*ls->extmx->rows+i+1])<fabs(ls-
>lu[(i+1)*ls->extmx->rows+i+1])){
            /*csere*/
            int size=(ls->extmx->rows-i-1)*sizeof(double);
            double* first=&ls->lu[i*ls->extmx->rows+i+1];
            double* second=&ls->lu[(i+1)*ls->extmx->rows+i+1];
            double *temp=(double *)malloc(size);
            double tmp2;
```

```
        if(temp==NULL){
           ls->luvalid=_FALSE;
           return 0;
        }
        memcpy(temp,first,size);
        memcpy(first,second,size);
        memcpy(second,temp,size);
        free(temp);
        /*f -ben is megcseréljük a sorokat*/
        tmp2=newcol[i];
        newcol[i]=newcol[i+1];
        newcol[i+1]=tmp2;
        ls->Q[ls->num_eta].swap=_TRUE;
    }else { ls->Q[ls->num_eta].swap=_FALSE;}

    /*mü meghatározása*/
    if(fabs(ls->lu[(i+1)*ls->extmx->rows+i+1])<
       ls->extmx->eps) val=0;
    else val=-ls->lu[(i+1)*ls->extmx->rows+i+1]/
             ls->lu[i*ls->extmx->rows+i+1];

    ls->Q[ls->num_eta].val=val;

      /*frissítés*/
    for(k=i+1;k<ls->extmx->rows;++k){
        ls->lu[i*ls->extmx->rows+k-1]=
           ls->lu[i*ls->extmx->rows+k];
        ls->lu[(i+1)*ls->extmx->rows+k]=
           ls->lu[(i+1)*ls->extmx->rows+k]+
           val*ls->lu[i*ls->extmx->rows+k];
    }
    newcol[i+1]+=val*newcol[i];
    ++ls->num_eta;
}

/*utolsó oszlop bemásolása*/
ls->bases[i]=entering;
if(fabs(newcol[i])<ls->extmx->eps){
    /*az új bázis szinguláris*/
    ls->luvalid=_FALSE;
    return 0;
}
for(i=0;i<ls->extmx->rows;++i){
    ls->lu[(i+1)*ls->extmx->rows-1]=newcol[i];
}
free(newcol);

++ls->num_updates;
ls->luvalid=_TRUE;

return 1;}
```

1.2.4 Forrest-Tomlin frissítés

Ebben a részben a Forrest és Tomlin által kidolgozott frissítési módszer főbb elgondolásait szeretném áttekinteni.

Épp úgy ahogy a Bartels-Golub frissítés esetén, B a jelenlegi bázis A_r a bázisból kikerülő oszlopvektor. Toljuk el az A_r -től jobbra található oszlopokat eggyel balra, majd az utolsó oszlopba illesszük be a belépő új oszlopvektort A_k-t. Így megkapjuk $\tilde{B}$ -t az új bázist, aminek dekompozíciója a majdnem felső trianguláris U', ami az $u_{r+1,r+1},...,u_{m,m}$ főátló alatti elemekkel rendelkezik. Forrest és Tomlin észrevette, hogy ha L és U nem szinguláris, akkor ezek a főátló alatti elemek nem nullák. Így kinullázhatjuk vele az r. sor r,r+1, ..., m-1edik elemeit. Így a következő alakúra hozzuk:

$$U'' = \begin{pmatrix} u_{11} & \cdots & u_{1,r-1} & u_{1,r+1} & u_{1,r+2} & \cdots & u_{1,m} & f_1 \\ 0 & \cdots & u_{2,r-1} & u_{2,r+1} & u_{2,r+2} & \cdots & u_{2,m} & f_2 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & u_{r-1,r-1} & u_{r-1,r+1} & u_{r-1,r+2} & \cdots & u_{r-1,m} & f_{r-1} \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & f'_r \\ 0 & \cdots & 0 & u_{r+1,r+1} & u_{r+1,r+2} & \cdots & u_{r+1,m} & f_{r+1} \\ 0 & \cdots & 0 & 0 & u_{r+2,r+2} & \cdots & u_{r+1,m} & f_{r+2} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & u_{m,m} & f_m \end{pmatrix}$$

Ha ebben a mátrixban az alsó sorokat eggyel feltoljuk *(i = r+1,r+2,...,m-1*), az r-ediket pedig a mátrix aljára visszük, akkor a kívánt felső trianguláris mátrixot kapjuk.

$$\hat{U} = \begin{pmatrix} u_{11} & \cdots & u_{1,r-1} & u_{1,r+1} & u_{1,r+2} & \cdots & u_{1,m} & f_1 \\ 0 & \cdots & u_{2,r-1} & u_{2,r+1} & u_{2,r+2} & \cdots & u_{2,m} & f_2 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & u_{r-1,r-1} & u_{r-1,r+1} & u_{r-1,r+2} & \cdots & u_{r-1,m} & f_{r-1} \\ 0 & \cdots & 0 & u_{r+1,r+1} & u_{r+1,r+2} & \cdots & u_{r+1,m} & f_{r+1} \\ 0 & \cdots & 0 & 0 & u_{r+2,r+2} & \cdots & u_{r+2,m} & f_{r+2} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & \cdots & u_{m,m} & f_m \\ 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & f'_r \end{pmatrix}$$

A permutációs mátrix, amivel a sorokat cseréltük, pontosan az inverze annak, amivel az oszlopokat rendeztük át (legyen ez P^O). Mivel permutációs mátrixok,ezért $(P^O)^{-1}=(P^O)^T$.

Legyen $\tilde{M} = \tilde{M}^{(m-1)} \ldots \tilde{M}^{(r+1)} \tilde{M}^{(r)}$ azoknak a Gauss mátrixoknak a szorzata, amikkel az r. sort kinulláztuk.

$$\tilde{M}^{(i)} = \begin{pmatrix} 1 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & -\mu_i & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

, i = r, r+1, ...,m-1 , ahol $\mu_i = u_{r,i+1} / u_{i+1,i+1}$ és μ_i az r. sor i+1. oszlopában található.

Ekkor

$$(P^O)^T \tilde{M} L^{-1} \tilde{B} P^O = \tilde{U}$$

, tehát az új bázis a frissített $\tilde{U}$ mátrixszal kifejezve:

$$\tilde{B} = L \tilde{M}^{-1} P^O \hat{U} (P^O)^T$$

Tekintsünk egy példát a Forrest-Tomlin frissítés működésére:

$$B = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}$$

Az előző példák alapján tudjuk, hogy

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} B = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & -1 \\ 0 & 0 & 1 \end{pmatrix} = U$$

Tegyük fel, hogy a $B = (A_1, A_2, A_3)$ bázisban kicseréljük az A_1-et $A_4 = (5, 1, 3)^T$-re (tehát $r = 1$). A_2 és A_3 balra tolása után írjuk be az A_4 -et.

$$\tilde{B} P^O = \begin{pmatrix} 5 & 1 & 1 \\ 1 & 2 & 1 \\ 3 & 1 & 2 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 5 \\ 2 & 1 & 1 \\ 1 & 2 & 3 \end{pmatrix}$$

Szükségünk van még

$$f = \tilde{M}^{(2)} \tilde{M}^{(1)} A_4 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 5 \\ 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 5 \\ -9 \\ -11 \end{pmatrix}$$

vektorra. Ekkor

$$U' = L^{-1} \tilde{B} P^O = (U_{2,} U_{3,} f) = \begin{pmatrix} 1 & 1 & 5 \\ -1 & -1 & -9 \\ 0 & 1 & -11 \end{pmatrix}$$

Ezután nullázzuk ki az *r.* sort (r = 1) a második és harmadik sor segítségével!

$$\tilde{M}^{(1)} U' = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 5 \\ -1 & -1 & -9 \\ 0 & 1 & -11 \end{pmatrix} = \begin{pmatrix} 0 & 0 & -4 \\ -1 & -1 & -9 \\ 0 & 1 & -11 \end{pmatrix}$$

Több átalakításra már nincs is szükség, hiszen ezzel a lépéssel a második elem is kinullázódott az első sorban. $\tilde{M} = \tilde{M}^{(1)}$. Már csak arra van szükség, hogy az első sort a mátrix végére vigyük, a többi sort pedig eggyel felcsúsztassuk. Ezt $(P^O)^T$-vel való szorzással érhetjük el balról.

$$\hat{U} = (P^O)^T U' = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & -4 \\ -1 & -1 & -9 \\ 0 & 1 & -11 \end{pmatrix} = \begin{pmatrix} -1 & -1 & -9 \\ 0 & 1 & -11 \\ 0 & 0 & -4 \end{pmatrix}$$

Tehát megkaptuk a frissített felső trianguláris U mátrixot. $\hat{U} = (P^O)^T \tilde{M} L^{-1} \tilde{B} P^O$ .Ebből kifejezve a módosított bázis nem más, mint $\tilde{B} = L \tilde{M}^{-1} P^O \hat{U} (P^O)^T$. Ezt visszahelyettesítve a megoldandó egyenletbe, azt teljesen hasonló módon meg tudjuk oldani, mint a Bartels-Golub update esetén, új változók bevezetésével.

2. A példaprogram bemutatása

Az előző fejezetben bemutatott módszereket megvalósító függvénykönyvtárhoz készítettem egy példaprogramot, amely bemutatja annak használatát. A példaprogram neve Linsys, Borland C++ Builder 5 környezetben íródott, lineáris egyenletrendszerek megoldását végzi.

Bemenetként szükséges megadnunk egy n ismeretlent tartalmazó, m egyenletből álló lineáris egyenletrendszert, ahol $n \geq m$. Ezt megtehetjük úgy, hogy betöltjük egy MPS formátumú fájlból vagy akár manuálisan is megadhatjuk az együtthatók értékét.

Mivel az egyenletrendszerek több ismeretlent tartalmazhatnak mint egyenletet, ezért n-m változó értékét szabadon meghatározhatjuk. Az egyszerűség kedvéért ezek a változók 0 értéket vesznek fel, valamint ugyanezt a problémát kell megoldanunk a Szimplex módszer egy-egy iterációja során is.

Így a programon belül meg kell adnunk azokat a változókat, amikre ki szeretnénk számolni az egyenletrendszert, gyakorlatilag ezzel megadva a bázist. Így lehetőségünk van egyrészt az LU felbontás használatára az egyenletrendszer megoldása során, valamint a bázis megváltoztatásakor felhasználhatjuk a bázismátrix frissítéséről leírtakat. Ezen kívül a program tartalmazza azt a lehetőséget is, hogy skálázzuk a bázis együtthatóit akár csak oszloponként vagy csak soronként, vagy mindkettőt egyszerre is, ezzel növelve a számítás pontosságát. A programba beépített korlátok maximum ezer változót és ezer egyenletet engednek meg.

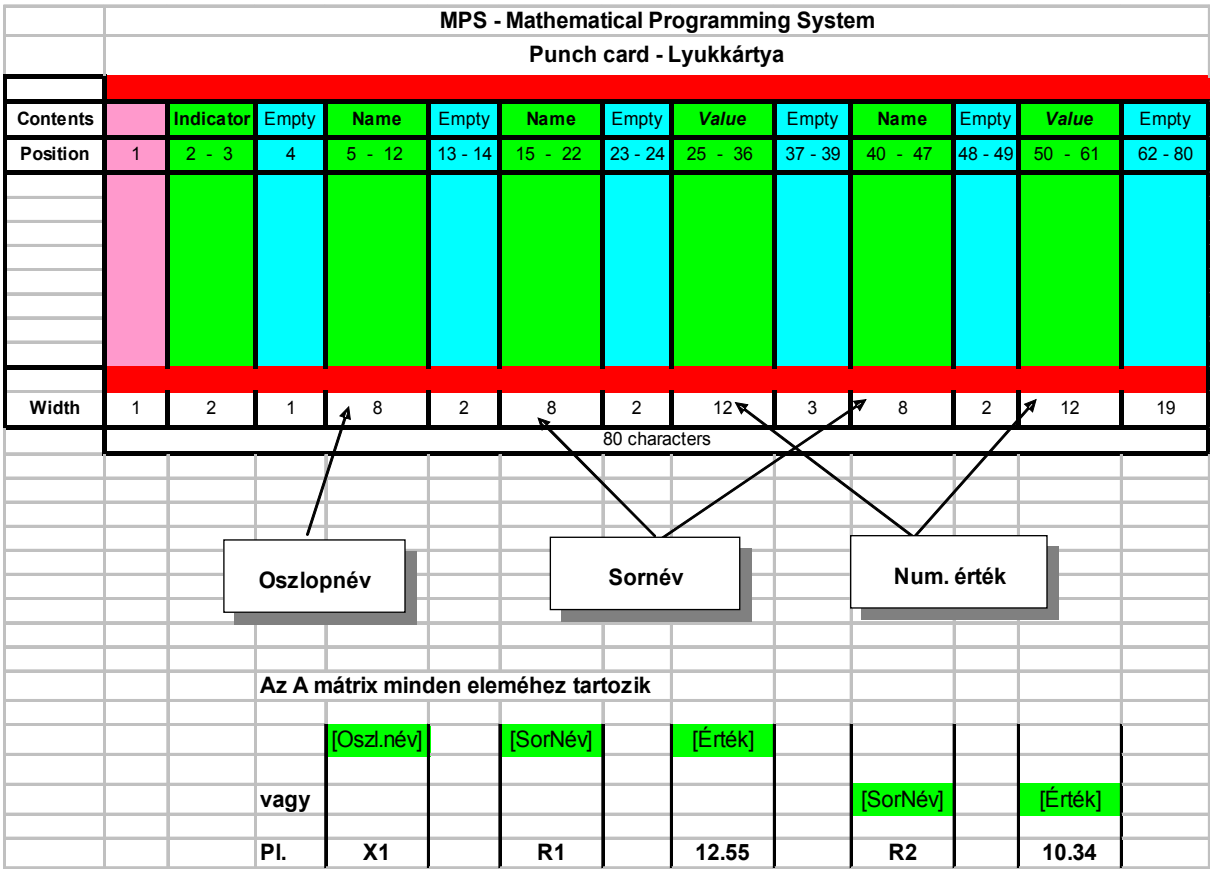
A program írása során felhasznált legfontosabb két külső modul az AdvStringGrid komponens és a GLPK programcsomag.

Az AdvStringGrid komponens a Borland StringGrid komponens egy továbbfejlesztett változata, amelyet a TMS Software cég készített. Egy rácsos, táblázatkezelőhöz hasonló felületet biztosít, számos kényelmi funkcióval.

A GLPK programcsomag nem más, mint a GNU Linear Programming Kit, amely egy függvénykönyvtár, amely nagyméretű lineáris programozási problémák megoldására használható. Ennek a program írása során az MPS fájlok beolvasását végző funkcióját használtam fel.

2.1 Az MPS formátum

IBM cég által bevezetett matematikai programozási feladatok tárolási szabványa (szöveges állományokban), alapja a 80 oszlopos lyukkártya. Főként a lineáris programozásban és mátrix-cserére használják. A lényege, hogy 6 mereven rögzített fix hosszúságú mező alkotja a sorokat. A feladat optimalizálási célját nem tartalmazza!



Az MPS fájlok kulcsszavai:

- NAME

  Csak az első sorban szerepelhet az 1-4 pozícióban. 15-22 pozícióban tartalmazhatja a feladat azonosítóját. Az adott sor előtt elhelyezett adatoknak nincsen semmi szerepük.

- ROWS

  Csak egy külön sorban szerepelhet az 1-4 pozícióban, a sor nem tartalmazhat semmi

mást. A sorok azonosítóit tartalmazó szekció elején áll, az utána következő sorok tartalmazzák a sorok (cél-függvény és feltételek) definícióját. Csak az adott szekcióban bevezetett azonosítókkal rendelkező sorok szerepelhetnek a ROWS szekció után következő szekciókban.

Sor definíció:=<típus> <azonosító>

<típus > Pozíció: 2 (vagy 3 – felesleges szabadság). N cél-függvény, L ≤, E =, G ≥

<azonosító > Pozíció: 5-12. Sor azonosítója – alfa-numerikus karakterlánc. Szóközt, +,-,*,=, nem tartalmazhat.

- COLUMNS

  Csak egy külön sorban szerepelhet az 1-7 pozícióban, a sor nem tartalmazhat semmi mást. Az oszlopok (változók) azonosítóit tartalmazó szekció elején áll, a szekció második szerepe – a feladatban szereplő együtthatók meghatározása (kivéve a jobboldali vektor együtthatóit).

  A szekció minden sora a következő struktúrájú:

  <oszlopnév> <sornév> <érték> , ahol

  Pozíció 5-12 - <oszlopnév>

  Pozíció 15-22 - <sornév>

  Pozíció 25-36 - <érték>

  Csak új vagy az előző sorban szereplő <oszlopnév> szerepelhet és csak olyan <sornév>, amely szerepel ROWS szekcióban. Kötelező sorrend: <oszlopnév> szerint csoportosan.

- R HS

  Csak egy külön sorban szerepelhet az 1-3 pozícióban, a sor nem tartalmazhat semmi mást. A jobboldali (RHS) vektor(ok)ról szóló információt tartalmazó szekció elején áll. A szekció több RHS vektort tartalmazhat.

  A szekció minden sora a következő struktúrájú: <RHSvektornév> <sornév> <érték> ahol

Pozíció 5-12 - <RHSvektornév>

Pozíció 15-22 - <sornév>

Pozíció 25-36 - <érték>

Csak olyan <sornév> szerepelhet, amely szerepel ROWS szekcióban. Kötelező sorrend: < RHSvektornév > szerint csoportosan.

- RANGES

Opcionális kulcsszó, csak egy külön sorban szerepelhet az 1-6 pozícióban, a sor nem tartalmazhat semmi mást. A tartomány alakú feltételekről szóló információt tartalmazó szekció elején áll.

A szekció minden sora a következő struktúrájú: <RANGESvektornév> <sornév> <R>

ahol

Pozíció 5-12 - <RANGESvektornév>

Pozíció 15-22 - <sornév>

Pozíció 25-36 - <R>

Csak olyan <sornév> állhat itt, amely szerepel ROWS szekcióban. Kötelező sorrend: <RANGESvektornév > szerint csoportosan.

R érték kialakítása:

Ha adott a következő alakú tartományfeltétel: $L_i \le \sum_{j=1}^{n} a_{ij} x_j \le U_i$, akkor $R = U_i - L_i$, és U_i szerepel az RHS szekcióban.

- BOUNDS

Opcionális kulcsszó, csak egy külön sorban szerepelhet az 1-6 pozícióban, a sor nem tartalmazhat semmi mást. A változók korlátairól szóló információt tartalmazó szekció elején áll. Alapértelmezés szerint minden változó nem-negatív: x(j) >=0, j=1,2,..,n

A szekció minden sora a következő struktúrájú: <relációtípus> <oszlopnév> <K>

<relációtípus> :

- LO alsó határ K <=x(j) <= ∞
- UP felső határ 0 <=x(j) <= K
- FX rögzített változó x(j) = K
- FR korlátlan változó -∞ <=x(j) <= ∞
- MI nem-pozitív -∞ <=x(j) <= 0
- PL nem-negatív 0 <=x(j) <= ∞ (alapértelmezett)

- ENDATA

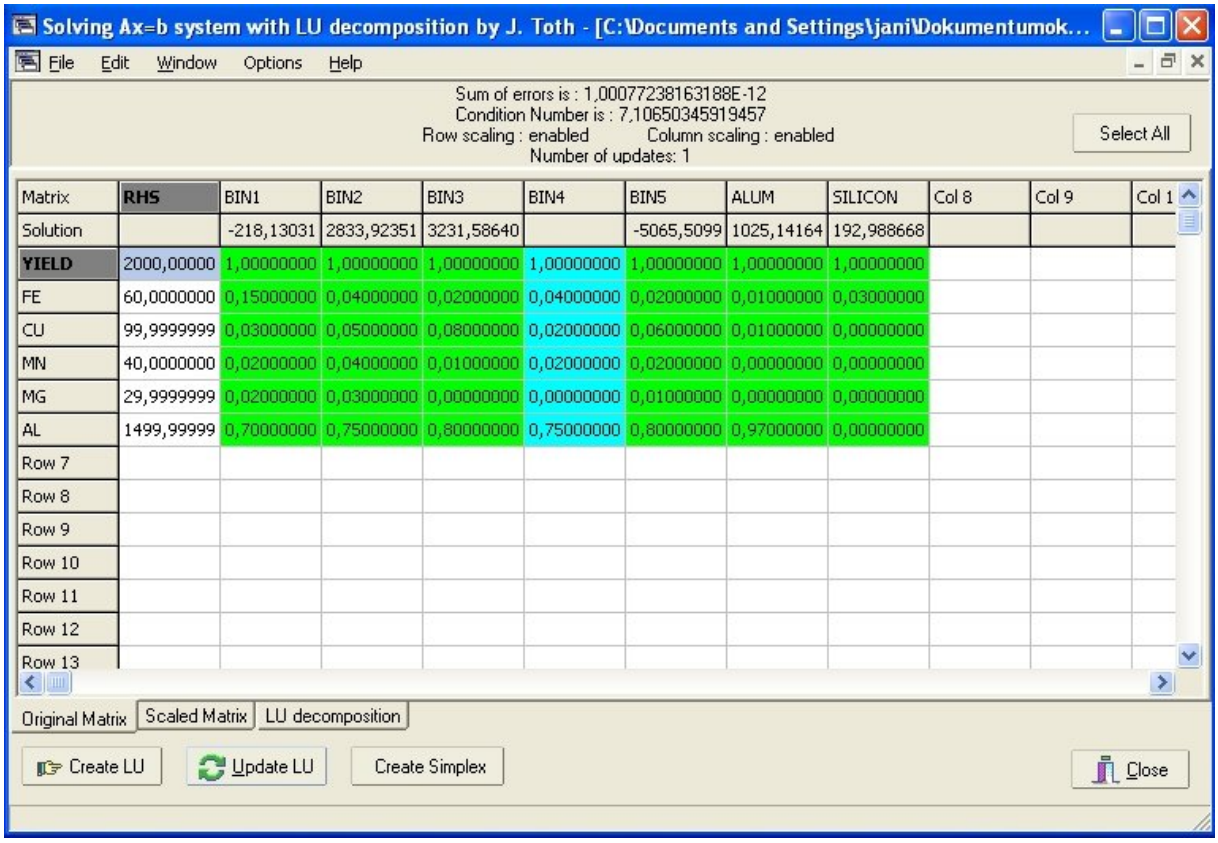
Opcionális kulcsszó, csak egy külön sorban szerepelhet az 1-6 pozícióban, a sor nem tartalmazhat semmi mást. A fájl végén áll, csak egy „D” betűvel !

MPS fájl:

```
NAME          PELDA
ROWS
 N  OBJ
 L  ROW1
 L  ROW2
COLUMNS
    X1        OBJ       1
    X1        ROW1      2
    X1        ROW2      12
    X2        OBJ       2
    X2        ROW1      10
    X2        ROW2      5
    X3        OBJ       3
    X3        ROW2      6
RHS
    RHS1      ROW1      150
    RHS1      ROW2      250
ENDATA
```

2.2 A program használata

A Linsys alkalmazás számára a megoldandó feladatot kétféleképpen adhatjuk meg. Egyrészt a program által biztosított felületen megadhatjuk az együtthatókat, vagy MPS fájlból tölthetjük be azokat. Álljon itt egy kép a program felhasználói felületéről!



Matrix	RHS	BIN1	BIN2	BIN3	BIN4	BIN5	ALUM	SILICON	Col 8	Col 9	Col 1
Solution		-218,13031	2833,92351	3231,58640		-5065,5099	1025,14164	192,988668			
YIELD	2000,00000	1,00000000	1,00000000	1,00000000	1,00000000	1,00000000	1,00000000	1,00000000			
FE	60,0000000	0,15000000	0,04000000	0,02000000	0,04000000	0,02000000	0,01000000	0,03000000			
CU	99,9999999	0,03000000	0,05000000	0,08000000	0,02000000	0,06000000	0,01000000	0,00000000			
MN	40,0000000	0,02000000	0,04000000	0,01000000	0,02000000	0,02000000	0,00000000	0,00000000			
MG	29,9999999	0,02000000	0,03000000	0,00000000	0,00000000	0,01000000	0,00000000	0,00000000			
AL	1499,99999	0,70000000	0,75000000	0,80000000	0,75000000	0,80000000	0,97000000	0,00000000			
Row 7											
Row 8											
Row 9											
Row 10											
Row 11											
Row 12											
Row 13											

2.2.1 Új feladat létrehozása

A **File>New** parancs segítségével hozhatunk létre új feladatot. Az itt megnyíló párbeszédablakban adhatjuk meg az új feladat tulajdonságait. Pontosabban azt, hogy mennyi egyenletből álljon és hány változót tartalmazzon. A változók számának nagyobbnak vagy egyenlőnek kell lennie, mint az egyenletek száma. A megadható legnagyobb érték az ezer. Az együtthatók alapértelmezésben 0-k. Ezeket, valamint az egyenletrendszer jobboldalát (RHS) egy rácsos felületen módosíthatjuk, amelyet az AdvStringGrid komponens biztosít.

2.2.2 Feladat betöltése

A **File->Open** parancs segítségével már létező MPS fájlból tölthetjük be a feladatot. Ez az MPS csak rögzített (fixed MPS) lehet, aminek már fentebb megadtam a leírását, szabad formátumú vagy tömörített nem. A program az MPS fájlból csak azokat a megszorításokat olvassa be, amelyek egyenlőségek, valamint nincs tekintettel a változókra megadott korlátozásokra. Betöltés után a változók és megszorítások nevei egy rácson jelennek meg az együtthatókkal együtt. Az együtthatókat bármikor módosíthatjuk, dupla kattintás után szerkeszthetőek.

2.2.3 Bázis kiválasztása

A feladat megoldásához szükségünk van még a bázisvektorok kiválasztására. Az oszlopvektorokból pontosan annyit kell kiválasztanunk, amennyi egyenletet tartalmaz a rendszer. Ezt úgy tehetjük meg, hogy a kívánt oszlop fejlécére kattintunk. Még egy kattintásra az adott vektor kikerül a bázisból. Nagy számú egyenlet esetén jól jöhet a **Select All** gomb, ami az összes oszlopvektort kiválasztja.

2.2.4 Beállítások

A feladat megoldása előtt érdemes egy pillantást vetnünk az **Options** menüpontra. Itt állíthatjuk be, hogy az együtthatómátrix sorai (Row scaling), oszlopai (Column scaling) legyenek-e skálázva. Valamint kiválaszthatjuk, hogy ez a skálázás milyen módszerrel történjen. Röviden összefoglalva a két lehetőséget a következő a különbség.

A Hall-szabály szerint skálázott mátrixban egy-egy iteráció során a skálázási faktort úgy határozzuk meg, hogy összeszorozzuk az összes nem-nulla elem abszolút értékét a vektorban és n-edik gyököt vonunk belőle, ahol n a nem-nulla elemek száma.

A Gondzio-szabály szerint vesszük a nem-nulla elemek közül az abszolút értékben legnagyobbat és legkisebbet és szorzatukból gyököt vonunk.

Lehetőségünk van még megadni, hogy a program mit tekintsen a legkisebb lebegőpontos értéknek (Set precision), ez feladatonként más-más lehet. Egyúttal ez azt is megadja, hogy a rácson a számok hány tizedesjegyig legyenek ábrázolva. Alapértelmezésben ez az érték 1e-15.

2.2.5 A feladat megoldása

Miután megadtuk a feladatot, az együtthatókat és az RHS vektort, beállítottuk az opciókat és megadtuk a bázisvektorokat, amiknek a száma pontosan az egyenletek számával egyezik, akkor már megoldhatjuk az egyenletet, a fő panel alsó részén található Create LU gomb segítségével. Ez létrehozza a kiválasztott bázis LU felbontását és el is végzi az egyenlet megoldását.

Az LU mátrixot meg is tekinthetjük a megjelenő LU decomposition fülön. Az eredeti mátrix lapján pedig megjelenik a megoldás a Solution sorban, a neki megfelelő változó neve alatt. Ha a skálázást bekapcsoltuk, akkor egy Scaled matrix lap is létrejön, ahol pedig a skálázott mátrix-együtthatókat nézhetjük meg, valamint az első sorban illetve oszlopban az adott sorhoz illetve oszlophoz tartozó skálázási faktort.

Megoldás után a felső panelen kiírásra kerülnek a megoldás különböző paraméterei. Az első a megoldás abszolút hibáinak összege (Sum of errors), továbbá hogy bekapcsoltuk-e a sorok skálázását (Row scaling), valamint az oszlopok skálázását (Column scaling). Ezen kívül kiírja a bázis mátrix kondíciószámát, ami ebben az esetben a mátrix abszolút értékben legnagyobb és legkisebb nem-nulla elemének hányadosa. Az utolsó adat pedig az eddigi frissítések száma a megoldás során (Number of updates).

2.2.6 A bázis frissítése

Amint létezik már a feladat egy bázisának LU felbontása, akkor lehetőségünk van azt felhasználni a feladat megoldására egy másik bázisban. Ha nem módosítjuk az alapfeladatot, akkor ha más bázist választunk az egyenletrendszernek, akkor aktívvá válik az Update LU gomb, ami az új bázisban oldja meg a feladatot úgy, hogy a régi LU felbontását frissíti a Bartels-Golub módszerrel, majd meghatározza annak megoldását.

2.2.7 A Szimplex táblázat kiszámolása

Ha a megadott feladat meg van oldva akár új LU felbontás létrehozásával, akár frissítéssel, akkor létrehozhatjuk a feladat szimplex tábláját. Ez nem más, mint hogy az együtthatómátrix nem-bázis oszlopvektorait átszámoljuk az új bázisba. A Create Simplex gombra kattintva létrejön ez a táblázat, egy új fül jelenik meg (Simplex tableau), amin ez látható is. Baloldalt az

első oszlopban az aktuális bázis látható, mellette pedig az eredeti feladat átalakítottja az új bázisban.

3. A függvénykönyvtár bemutatása

Ebben a részben szeretném bemutatni a liblinsys függvénykönyvtár főbb funkcióit.

3.1 A felhasznált adatszerkezetek

A könyvtár alapvetően a linsys struktúra köré épül. Ez tárolja el a lineáris rendszer jellemzőit és a megoldáshoz szükséges plusz információkat. Ennek a struktúrának a tagja egy linmx struktúra, ami az egyenletrendszer együtthatóit, a jobb oldali vektort és a skálázási faktorokat tartalmazza. A megadható feladat mérete nincs korlátozva, annak csak a felhasználható memória szab határt. A memóriafoglalás dinamikusan történik az együtthatómátrix esetében, a frissítés során használt éta mátrixok száma viszont előre megadott a könyvtárban definiált konstanssal. Nézzük meg részletesebben, hogy mit is tartalmaznak a fent említett struktúrák!

3.1.1 A linsys struktúra

```
typedef struct _eta_mx{
         int col;
         MY_BOOL swap;
         double  val;
} eta_mx;

typedef struct _linsys{
        linmx* extmx;
        double* lu;
        int * bases;
        int * perm;
        int    num_updates;
        int    num_eta;
        eta_mx  Q[MAX_ETA];
        MY_BOOL luvalid;
        MY_BOOL rowscaling;
        MY_BOOL colscaling;
} *linsys,LinSystem;
```

Az *extmx* struktúrában kerülnek eltárolásra a lineáris egyenletrendszer együtthatói, a *b* vektor (RHS), valamint az oszlop- és sorskálázási faktorok. A konkrét leírása a következő pont témája.

Az *lu* egy egydimenziós, lebegőpontos értékeket tartalmazó tömb mutatója, amibe a kiválasztott bázis LU felbontása kerül sorfolytonosan.

A *bases* egy egészeket tartalmazó tömb. Az i. eleme azt jelenti, hogy az i. oszlopvektor a bázisban hányadik oszlopvektornak felel meg az eredeti feladatban.

A *perm* egy egészeket tartalmazó tömb. Egy permutációs mátrixot reprezentál. Az i. eleme azt jelenti, hogy a permutációs mátrix i. sorában hányadik helyen van az egyes. Ez a sorcseréket végző permutációs mátrix nem más, mint az elméleti részben már tárgyalt *P* mátrix. Azt rögzíti, hogy milyen cseréket kellett végrehajtanunk az eredeti bázison, hogy el tudjuk végezni az LU felbontást. Ez a megoldásnál lesz fontos, hiszen az RHS vektoron is ugyanezeket a cseréket kell elvégeznünk.

A *num_updates* a bázison elvégzett frissítések száma.

A *num_eta* a Q tömb aktuális elemeinek száma.

Q tartalmazza azokat a mátrixokat, amiknek segítségével a frissítés során újra felső triangulárissá alakítottuk U-t. Mivel ezek sorcserék, illetve olyan mátrixok amik a főátlóban 1-et, ezen kívül pedig csak egy helyen tartalmaznak nem-nulla elemet, reprezentálhatók az alábbi módon az *eta_mx* struktúrával. Egyrészt tároljuk annak az oszlopnak az indexét, ami a főátlóban szereplő 1-es alatt a nem-nulla értéket tartalmazza (*eta_mx->col*), másrészt azt, hogy a művelet végrehajtása előtt meg kell-e cserélni a *col* és *col*+1. sort. Természetesen a bizonyos nem-nulla értéket is megőrizzük (*val*).

Az *luvalid* egy logikai változó, _TRUE akkor, ha a struktúrában szereplő *lu* mátrix érvényes, ha sikeresen lefutott a felbontás és az valóban a struktúrában megadott bázis felbontása. Egyébként _FALSE az értéke.

A *rowscaling* és a *colscaling* azt rögzíti, hogy az együtthatómátrix sorai illetve oszlopai skálázva lettek-e.

3.1.2 A linmx struktúra

```
typedef struct _linmatrix{
     int rows;
     int cols;
     double eps;
     double* data;
} linmx;
```

A *rows* a mátrix sorainak száma, ennyi egyenletet tartalmaz a rendszer.

A *cols* az oszlopok száma, vagy másképpen a változók száma.

Az *eps* a feladatra jellemző érték, megadható, hogy az algoritmusok során mit tekintünk a legkisebb nem-negatív lebegőpontos számnak. Alapértelmezésben ez 1e-15.

A *data* tömb a következő mátrix sorfolytonos ábrázolása:

$$\begin{pmatrix} * & \gamma_1 & \gamma_2 & \cdots & \gamma_n & * \\ \rho_1 & a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ \rho_2 & a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \rho_m & a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{pmatrix}$$

, ahol n a változók száma (*cols*), m az egyenletek száma (*rows*), γ_i ρ_i az oszlopok illetve sorok skálázási faktora, a_{ij} pedig az együtthatómátrix skálázás után.

3.2 A függvénykönyvtár használata

3.2.1 Főbb funkciók

linsys_ls_init(int rows, int cols);

Új lineáris egyenletrendszert inicializáló függvény. Szükséges megadni a sorok és oszlopok számát. A memóriát dinamikusan foglalja le, minden együtthatót és a b vektort kinulláz, a skálázási faktorokat 1-re állítja. Az együtthatókat direkt módon kell megadnunk a linmx struktúra segédfüggvényeinek segítségével. Minden új példányra használatuk befejeztével hívjuk meg a következő függvényt.

void ls_dispose(linsys ls);

Felszabadítja az *ls* által lefoglalt erőforrásokat.

int ls_lu_crout(double m[], int n,int perm[],double eps);

Valójában ez egy segédfüggvény, amelyet a könyvtár egy magasabb szintű funkciója használ, azonban ez ettől függetlenül is hasznos lehet.

Bemenő paraméterei: *m* egy tömb, ami a felbontandó mátrixot tartalmazza sorfolytonosan, *n* a mátrix sorainak/oszlopainak száma, *eps* az, amit az algoritmus a legkisebb pozitív lebegőpontos számnak tekint.

Kimenő paraméterek: *m* tartalmazza az algoritmus végén a bemenő mátrix LU felbontását, a *perm* tömb pedig rögzíti a végrehajtott sorcseréket. Az i. eleme azt jelenti, hogy a permutációs mátrix i. sorának hányadik eleme az egyes.

A visszatérési érték 0, ha a mátrix szinguláris, 1 ha sikeres a felbontás és páros számú sorcsere történt, -1 ha páratlan.

int ls_create_lu(linsys ls, int newbase[]);

Az *ls* változóban megadott lineáris egyenletrendszer *lu* mezőjét tölti ki, elvégzi az LU felbontást a *newbase* paraméter által meghatározott bázishoz. A *newbase* tömbnek pontosan annyi eleme kell hogy legyen, mint ahány változót tartalmaz az egyenletrendszer. Az i. eleme 1, ha az eredeti egyenlet i+1. változója része a bázisnak, különben 0. Pontosan annyi 1-es

szerepelhet, ahány egyenlet alkotja az egyenletrendszert.

Mielőtt megoldhatnánk az egyenletrendszert, legalább egyszer le kell futtatnunk a függvényt.

Ha bármilyen hiba lépett fel a működés során, 0-va tér vissza, siker esetén 1-gyel.

int ls_solve(linsys ls,double b[]);

Az ls rendszerhez tartozó bázis LU felbontását felhasználva megoldja az egyenletrendszert a paraméterben megadott *b* vektorra. A *b*-nek pontosan ugyanannyi eleme kell hogy legyen, mint ahány egyenlet alkotja *ls*-t.

A megoldást a *b* vektorban adja vissza. Azt, hogy b[i] melyik változónak az értéke, a ls->bases[i] adja meg.

int ls_updatelu(linsys ls, int entering, int leaving);

Az *ls* egyenletrendszer LU felbontásának frissítését végzi. A belépő új oszlopvektor sorindexét az eredeti mátrixban az entering paraméter, a bázisból kikerülőét a leaving paraméterben adhatjuk meg. A függvény frissíti U-t, valamint *Q*-ban rögzíti a frissítésben használt mátrixokat.

0-val tér vissza, ha az új bázis szinguláris, egyébként 1-gyel.

void ls_scaleHall(linsys ls,int rowscale,int colscale);

Az l*s* rendszer együtthatómátrixának skálázását végzi a Hall-szabály szerint. Ha a *rowscale/colscale* paraméter 1, akkor skálázza a sorokat/oszlopokat.

void ls_scaleGondzio(linsys ls, int rowscale,int colscale);

Az l*s* rendszer együtthatómátrixának skálázását végzi a Gondzio-szabály szerint. Ha a *rowscale/colscale* paraméter 1, akkor skálázza a sorokat/oszlopokat.

3.3 Segédfüggvények

3.3.1 A linsys struktúrához

linsys_ls_init_copy (linsys ls);

Az *ls* egyenletrendszer alapján létrehoz egy új másolatot.

int ls_varinbase(linsys ls,int varno);

1-gyel tér vissza, ha az eredeti mátrixban *varno* indexű oszlopvektor eleme a bázisnak, egyébként 0-val.

int ls_applypermut(linsys ls,double v[]);

Alkalmazza ugyanazt a permutációt a v vektor elemeire, mint amit az egyenletrendszer LU felbontása során alkalmaztunk (ls->perm). A v elemeinek száma természetesen megegyezik az egyenletek számával.

void ls_forwardsub(linsys ls, double b[]);

Előrehelyettesítést végez az egyenletrendszerhez tartozó L mátrix alapján a *b* vektorral. A *b*-ben gyakorlatilag $L^{-1}b$-t kapjuk meg.

void ls_backward sub(linsys ls, double b[]);

Visszahelyettesítést végez az egyenletrendszerhez tartozó U mátrix alapján a *b* vektorral. A *b*-ben gyakorlatilag $U^{-1}b$-t kapjuk meg.

double Hall factor(double v[],int n);

Az *n* elemű *v* vektorhoz számolja ki a Hall-szabály szerint, hogy milyen számmal faktorizáljon és el is végzi az osztást.

double Gondzio factor(double v[], int n);

Az *n* elemű *v* vektorhoz számolja ki a Hall-szabály szerint, hogy milyen számmal faktorizáljon és el is végzi az osztást.

3.3.2 A linmx struktúrához

linmx* linmx_create(int rows, int cols);

Létrehoz egy *linmx* struktúrát, aminek *rows* darab egyenlete és *cols* darab változója van. A memóriafoglalás dinamikus, minden létrehozott *linmx* struktúrára meg kell hívnunk az alábbi függvényt.

void linmx_dispose(linmx *mx);

Felszabadítja az *mx* által használt erőforrásokat.

int linmx_copy(linmx* from,linmx *to);

Egy másolatot hoz létre a *from* -ról a *to*-ba. A *to* eddigi adatai elvesznek!

inline double linmx_getat(linmx* mx, int i, int j);

Lekérdezhetjük az *mx* i. egyenletéből a j. változó együtthatóját. (1 alapú indexelés!)

inline double linmx_setat(linmx* mx,int i, int j, double val);

Beállíthatjuk az *mx* i. egyenletéből a j. változó együtthatóját a *val* értékre. (1 alapú indexelés!)

inline double linmx_getrowfactor(linmx *mx,int row);

Lekérdezhetjük az *mx row*. sorának skálázási faktorát. (1 alapú indexelés!)

inline double linmx_getcolfactor(linmx* mx, int col);

Lekérdezhetjük az *mx col*. oszlopának skálázási faktorát. (1 alapú indexelés!)

inline double linmx_setrowfactor(linmx* mx, int row,double val);

Beállíthatjuk az *mx row*. sorának skálázási faktorát *val*-ra. (1 alapú indexelés!)

inline double linmx_setcolfactor(linmx *mx,int col,double val);

Beállíthatjuk az *mx col*. oszlopának skálázási faktorát *val*-ra. (1 alapú indexelés!)

inline double linmx_updaterowfactor(linmx* mx, int row,double val);

Az *mx row*. sorának skálázási faktorát szorozza *val* értékkel. (1 alapú indexelés!)

inli ne doub le lin mx _upd ate co lf ac to r(li nm x* mx,in t col, doub le val) ;

Az *mx col.* oszlopának skálázási faktorát szorozza *val* értékkel. (1 alapú indexelés!)

inli ne doub le lin mx _g et rhs(li nm x* mx, int row) ;

Az *mx* RHS vektorának *row*. elemével tér vissza.

double * li nmx _g et rhs ve ct (l in mx * mx);

Az *mx* egész RHS vektorával tér vissza. A tömb dinamikusan kerül lefoglalásra, a visszaadott mutatóra mindenképpen hívjuk meg a free() -t használat után!

inli ne doub le lin mx _se tr hs(li nm x* mx,i nt row,d oubl e val);

Az *mx* RHS vektorának *row*. elemét állítja *val*-ra. (1 alapú indexelés!)

double * lin mx_ ge tc ol (l in mx * mx,i nt col);

Az *mx col.* oszlopvektorával tér vissza. A tömb dinamikusan kerül lefoglalásra, a visszaadott mutatóra mindenképpen hívjuk meg a free() -t használat után! (1 alapú indexelés!)

void linm x_s et co l(li nm x* mx, int co l, doubl e val[ ]) ;

Az *mx col.* oszlopát állítja be a *val*-ban található értékekre.

double * lin mx_ ge tr ow(li nm x* mx, int row) ;

Az *mx row.* sorvektorával tér vissza. A tömb dinamikusan kerül lefoglalásra, a visszaadott mutatóra mindenképpen hívjuk meg a free() -t használat után! (1 alapú indexelés!)

void linm x_s et row (l in mx * mx,in t row, double va l[]);

Az *mx row.* sorát állítja be a *val*-ban található értékekre.

double li nmx _g et co nd(li nm x* mx);

Az *mx* együtthatómátrixának kondíciószámával tér vissza. Ez nem más ebben az esetben, mint az abszolút értékben legnagyobb és legkisebb nem-nulla elem hányadosa.

Összefoglalás

A kereskedelmi forgalomban lévő operációkutatási szoftverekben használt számos numerikus módszer közül, ami arra hivatott, hogy az algoritmusok stabilitását és a megoldás megbízhatóságát növelje, elsősorban a Szimplex módszer javítására történő eljárásokat sikerült nagy vonalakban áttekintenem. Természetesen csak a módszerek alapjairól esett szó, hiszen az élesben alkalmazott szoftverekben rendkívül kifinomult technikákat használnak, amiket hatalmas cégek pénzt és időt nem kímélve tökéletesítettek.

Ezek a módszerek a bázismátrix faktorizációja és az ehhez szorosan kapcsolódó frissítés, amely során ezt az LU faktorizációt módosítják a bázismátrix kis változásának megfelelően.

A megértéshez nagy segítséget nyújtott az algoritmusok gyakorlati megvalósítása, amelynek eredménye, egy függvénykönyvtár, ami már alkalmas viszonylag nagy méretű lineáris egyenletrendszerek hatékony megoldására és reményeim szerint könnyen bővíthető újabb funkciókkal is.

Ezt szerettem volna bemutatni a példaalkalmazáson keresztül, amely az említett feladatot, lineáris egyenletrendszerek megoldását végzi.

Irodalomjegyzék

Bajalinov, E.B., Linear-Fractional Programming: Theory, Methods, Applications and Software,

Hillier, F.S., Lieberman, G.J., Introduction to Operations Research,

Bartels, R. H. , Golub. G. H., The simplex method of linear programming using the LU decomposition. Commun. ACM, 12:266–268,

Forrest, J. J. H.. Tomlin. J. A., Updating triangular factors of the basis to maintain sparsity in the product form simplex method. Math. Program., 2:263–278