Debreceni Egyetem
Informatikai Kar

SKÁLÁZÁSI ELJÁRÁSOK ÉRTÉKELÉSE

Konzulens:
Dr. Bajalinov Erik
Tudományos főmunkatárs

Készítette:
Rácz Anett
Programtervező matematikus

Debrecen
2007

Tartalomjegyzék:

I. Skálázás gyakorlati jelentősége

A való élet problémáit leíró modellek jellemzője a nagy méret. Ez a minél realisztikusabb modellezésre való törekvés velejárója. Az, hogy mit nevezünk nagy méretnek egy változó fogalom. Míg a hatvanas években egy néhány száz sorból álló probléma is nagynak számított, addig az ezredfordulóra a több tízezer sorból álló problémákat mondhatjuk tekintélyes méretűnek. A nagy mennyiségű adatokban számok széles intervalluma fordul elő, amiket fel kell dolgozni. Manapság egyre elterjedtebbé válik az operációkutatási szoftverek használata, ez a fejlődés azonban csak akkor fejtheti ki igazán hasznos hatását, ha az egyes alkalmazási területeken (vegyipar, gyógyszergyártás, üzemanyag keverékek előállítása) megkövetelt nagy pontosságot is biztosítani tudják. A véges gépi pontosság miatt kerekítési hibák keletkeznek a számítógéppel történő számítások közben. Ezen hibák kezdetben kicsik, de ha tovább dolgozunk a pontatlan adatokkal, akkor elég naggyá nőhetik ki magukat. A skálázási eljárások szolgálják a célt, hogy ezeket a pontatlanságokat megelőzzük az input-mátrix módosításával. Persze ezen átalakításoknak olyannak kell lenniük, hogy a faktorizált és az eredeti feladat közti ekvivalencia az eredményen történő *"re-faktorizációval"* biztosítva legyen. Egy, a szimplex módszer közben esetlegesen keletkező kis hiba jelentős pontatlansághoz vezethet, hiszen a módszer jellegzetessége a nagy iterációszám, mely lehetőséget ad az apró eltéréseknek, arra hogy elhatalmasodjanak. Ezen hibák a végeredményben is megjelennek és veszteségekhez, problémákhoz vezetnek a való életben. Ugyanez a számítási gond lép fel a nagyméretű hiperbolikus programozási feladatoknál is. Az ilyen problémák elkerülése végett minden jól elkészített operációkutatási szoftver tartalmaz speciális, kifinomult technikát, mely nagymértékben csökkenti a kerekítésből adódó hibahalmozódás jelenséget és gyakran tetemes előrelépést jelent a megoldók teljesítményében is.

A skálázás egy, az ilyen típusú hatékony és elterjedt technikák között. Ez a módszer azt jelenti, hogy az eredeti feladathoz tartozó $A = \|a_{ij}\|_{m\times n}$ mátrix azon sorait és/vagy oszlopait, melyek gyengén skálázottak, azaz elemek széles intervallumát tartalmazzák, egyenként el kell osztani (vagy meg kell szorozni) az ő saját skálázási faktoraival.

A mátrixokat elláthatjuk egy jellemző mennyiséggel, mely jelzi, hogy az adott mátrixszal való számítások során milyen mértékű pontatlanságok várhatóak. Ezt a mérőszámot először 1962-ben Fulkerson és Wolf, majd 1968-ban Orchard-Hays definiálta. Ez a szám a mátrixban a legnagyobb és a legkisebb abszolút értékű nem nulla elem hányadosa. Azaz a mátrixban lévő elemek abszolút értékeinek intervallumhossza a meghatározó.

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

Ezen intervallumhossz szűkítése a cél. Azt is kijelentették, hogy a mátrix akkor jól skálázott, ha ez a mérőszám a 10^6 érték alatt van. Én a tesztelés során 10^3 határértéket használtam, mivel a hardver korlátok miatt a mátrixok mérete korlátozott volt.

A szoftver válaszideje is igen fontos tényező, ami jelentősen megnőhet, ha nagy munkaigényű átalakításokat építünk be. Kompromisszumot kell állítanunk szerzett pontosság és a számítógépes hatékonyság között.

Skálázási módszereket már mások is fejlesztettek ki, munkám során az volt a célom, hogy egy olyan egyszerűen leprogramozható módszert találjak, amely legalább olyan javulást ér el a pontosság terén, mint az eddigi módszerek, de kevésbé függ az input mátrix tulajdonságaitól, és gyorsabb azoknál. Majd ezt a módszert egy saját készítésű tesztelő program segítségével több szempont alapján összehasonlítottam az eddigi eljárásokkal. A tesztelés több szemszögből vizsgálja az egyes módszerek hatékonyságát. Egy ilyen eljárásnál fontos a gyorsaság, az implementálhatóság, egyszerűség és természetesen a hatékonyság. Ezen jellemzőket külön opcióként kiválasztva lehet tesztelni az egyes módszereken. A teszteredményekből grafikonokat készítettem, hogy a következtetéseket szemléltessem. A grafikonok alapjául szolgáló adatokat a függelékben helyeztem el.

II. A véges gépi pontosságról

II.1. A számítógépes számábrázolás

A számítógépen több módon tárolódnak a számok. Beszélhetünk egész számokról és lebegőpontos számokról. Az egész számokkal történő aritmetikai műveletek nagyságrenddel gyorsabbak és hibamentesnek tekinthetők. A lebegőpontos számoknál a helyzet nem ilyen egyszerű. Egy $\pm a^k(\frac{m_1}{a}+\frac{m_2}{a^2}+\ldots+\frac{m_t}{a^t})$ normalizált formában felírt lebegőpontos szám tárolása a következő formában történik: $[\pm,k,m_1,\ldots,m_t]$, ahol $m=(m_1,m_2,\ldots,m_t)$ a mantissza, k kitevőt pedig a szám karakterisztikájának hívjuk. A géptől és pontosságtól függően a mantisszát négy, nyolc vagy tizenhat bájton tárolja, ezzel párhuzamosan nő a k értékkészlete is. Ekkor a legnagyobb ábrázolható szám:

$$M_\infty = a^{k_+}\sum_{i=1}^{t}\frac{a-1}{a^i} = a^{k_+}(1-a^{-t})$$

A k_+ a k értékkészletének legnagyobb eleme, míg a későbbiekben használt k_- az értékkészlet legkisebb eleme. A legkisebb szám $-M_\infty$. Tehát az ábrázolható számok intervalluma a $[-M_\infty, M_\infty]$. Ezen az intervallumon belüli racionális számok egy részhalmazát adják a lebegőpontos számok. Ez a részhalmaz nullára nézve szimmetrikus.

A nulla egy pontosan ábrázolható szám. Fontos azonban, hogy a nullához legközelebbi ábrázolható szám és a nulla közé eső számokat már nem tudjuk pontosan ábrázolni. Ez a szám: $\varepsilon_0 = a^{k_- -1}$. Így a $(-\varepsilon_0,\varepsilon_0)$ intervallumban a nullán kívül nincs más ábrázolható szám. Az egy mindig hozzátartozik a lebegőpontos számokhoz. Az egy utáni következő szám az $1+\varepsilon_1$. Ahol $\varepsilon_1 = a^{1-t}$. Ezt a számot hívjuk relatív pontosságnak vagy gépi epszilonnak. Az ε_0 és ε_1 számok abszolút és relatív hibakorlátként szolgálnak.

II.2. Hibák

Legyen x egy inputként szolgáló szám, amely kisebb M_∞-nél. Jelölje *fl(x)* a számítógép által hozzárendelt lebegőpontos számot. Ez a hozzárendelés a következőképpen történik:

$$fl(x) = \begin{cases} 0 \quad ha\ |x| < \varepsilon_0 \\ \text{az x - hez legközelebbi lebegõpontos szám} \quad ha\ \varepsilon_0 \le |x| \le M_\infty \end{cases}$$

Ekkor előfordul, hogy az eredeti szám és a hozzárendelt lebegőpontos szám értéke nem egyenlő. Ezen hiba mértéke a következőképpen alakul:

$$|fl(x) - x| \le \begin{cases} \varepsilon_0 & ha\ |x| < \varepsilon_0 \\ \frac{1}{2}\varepsilon_1 |x| & ha\ |x| \ge \varepsilon_0 \end{cases}$$

A négy alapműveletet tekintve is előállhatnak hibák. Feltéve, hogy a gép *x* és *y* értékében nem tévedett és a műveletet is pontosan hajtotta végre és ezután az *fl(x◊y)* függvénnyel hozzárendelte a megoldáshoz a lebegőpontos számot, a következő hibák keletkezhetnek:

$$|fl(x \Diamond y) - x \Diamond y| \le \begin{cases} \varepsilon_0 & ha\ |x \Diamond y| < \varepsilon_0 \\ \frac{1}{2}\varepsilon_1 |x \Diamond y| & ha\ |x \Diamond y| \ge \varepsilon_0 \end{cases}$$

Más a helyzet, ha a művelet eredménye nagyobb lesz M_∞-nél, ekkor túlcsordulás történik, illetve, ha $(-\varepsilon_0, \varepsilon_0)$-ból való, ekkor alulcsordulásról beszélünk. Túlcsordulás esetén, kaphatunk információt arról, hogy hiba történ, ám alulcsordulás esetén az eredményt automatikusan nullának tekinti a gép, és hiba nélkül dolgozik vele tovább, így értékes adatok veszhetnek el.

III. Egy kisebb példa a hibahalmozódásra

Egy rövid numerikus példa bemutatásával szeretném illusztrálni, hogy egy apró hiba is milyen nagy pontatlansághoz vezethet a végeredményben. Tekintsük a következő lineáris egyenletrendszert:

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

Ezen feladat pontos megoldása az $x_1 = 10$ és $x_2 = 1$ értékek.

Oldjuk meg az egyenletrendszert Gauss eliminációt használva, négy tizedesjegy pontossággal. Miután kiválasztjuk az $a_{11} = 0.003$ elemet, mint főelemet és ezzel kiszámoljuk a szorzótényezőt, a következő eredményt kapjuk: $\lambda = a_{21} / a_{11} = 5.291 / 0.003 = 1763.666(6)$, amit a négy tizedes pontosság miatt 1763.6667–re kerekítünk. Miután a második oszlopból kivontuk az első oszlop λ-szorosát a következőket kapjuk:

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

Az összefüggésből x_2-re adott megoldás, $x_2 = 1.00000000001$. Ezt a viszonylag pontos eredményt felhasználva x_1 értéke $x_1 = \dfrac{59.17 - (59.140 \cdot 1.00000000001)}{0.003} = -9.7133$, ami nagyon távol áll a feladat egzakt megoldásától.

A feladat során azt történt, hogy a kerekítésből adódott egy 0.00000000001 mértékű apró hiba x_2 értékében, melyet aztán egy nagyon nagy számmal szoroztuk, így az elhatalmasodhatott. Persze a számítógépek ennél jóval pontosabbak, legtöbbjük az IEEE szabvány alapján 16 tizedesjegy pontossággal dolgozik, de ez is véges.

IV. Egyéb eljárások a hibák elkerülésére

IV.1. Bázis mátrix frissítése

A szimplex módszert elvégezve a számítógépen a legtöbb véges gépi pontosságból adódó hiba az egyik fő lépésnél adódhat, amikor az új tábla elemeket számoljuk a régieket felhasználva az alábbi képlet alapján.

$$x'_{ij} = \begin{cases} x_{ij} - \dfrac{x_{rj} x_{ik}}{x_{rk}} & i = 1,2,...,m \quad i \neq r \\ \dfrac{x_{rj}}{x_{rk}} & i = r \end{cases}$$

Ahol *r* jelöli azon vektor indexét, amely kikerül a bázisból és *j* pedig azé, amelyik bekerül a bázisba. Az itt keletkező kis kerekítési hibákat a következő lépésekben újra és újra felhasználjuk, ami nagy pontatlansághoz vezethet.

A szimplex táblában az $x_{\bullet j}$ oszlopvektorok geometriai jelentése, hogy megadják az eredeti *A* mátrix megfelelő oszlopvektorának az aktuális bázisban vett előállítását. Tehát amikor újabb bázisra áttérve ezeket újraszámoljuk, a következő lineáris egyenletrendszereket oldjuk meg.

$$\sum_{i=1}^{m} A_{si} x_{ij} = A_j \quad j \in J_N$$

Ahol J_N a bázison kívüli vektorok indexei. Ez az $Ax=b$ típusú egyenletrendszereknek egy olyan halmaza, ahol a baloldal mindig ugyanaz és a jobboldal váltakozik. Ezen egyenletrendszereket más módszerrel is megoldhatjuk, mint például az Gauss elimináció vagy az LU felbontás. A következő fejezetekben ezek előnyeiről és hátrányairól lesz szó, valamint arról, hogy konkrétan hogyan lehetne ezeket a módszereket a szimplex algoritmusban használni.

IV.2. LU felbontás

Az LU felbontás azon alapszik, hogy a tetszőleges $A \in R^{mxm}$ felbontható egy *L* alsó háromszög és egy *U* felső háromszög mátrix valamint egy *P* permutációs mátrix szorzatára $A = PLU$. Ezen felbontás után az egyenletrendszer megoldható a

$$(1.) \quad Ly = P^{-1}b = P^T b$$
$$(2.) \quad Uz = y$$

lépésekkel.

A permutációs mátrixra azért van szükség, mert mielőtt felbontjuk az *A* mátrixot *A=LU* alakra, ellenőrizni kell, hogy a főátlóban lévő elemek egyike sem nulla. Ha mégis, akkor permutációs mátrixszal szorozva sor vagy oszlopcserét kell végrehajtani.

Az LU felbontás alkalmazásánál, először felírjuk az *A=LU* kifejezést a következő alakban.

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

Amiből az alábbi három összefüggést tudjuk felírni.

$$l_{i1}u_{1j}+l_{i2}u_{2j}+\ldots+l_{ii}u_{ij}=a_{ij} \qquad ha\ i<j \qquad (1)$$
$$l_{i1}u_{1j}+l_{i2}u_{2j}+\ldots+l_{ii}u_{jj}=a_{ij} \qquad ha\ i=j \qquad (2)$$
$$l_{i1}u_{1j}+l_{i2}u_{2j}+\ldots+l_{ij}u_{jj}=a_{ij} \qquad ha\ i>j \qquad (3)$$

Ebben az egyenletrendszerben m^2 egyenlet és m^2+m ismeretlen. Mivel az ismeretlenek száma nagyobb, mint az egyenleteké, le kell rögzíteni m darab ismeretlent. Általános és mindig kivitelezhető a $l_{ii}=1 \quad i=1,..,m$ választás.

Jól bevált eljárás az LU felbontásra a *Crout's algoritmus*:

1. Állítsuk be $l_{ii}=1 \quad i=1,..,m$
2. $j=1,\ldots,m$-re hajtsuk végre az alábbi két lépést
   a. Először $i=1,\ldots,j$ (1)–(3) egyenletrendszert felhasználva

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

   b. Másodszor $i=j+1,\ldots,m$ a (3) összefüggést használva

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

Az 2/a. és 2/b. lépésekben az u_{ij} *és* l_{ij} megjelennek a jobb oldalon is, de ekkora azok már definiáltak.

Az A mátrix a_{ij} elemeit pontosan egyszer használjuk fel, ezért lehetőség van a szükséges memória terület szűkítésére úgy, hogy az a mátrix memória-beli helyén fogjuk letárolni az u_{ij} *és* l_{ij} változókat (az L főátlója csupa egyes, ezért azt nem szükséges letárolni).

IV.3. Az LU felbontás és a Gauss-elimináció kapcsolata

Az LU felbontást megvalósíthatjuk a Gauss-elimináció segítségével is. A Gauss-elimináció első lépéseként az

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

mátrixban az *i.* sorból kivonjuk az első sor μ_{i1} szeresét, ahol $\mu_{i1} = a_{i1} / a_{11}$. Ezen transzformáció mátrix szorzat alakban a követező:

$$A^{(2)} = M^{(1)} A$$

ahol

$$A^{(2)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} & \dots & a_{2m}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} & \dots & a_{3m}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & a_{m2}^{(2)} & a_{m3}^{(2)} & \dots & a_{mm}^{(2)} \end{pmatrix}$$

$$M^{(1)} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ -\mu_{21} & 1 & 0 & \dots & 0 \\ -\mu_{31} & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\mu_{m1} & 0 & 0 & \dots & 1 \end{pmatrix}$$

Ezen lépések ismétléséhez az $M^{(k)}$ mátrix alakja

$$M^{(k)} = \begin{pmatrix} I_{k-1} & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & -\mu_{k+1k} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -\mu_{mk} & 0 & \dots & 1 \end{pmatrix}$$

és $A^{(k+1)} = M^{(k)} P^{(k)} M^{(k-1)} P^{(k-1)} \ldots M^{(1)} P^{(1)} A$. Ahol a $P^{(i)}$ $i = 1,2,\ldots,k$ az esetleges permutációs mátrixok. Az utolsó lépés után, az A^m felső háromszög mátrix előáll az alábbi kifejezés alapján:

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

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

Ezek után végezzünk el egy balról szorzást a $\left(M^{(m-1)}\right)^{-1}$, majd a $\left(P^{(m-1)}\right)^{-1}$ inverzzel. $\left(P^{(m-1)}\right)^{-1}\left(M^{(m-1)}\right)^{-1} A^{(m)} = \left(P^{(m-1)}\right)^{-1}\left(M^{(m-1)}\right)^{-1} M^{(m-1)} P^{(m-1)} M^{(m-2)} P^{(m-2)} \ldots M^{(1)} P^{(1)} A$.

Mivel $\left(M^{(m-1)}\right)^{-1} M^{(m-1)} = I$ és $\left(P^{(m-1)}\right)^{-1} P^{(m-1)} = I$, a fenti kifejezésből a következőt kapjuk:

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

Ehhez hasonló lépések ismétlésével a végső eredmény:

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

Vizsgáljuk meg az $\left(M^{(k)}\right)^{-1}$ inverzeket.

$$\left(M^{(k)}\right)^{-1} = \begin{pmatrix} I_{k-1} & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ 0 & \mu_{k+1k} & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \mu_{mk} & 0 & \ldots & 1 \end{pmatrix}$$

Ezen mátrixokat képezhetjük az eredeti $M^{(k)}$ mátrixból úgy, hogy a főátlós alatti elemek mínusz egy szeresét tekintjük. Ezek a mátrixok mind alsó háromszög mátrixok, és ilyen típusú mátrixok szorzata is alsó háromszög mátrix lesz. Permutációs mátrixok nélkül, az $\prod_{k=1}^{m-1}\left(M^{(k)}\right)^{-1}$ mátrixok szorzatát megfeleltethetjük a L-nek az átalakított $A^{(m)}$ mátrixot pedig U-

nak. Amennyiben a permutációs mátrixok szükségesek voltak, azok átalakításait figyelembe véve és az *A* mátrixon végrehajtva, elvégezhetjük a felbontást.

A permutációs mátrixokat nem csak akkor alkalmazhatjuk, amikor a fődiagonálisban nulla elemet találunk, hanem stabilitásjavításra is. Szokás, hogy a *k.* sorban a főelemnek sorcserével azt az elemet választjuk, mely az oszlopban lévő lehetséges elemek között a legnagyobb.

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

Ezt nevezzük részleges főelem-kiválasztásnak. Amennyiben a felbontás *k.* lépése előtt nem a *k.* oszlop legnagyobb elemét határozzuk meg, hanem a még feldolgozandó teljes $A^{(k)}$ mátrix legnagyobb elemét, akkor teljes főelem-kiválasztásról beszélünk. A gyakorlatban a teljes főelem-kiválasztást nem sűrűn alkalmazzák, mert nagymértékben megnöveli a műveletigényt.

IV.4. A *Bartels-Golub* frissítés

Amikor a szimplex algoritmus során a bázisban kicserélünk egy vektort, az nem nagy változást eredményez a mátrix szerkezetében. Éppen ezért az ötlet, hogy hogyan lehetne az új bázis mátrixának LU felbontását meggyorsítani.

Tekintsük a *B* bázis, mely az $A_j = (a_{1j}, a_{2j}, \ldots, a_{mj})\ j = 1,2,...,m$ oszlopvektorokból áll és ennek az LU felbontását.

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

Ahol $M^{(i)}$ a Gauss elimináció *i.* lépésének transzformációs mátrixa, a $P^{(i)}$ a szükséges permutációs mátrix és *U* a kapott felső háromszög alakú mátrix.

Ezután tegyük fel, hogy új bázisra kell áttérni, miszerint a $B = (A_1, A_2, \ldots, A_{r-1}, A_r, A_{r+1}, \ldots, A_m)$ aktuális bázisban az A_r vektort ki kell cserélni egy A_k vektorra. Az így kapott új bázis $\widetilde{B} = (A_1, A_2, \ldots, A_{r-1}, A_k, A_{r+1}, \ldots, A_m)$. A fenti képletben a *B*-t $\widetilde{B}$-ra cserélve elveszítjük az *U* mátrix felső háromszög jellegét. Az újonnan kapott $\widetilde{U}$ mátrix alakja:

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

Végezzünk el egy permutációt a $\widetilde{B}$ mátrixon, hogy a bekerült A_k vektor az utolsó helyre kerüljön. $\widetilde{B}P^{(R)} = (A_1, A_2, \ldots, A_{r-1}, A_{r+1}, \ldots, A_m, A_k)$. Ezt behelyettesítve az *U*-ra a következő alakot kapjuk:

$$M^{(m-1)}P^{(m-1)}M^{(m-2)}P^{(m-2)}\ldots M^{(1)}P^{(1)}\widetilde{B}P^{(r)} = \widetilde{U}P^{(r)} = \begin{pmatrix} * & * & * & * & * & * & * \\ & * & * & * & * & * & * \\ & & * & * & * & * & * \\ & & * & * & * & * & * \\ & & & * & * & * & * \\ & & & & * & * & * \\ & & & & & * & * \end{pmatrix} = U'$$

Ha bevezetjük *f*-fel jelölve a következő kifejezést:

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

Akkor az *U'*-t felírhatjuk az alábbi alakban:

$$U' = (U_1, U_2, \ldots, U_{r-1}, U_{r+1}, \ldots, U_m, f)$$

$$U' = \begin{pmatrix} u_{11} & \ldots & u_{1,r-1} & u_{1,r+1} & \ldots & u_{1m} & f_1 \\ 0 & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & 0 & u_{r-1,r-1} & \vdots & \vdots & \vdots & f_{r-1} \\ \vdots & \vdots & 0 & u_{r,r+1} & \vdots & \vdots & f_r \\ \vdots & \vdots & \vdots & u_{r+1,r+1} & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & u_{mm} & f_m \end{pmatrix}$$

Ez a mátrix nem sokban különbözik egy felső háromszög alakú mátrixtól. Ahhoz, hogy olyan alakra hozzuk, nem kell végrehajtanunk a teljes LU felbontást. Elegendő a következő lépések végrehajtása, mellyel jelentős időt spórolhatunk.

Hajtsuk végre $j = r+1, r+2, \ldots, m-1$-re az alábbi lépéseket.

1. Cseréljük ki az j. sort a j+1. sorral, amennyiben

$$\tilde{u}_{j,j} \geq \tilde{u}_{j+1,j}$$

Az itt használt $\tilde{P}^{(j)}$ permutációs mátrixok csupán egy sornak a cseréjét hajtják végre.

2. Alkalmazzuk a Gauss elimináció megfelelő lépését a j. sorban lévő főelemmel és az $\tilde{M}^{(j)}$ mátrix segítségével, ahol

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

m-r darab ilyen iteráció után a következőket kapjuk.

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

Vagy

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

Ahol U'' már felső háromszög alakú mátrix.

Meg kell jegyeznünk, hogy a $\tilde{P}^{(j)}$ permutációs mátrixok csupán egyetlen sor cseréjére használandók és a $\tilde{M}^{(j)}$ mátrixok is csak egyetlen fődiagonális alatti elem kinullázására szükségesek. Azaz elég gyors végrehajtás lehetséges.

A fő problémája ennek a módszernek, hogy egy idő után az LU faktorok hossza nagyon megnőhet.

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

Persze erre is születtek megoldások, miszerint periódusonként aktualizáljuk az adatokat.

V. A skálázási eljárások fajtái

Ebben a fejezetben a már meglévő technikák bemutatásáról ejtenék néhány szót. Szeretném felvázolni, hogy milyen átalakítási lehetőségek vannak, és ezek hogyan biztosítják az ekvivalenciát az eredeti és a skálázott feladat megoldása között.

Egy hiperbolikus programozási feladatot általános alakban a következő jelölésekkel írhatunk le:

$$Q(x) = \frac{P(x)}{D(x)} = \frac{\sum_{j=1}^{n} p_j x_j + p_0}{\sum_{j=1}^{n} d_j x_j + d_0} \to \max \qquad \textit{(Ábra 1)}$$

Az alábbi feltételekkel:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1,...,m \qquad \textit{(Ábra 2)}$$

$$x_j \geq 0, \qquad j = 1,...,n \qquad \textit{(Ábra 3)}$$

Ezen terminológiát figyelembe véve, a skálázási eljárások az alábbiak lehetnek:

1.) A feltételek skálázása esetén:

- Csak a jobboldali $b = (b_1,...,b_m)$ vektor módosítása
- A feltételekben lévő együtthatók által alkotott *A* mátrix oszlopainak változtatása.
- Illetve ugyanezen mátrixnak csak a sorainak átalakítása.

2.) A célfüggvényben végzett átalakítások esetén:

- Csak a számlálóban lévő együtthatók alkotta $p = (p_0, p_1,..., p_n)$ vektort skálázzuk.
- Csak a nevezőben lévő együtthatók alkotta $d = (d_0, d_1,..., d_n)$ vektort alakítjuk át.
- Mindkét vektor módosítjuk.

V.1. A jobboldali vektor átalakítása *b*→ρ*b*

Feltételezzük, hogy $x^* = (x_1^*,..., x_n^*)$ vektor az *(Ábra 1)-(Ábra 3)* –ban leírt hiperbolikus programozási feladat optimális megoldása. Ekkor:

$$\sum_{j=1}^{n} A_j x_j^* = b \quad és \quad x_j^* \geq 0 \; j = 1,...,n$$

A *b* jobboldali vektor, $b' = \rho b$ vektorral történő helyettesítése után, ahol $\rho > 0$ *t*ekintve az új $x' = \rho x^*$ vektort, nyilvánvaló, hogy ez az x' vektor eleget tesz a feltételeknek, azaz

$$\sum_{j=1}^{n} A_j (p x_j^*) = pb \quad ahol \quad x' = \rho x^* \geq 0$$

Így az x' vektor lehetséges megoldása a következő hiperbolikus programozási feladatnak:

$$Q(x) = \frac{P(x)}{D(x)} = \frac{\sum_{j=1}^{n} p_j x_j + p_0}{\sum_{j=1}^{n} d_j x_j + d_0} \rightarrow \max \qquad \textit{(Ábra 4)}$$

Feltételek:

$$\sum_{j=1}^{n} a_{ij} x_j = \rho b_i \quad i = 1,...,m \qquad \textit{(Ábra 5)}$$

$$x_j \geq 0 \quad j = 1,...,n \qquad \textit{(Ábra 6)}$$

Ellenőrizni kell, hogy ez az x' vektor optimális megoldása-e az *(Ábra 4)- (Ábra 6)* –ban leírt problémának. Miután az x^* vektor optimális megoldása az eredeti *(Ábra 1)- (Ábra 3)* problémának, a következő deltákat kapjuk:

$$\Delta_j (x^*) \geq 0, \quad j = 1,...,n$$

Ahol

$$\Delta_j (x^*) = D(x^*)\Delta_j^{'} - P(x^*)\Delta_j^{''}, \qquad j = 1,2,...,n,$$

$$\Delta_j^{'} = \sum_{j=1}^{n} p_{si} x_{ij} - p_j, \qquad j = 1,2,...,n,$$

$$\Delta_j^{''} = \sum_{j=1}^{n} d_{si} x_{ij} - d_j, \qquad j = 1,2,...,n,$$

Az x_{ij} tagok a rendszer által definiáltak, a

$$\sum_{j=1}^{n} A_{si} x_{ij} = A_j, \qquad j = 1,2,...,n.$$

egyenletrendszerből adódóan, ahol az $A_{s1}, A_{s2}, ..., A_{sm}$ oszlopvektorok alkotják az optimális megoldás bázisát alkotó vektorokat. A_j az $A=||a_{ij}||_{mxn}$ mátrix oszlopvektorait jelöli

$$A_j = (a_{1j}, a_{2j}, ..., a_{mj})^T, \qquad j = 1,2,...,n.$$

Észrevehető, hogy a redukált költségek Δ'_j *és* Δ''_j nem függnek a jobboldali *b* vektortól, így a $b \rightarrow \rho b$ behelyettesítés a Δ'_j *és* Δ''_j értékeket nem befolyásolja. De a *(Ábra 4)- (Ábra 6)* –ban leírt probléma célfüggvénye már függ a jobboldali vektortól, ezért figyelembe kell venni a redukált költségeket $\Delta_j(x')$ *ahol* $x' = \rho x^*$. A következő eredményt kapjuk:

$$\begin{aligned}
\Delta_j(px^*) &= D(px^*)\Delta'_j - P(px^*)\Delta''_j = \\
&= \left(\sum_{j=1}^{n} d_j(px_j^*) + d'_0\right)\Delta'_j - \left(\sum_{j=1}^{n} p_j(px_j^*) + p'_0\right)\Delta''_j = \\
&= \left(\sum_{j=1}^{n} d_j(px_j^*) + d'_0 + pd_0 - pd_0\right)\Delta'_j - \left(\sum_{j=1}^{n} p_j(px_j^*) + p'_0 + pp_0 - pp_0\right)\Delta''_j = \\
&= pD(x^*)\Delta'_j + (d'_0 - pd_0)\Delta'_j - pP(x^*)\Delta''_j - (p'_0 - pp_0)\Delta''_j = \\
&= p\Delta_j(x^*) + (d'_0 - pd_0)\Delta'_j - (p'_0 - pp_0)\Delta''_j = \\
&= p\Delta_j(x^*) - G_{j,}
\end{aligned}$$

ahol

$$G_j = \begin{vmatrix} p'_0 - pp_0 & \Delta'_j \\ d'_0 - pd_0 & \Delta''_j \end{vmatrix}.$$

Ez azt jelenti, hogy ha p'_0 *és* d'_0 olyanok, hogy $\rho\Delta_j(x^*) - G_j \geq 0, \quad j = 1,...,n$, vagy különösen, ha $p'_0 = pp_0$ *és* $d'_0 = pd_0$, akkor $\Delta_j(\rho x^*) = \rho\Delta_j(x^*) \geq 0, \quad \forall j = 1,2,...,n$. És ezért az x' vektor a *(Ábra 4)- (Ábra 6)* –ban leírt feladatnak optimális megoldása.

Tehát, ha helyettesítjük a jobboldali *b* vektort egy $b' = \rho b$ vektorral, akkor egyidejűleg ki kell cserélni a p_0 és d_0 tényezőket az eredeti célfüggvényben, mégpedig $p'_0 = \rho p_0$ *és* $d'_0 = \rho d_0$. Ez a két helyettesítés fogja garantálni az ekvivalenciát az eredeti probléma *(Ábra 1)- (Ábra 3)* és az új, skálázott probléma *(Ábra 4)- (Ábra 6)* között. Nyilvánvaló,

hogy ha az x' vektor optimális megoldása az új, skálázott problémának , akkor $x^* = x'/\rho$ optimális megoldása az eredeti problémának.

V.2. Az oszlopvektorok átalakítása. $A_j \rightarrow \rho A_j$

Feltételezzük, hogy $x^* = (x_1^*,...,x_n^*)$ vektor az *(Ábra 1)-(Ábra 3)* –ban leírt hiperbolikus programozási feladat optimális megoldása. Ekkor:

$$\sum_{j=1}^{n} A_j x_j^* = b \quad és \quad x_j^* \geq 0 \; j = 1,...,n \;,$$

és $A_{s1}, A_{s2},..., A_{sm}$ oszlopvektorok alkotják az optimális megoldás bázisát. Ha kicserélünk néhány A_r vektort $r \in J = \{1,2,...,n\}$ egy másik $A_r' = \rho A_r$, $\rho > 0$ vektorral. Nyilvánvaló, hogy az új $x' = (x_1^*, x_2^*,..., x_{r-1}^*, \frac{x_r^*}{\rho}, x_{r+1}^*,..., x_n^*)$ vektor ki fogja elégíteni az újonnan kialakult feltételrendszert:

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

$$x_j^{'} \geq 0, \quad j = 1,...,n$$

Ezért az x' vektor lehetséges megoldása az új skálázott problémának, ami a következő:

$$Q(x) = \frac{P'(x)}{D'(x)} = \frac{\sum_{\substack{j=1 \\ j\neq r}}^{n} p_j x_j + p_r^{'} x_r + p_0}{\sum_{\substack{j=1 \\ j\neq r}}^{n} d_j x_j + d_r^{'} x_r + d_0} \rightarrow \max \qquad \textit{(Ábra 7)}$$

Az alábbi feltételekkel:

$$\sum_{\substack{j=1 \\ j\neq r}}^{n} A_j x_j + A_r^{'} x_r = b, \qquad \textit{(Ábra 8)}$$

$$x_j \geq 0, \qquad j = 1,2,...,n. \qquad \textit{(Ábra 9)}$$

Bizonyítható, hogy az x' vektor optimális megoldása az *(Ábra 7)-(Ábra 9)* –ban leírt új, skálázott problémának.

Miután az x^* vektor optimális megoldása a *(Ábra 1)-(Ábra 3)*–ban leírt eredeti problémának, a következőket kapjuk:

$$\Delta_j(x^*) = D(x^*)\Delta'_j - P(x^*)\Delta''_j \geq 0, \qquad j = 1,...,n$$

Első lépésben feltételezzük, hogy a helyettesítendő A_r vektor egy bázisban szereplő vektor, azaz $r \in J_B = \{s_1, s_2, ..., s_m\}$. Ebben az esetben az új, skálázott problémáról a következő megállapításokat tehetjük:

$$\begin{aligned}
\Delta_j(x') &= D'(x')\Delta'_j - P'(x')\Delta''_j = \\
&= \left(\sum_{\substack{j=1 \\ j\neq r}}^{n} d_j x_j^* + d_r' \frac{x_r^*}{\rho} + d_0 \right) \cdot \left(\sum_{\substack{i=1 \\ s_i \neq r}}^{m} p_{s_i} x_{ij} + p_r' \frac{x_{rj}}{\rho} - p_j \right) - \\
&- \left(\sum_{\substack{j=1 \\ j\neq r}}^{n} p_j x_j^* + p_r' \frac{x_r^*}{\rho} + p_0 \right) \cdot \left(\sum_{\substack{i=1 \\ s_i \neq r}}^{m} d_{s_i} x_{ij} + d_r' \frac{x_{rj}}{\rho} - d_j \right) = \\
&= \left(\sum_{\substack{j=1 \\ j\neq r}}^{n} d_j x_j^* + d_r' \frac{x_r^*}{\rho} + d_0 + d_r x_r^* - d_r x_r^* \right) \cdot \left(\sum_{\substack{i=1 \\ s_i \neq r}}^{m} p_{s_i} x_{ij} + p_r' \frac{x_{rj}}{\rho} - p_j + p_r x_{rj} - p_r x_{rj} \right) - \\
&- \left(\sum_{\substack{j=1 \\ j\neq r}}^{n} p_j x_j^* + p_r' \frac{x_r^*}{\rho} + p_0 + p_r x_r^* - p_r x_r^* \right) \cdot \left(\sum_{\substack{i=1 \\ s_i \neq r}}^{m} d_{s_i} x_{ij} + d_r' \frac{x_{rj}}{\rho} - d_j + d_r x_{rj} - d_r x_{rj} \right) = \\
&- \left(\sum_{\substack{j=1 \\ j\neq r}}^{n} p_j x_j^* + p_r' \frac{x_r^*}{\rho} + p_0 + p_r x_r^* - p_r x_r^* \right) \cdot \left(\sum_{\substack{i=1 \\ s_i \neq r}}^{m} d_{s_i} x_{ij} + d_r' \frac{x_{rj}}{\rho} - d_j + d_r x_{rj} - d_r x_{rj} \right) = \\
&= \left(D(x^*) - d_r x_r^* + d_r' \frac{x_r^*}{\rho} \right) \cdot \left(\Delta'_j + p_r' \frac{x_{rj}}{\rho} - p_r x_{rj} \right) - \\
&- \left(P(x^*) - p_r x_r^* + p_r' \frac{x_r^*}{\rho} \right) \cdot \left(\Delta''_j + d_r' \frac{x_{rj}}{\rho} - d_r x_{rj} \right)
\end{aligned}$$

A levezetés utolsó sorából látszik, hogy ha $p'_r = p_r \rho$ *és* $d'_r = d_r \rho$, akkor $\Delta_j(x') = \Delta_j(x^*) \geq 0, \qquad j = 1,2,...,n.$ Ez azt fogja jelenteni, hogy az x' vektor a *(Ábra 7)* - *(Ábra 9)* –ben leírt skálázott feladatnak optimális megoldása.

Tehát ha helyettesítünk, egy A_r bázisvektort egy $A'_r = \rho A_r, \rho > 0$ vektorral, és egyidejűleg cseréljük a p_r és d_r együtthatókat az eredeti *Q(x)* célfüggvényben a $p'_r = \rho p_r$ *és* $d'_r = \rho d_r$ értékekkel, akkor ezen két helyettesítés garantálja, az ekvivalenciát a két *(Ábra 1)-(Ábra 3) és (Ábra 7)-(Ábra 9)* programozási feladat között. Ha x' optimális megoldása az új, skálázott feladatnak, akkor az $x^* = (x'_1, x'_2, ..., \rho x'_r, x'_{r+1}, ..., x'_n)$ vektor optimális megoldása lesz az eredeti feladatnak.

Következő lépésben tekintsük azt az esetet, ha a helyettesítendő A_r vektor nem bázisbeli vektor, azaz $r \in J_N = J \setminus J_B$. Hasonlóan az előző esethez, az A_r vektor cseréjével párhuzamosan cseréljük a p_r és d_r együtthatókat az eredeti *Q(x)* célfüggvényben a $p'_r = \rho p_r$ *és* $d'_r = \rho d_r$ értékekkel. Mivel $r \in J_N = J \setminus J_B$, azaz *r* nem bázisindex, így $x^*_r = 0$, ezért igaz, hogy $x' = x^*$, $P'(x') = P(x^*)$, $D'(x') = D(x^*)$ *és ezért* $Q'(x') = Q(x^*)$. Tehát az $A_r \to \rho A_r$, $r \in J_N$ helyettesítés csak az Δ'_r, Δ''_r, *és* $\Delta_r(x')$ értékeket befolyásolja. Az A_r nem bázis vektorra kapjuk, hogy:

$$\sum_{i=1}^{m} A_{si} x_{ir} = A_r, \qquad j = 1,2,...,n,$$

Majd az $A_r \to A'_r$ csere után, ahol $A'_r = \rho A_r$ a következő előállítását kapjuk A_r vektornak, ugyanabban a bázisban:

$$\sum_{i=1}^{m} A_{si} (\rho x_{ir}) = \rho A_r, \qquad j = 1,2,...,n.$$

Ha az $A_r \to \rho A_r$ cserével egy-időben cseréljük a célfüggvényben a megfelelő együtthatókat $p_r \to p'_r$, *ahol* $p'_r = \rho p_r$ *és* $d_r \to d'_r$, *ahol* $d'_r = \rho d_r$, akkor a deltákra azt kapjuk, hogy:

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

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

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

$$= D(x^*)(\rho \Delta'_r) - P(x^*)(\rho \Delta''_r) = \rho \Delta_r(x^*) \geq 0$$

Ami azt jelenti, hogy x' vektor optimális megoldása a skálázott problémának.

Tehát, ha cseréljük A_r nem bázis vektort egy $A_r^{'} = \rho A_r, \rho > 0$ vektorral, és egy-időben cseréljük az eredeti *Q(x)* célfüggvény d_r és p_r együtthatóit a $p_r' = \rho p_r \; és \; d_r' = \rho d_r$ értékekkel, akkor biztosítjuk az ekvivalenciát a két feladat között és $x^* = x'$ garantáltan.

V.3. Sorok cseréje. $a_j \rightarrow \rho a_j$

Ebben a fejezetben az *(Ábra 1)-(Ábra 3)*–ban leírt feladat $A=||a_{ij}||_{mxn}$ mátrix $a_r = (a_{r1}, a_{r2}, ..., a_{rn})$ sorvektorának, egy $a_r^{'} = \rho a_r$ vektorral való helyettesítéséről lesz szó. Ha így járunk el, két esetet kell megkülönböztetnünk:

- Ha az $a_r \rightarrow \rho a_r$ helyettesítéssel párhuzamosan kicseréljük a jobboldali *b* vektor *r*-edik elemét $b_r \rightarrow b_r^{'} = \rho b_r$ értékkel.
- Nem végzünk módosítást a jobboldali *b* vektorban.

1.) Az első esetben a következőt kapjuk:

Az *r.* sorban felírt $\sum_{j=1}^{n} a_{rj} x_j = b_r$, feltétel helyett: $\sum_{j=1}^{n} (\rho a_{rj}) x_j = (\rho b_r)$ alakot kapjuk. Az ilyen átalakítás nem befolyásolja az *S* lehetséges halmaz alakulását, így az eredeti feladat teljesen ekvivalens az így kapott feladattal.

2.) A második esetben a következőket kapjuk:

Az ilyen skálázás kiszámíthatatlan deformációhoz vezethet az *S* lehetséges halmazban, így nem adhatunk semmiféle garanciát arra, hogy a skálázott probléma optimális bázisa ugyanaz lesz, mint az eredetié.

Tehát, az egyetlen elfogadható módszer az *A* mátrix sorainak skálázására a következő:

$$\tilde{a}_r \rightarrow \tilde{a}_r^{'}$$

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

Nyilvánvaló, hogy az eredeti és az ily-módon skálázott feladat optimális megoldásai megegyeznek, ezért nem lesz szükség re-skálázásra, az eredeti feladat optimális megoldásának elnyeréséhez.

V.4. Számlálóvektor skálázása. $p \rightarrow \rho p$

Cseréljük ki a *Q(x)* célfüggvény *P(x)* számlálójában a $p = (p_0, p_1, ..., p_n)$ vektort, valamilyen más $p' = (p_0', p_1', ..., p_n')$ vektorral, ahol $p_j' = \rho p_j$, $j = 0,1,2,...,n.$ Meg kell jegyeznünk, hogy a $p \rightarrow \rho p$ helyettesítés nem vezet változásokhoz az optimális bázisban vagy az x^* optimális megoldásban. Így ha megoldottuk a skálázott hiperbolikus problémát, akkor a kapott optimális megoldás ”re-skálázása” céljából a következő formulát kell alkalmaznunk:

$$Q(x^*) = \frac{1}{\rho} Q'(x^*),$$

mert a skálázott feladat optimális megoldás x' megegyezik az eredeti feladat x^* optimális megoldásával.

V.5. Nevező vektor $d \rightarrow \rho d$

Cseréljük ki a *Q(x)* célfüggvény *D(x)* nevezőjében a $d = (d_0, d_1, \ldots, d_n)$ vektort, valamilyen másik $d' = (d_0', d_1', ..., d_n')$ vektorra, ahol $d_j' = \rho d_j$, $j = 0,1,...,n.$ Ez a helyettesítés nem vezet semmiféle változásokhoz az optimális bázisban vagy az x^* optimális megoldást tekintve. Tehát ha megoldottuk a skálázott hiperbolikus problémát, a kapott optimális megoldás ”re-skálázása” céljából a következő formulát kell alkalmaznunk:

$$Q(x^*) = \rho Q'(x^*),$$

mert a skálázott feladat x' optimális megoldás megegyezik az eredeti feladat x^* optimális megoldásával.

VI. Konkrét skálázási eljárások

Ebben a fejezetben bemutatom a már meglévő és bevált skálázási módszereket, valamint az általam kidolgozott minimum-maximum eljárást. Az itt felsorolásra kerülő módszerek tesztelését végeztem el, különböző szempontok alapján. Első lépésben az eljárások lényegi bemutatásáról lesz szó, majd a konkrét teszteredményekről, és következtetésekről, ajánlásokról.

VI.1. Geometriai módszer

Ezt a skálázási eljárást a skóciai Edinburgh-i egyetem kutatója Dr. Julian Hall dolgozta ki és implementálta. Az eljárást a lineáris programozási feladatok rosszul skálázott, ritka mátrixaira alkalmazták. Ez szabály a következő ρ^r oszlop-vektort definiálja a sorok skálázási faktoraként

$$\rho^r = (\rho_1^r, \rho_2^r, ..., \rho_m^r)^T,$$

Ahol

$$\rho_i^r = \left(\prod_{j \in J_i^+} a_{ij} \right)^{1/K_i^r} \qquad i = 1,2,...,m;$$

$J_i^+ = \{ j : a_{ij} \neq 0 \}$, $i = 1,2,...,m$. Ez egy sorokhoz kapcsolódó halmaza, az a_{ij} nem nulla elemek j indexeinek. K_i^r az i. sor nem nulla elemeinek darabszáma.

Analóg módon, az oszlopok skálázásához a következő ρ^c sorvektorba szervezett faktorokat használjuk:

$$\rho^c = (\rho_1^c, \rho_2^c, ..., \rho_n^c),$$

Ahol

$$\rho_j^c = \left(\prod_{i \in I_j^+} a_{ij} \right)^{1/K_j^c} \qquad j = 1,2,...,n;$$

$I_j^+ = \{i : a_{ij} \neq 0\}, \quad j = 1,2,...,n$, egy oszlopokra vonatkozó halmaza a *j.* oszlop a_{ij} nem nulla elemek *i* indexeiknek és a K_j^c jelzi a *j.* oszlopban az a_{ij} nem nulla elemek számát.

A mátrix skálázható egymás után többször, viszont ezzel a módszerrel csak úgy lehet ezt megtenni, ha felváltva skálázzuk a sorokat és oszlopokat, mert miután skáláztunk egy sort vagy oszlopot ha ismét kiszámoljuk hozzá a faktorokat azokra egy értéket kapunk.

Ennek bizonyítása:

Tekintve a $(a_1, a_2, \ldots, a_k)$ nem nulla elemekből álló halmazt, amely lehet a mátrix egy sorának vagy oszlopának nem nulla elemei. A nulla elemek a skálázási faktort nem befolyásolják és a skálázás után is nullák maradnak. Ezen elemekhez tartozó faktor a következő:

$$\rho = \sqrt[k]{a_1 \cdot a_2 \cdot \ldots \cdot a_k}$$

A skálázás utáni új elemek:

$$\left(\frac{a_1}{\sqrt[k]{a_1 \cdot a_2 \cdot \ldots \cdot a_k}}, \frac{a_2}{\sqrt[k]{a_1 \cdot a_2 \cdot \ldots \cdot a_k}}, \ldots, \frac{a_k}{\sqrt[k]{a_1 \cdot a_2 \cdot \ldots \cdot a_k}} \right)$$

Ha ezen elemekhez újra faktorokat számolunk, a következőt kapjuk:

$$\rho = \sqrt[k]{\frac{a_1}{\sqrt[k]{a_1 \cdot a_2 \cdot \ldots \cdot a_k}} \cdot \frac{a_2}{\sqrt[k]{a_1 \cdot a_2 \cdot \ldots \cdot a_k}} \cdot \ldots \cdot \frac{a_k}{\sqrt[k]{a_1 \cdot a_2 \cdot \ldots \cdot a_k}}} =$$

$$= \sqrt[k]{\frac{a_1 \cdot a_2 \cdot \ldots \cdot a_k}{\left(\sqrt[k]{a_1 \cdot a_2 \cdot \ldots \cdot a_k}\right)^k}} = \sqrt[k]{\frac{a_1 \cdot a_2 \cdot \ldots \cdot a_k}{a_1 \cdot a_2 \cdot \ldots \cdot a_k}} = 1$$

Ezért a következő fejezetben lévő példában és a tesztelő programban is felváltva alkalmaztam ezt a módszert sorokra és oszlopokra. Az eredményeket tekintve nem volt befolyásoló tényező, hogy a skálázásokat a sorokkal vagy az oszlopokkal kezdtem.

VI.2. Középérték módszer

Szintén az Edinburgh-i egyetem egyik kutatója Prof. Jacek Gondzio dolgozta ki ezt az eljárást, ami egy másik alternatíva a skálázási faktor számítására, az előző szabályhoz hasonlóan szintén ρ^r oszlopvektort definiál a sorok skálázási faktoraihoz.

$$\rho^r = (\rho_1^r, \rho_2^r, ..., \rho_m^r)^T,$$

Ahol az i. sorhoz tartozó skálázó vektor:

$$\rho_i^r = \sqrt{r_i' r_i''}, \quad i = 1,2,...,m;$$

És

$$r_i' = \max_{j:a_{ij}\neq 0} |a_{ij}|, \quad r_i'' = \min_{j:a_{ij}\neq 0} |a_{ij}|, \quad i = 1,2,...,m.$$

Analóg módon a sorok skálázási faktoraihoz, az oszlopokhoz is definiálni kell a következő skálázási faktorokat tartalmazó ρ^c sor vektort:

$$\rho_j^c = \sqrt{c_j' c_j''}, \quad j = 1,2,...,n;$$

$$c_j' = \max_{i:a_{ij}\neq 0} |a_{ij}|, \quad c_j'' = \min_{i:a_{ij}\neq 0} |a_{ij}|, \quad j = 1,2,...,n.$$

Ezt a módszert egymás utáni skálázás során szintén felváltva kell alkalmazni sorokra és oszlopokra ugyanazon okok miatt, mint amelyek a geometriai módszernél felmerültek, vagyis ha tekintjük nem nulla elemek egy $(a_1, \ldots, a_{\max}, \ldots, a_{\min}, \ldots, a_k)$ halmazát, ezekhez tartozó skálázási faktor a következő:

$$\rho = \sqrt{a_{\max} \cdot a_{\min}}$$

Ezzel a faktorral skálázva az alábbi új elemeket kapjuk:

$$\frac{a_1}{\sqrt{a_{\max} \cdot a_{\min}}}, \ldots, \frac{a_{\max}}{\sqrt{a_{\max} \cdot a_{\min}}}, \ldots, \frac{a_{\min}}{\sqrt{a_{\max} \cdot a_{\min}}}, \ldots, \frac{a_k}{\sqrt{a_{\max} \cdot a_{\min}}}$$

Mivel minden elemet ugyanazzal a pozitív számmal osztottunk, az eredeti maximális elem a skálázás után szintén maximális marad ugyanígy a minimális elem is. Ha ezen új elemekhez ismét kiszámoljuk a skálázási faktort, a következő eredményt kapjuk:

$$\rho = \sqrt{\frac{a_{\max}}{\sqrt{a_{\max} \cdot a_{\min}}} \cdot \frac{a_{\min}}{\sqrt{a_{\max} \cdot a_{\min}}}} =$$

$$= \sqrt{\frac{a_{\max} \cdot a_{\min}}{\left(\sqrt{a_{\max} \cdot a_{\min}}\right)^2}} = \sqrt{\frac{a_{\max} \cdot a_{\min}}{a_{\max} \cdot a_{\min}}} = 1$$

Ezért a következő példában és a tesztelő programban ezt a módszert is felváltva alkalmaztam sorok, illetve oszlopok skálázására.

VI.3. Min-Max módszer

Végül ezt az eljárást dolgoztam ki, az előzőeket alapul véve. A cél az elemek intervallumának szűkítése egyszerű, nem költséges módon. A skálázási faktorok egy újabb megadási lehetősége, ami eltér az előzőektől. Egy lépésben csak egy sort vagy oszlopot faktorizálok, ami azért nagy előny, mert egyre nagyobb méret esetén egyre kevesebb a művelet igénye a másik két módszerrel szemben. Ez a mátrixban a minimális vagy maximális abszolút-értékű elem sora illetve oszlopa lehet. A skálázási faktorok a következők:

ρ_c' : a minimális elem oszlopának a skálázási faktora.

ρ_c'' : a maximális elem oszlopának a skálázási faktora.

ρ_r' : a minimális elem sorának a skálázási faktora.

ρ_r'' : a maximális elem sorának a skálázási faktora.

Ezen faktorok értéke:

$$\rho_c' = \frac{\max_i\{|a_{ij'}|\} + \max elem}{2 * \max_i\{|a_{ij'}|\}}$$, ahol j' a minimális elem oszlopindexe.

Egyértelmű, hogy ezzel beszorozva a minimális elem oszlopát, nem képezünk nagyobb abszolút-értékű új elemet, mint az eddigi maximális elem volt, hiszen az oszlop legnagyobb eleme a $\max_i\{|\, a_{ij'} \,|\}$, és ha ezt beszorozzuk a ρ'_c értékkel, akkor a $\frac{\max_i\{|\, a_{ij'} \,|\} + \max elem}{2}$ értéket kapjuk, ami a mátrix maximális és a faktorizálandó oszlop maximális elemének számtani közepe. Ez biztosítja, hogy minden egyes lépésnél kisebb legyen az elemek abszolút értékeinek intervallumának hossza. Viszont minden következő lépésnél a javulás egyre kisebb lesz, így egy értelmes határt kell adnunk, amelyen túl már nem térül meg a műveletek elvégzésére fordított számítógépes igénybevétel a $\sigma(A)$ értékben vett javulásban.

Analóg módon képezzük a többi faktort is:

$$\rho''_c = \frac{\min_i\{|\, a_{ij''} \,|\} + \min elem}{2 * \min_i\{|\, a_{ij''} \,|\}}, \text{ ahol } j'' \text{ a maximális elem oszlopának indexe.}$$

$$\rho'_r = \frac{\max_j\{|\, a_{i'j} \,|\} + \max elem}{2 * \max_j\{|\, a_{i'j} \,|\}}, \text{ ahol az } i' \text{ a minimális elem sorának indexe.}$$

$$\rho''_r = \frac{\min_j\{|\, a_{i''j} \,|\} + \min elem}{2 * \min_j\{|\, a_{i''j} \,|\}}, \text{ ahol az } i'' \text{ a maximális elem sorának indexe.}$$

Ezen faktorok bármelyikét használva, egyértelműen belátható, hogy a skálázás nem képez az eredeti minimális elemnél kisebbet, illetve az eredeti maximális elemnél nagyobbat.

VII. Numerikus példa a vizsgált módszerek működésére

Ebben a fejezetben egy konkrét mátrixon kerülnek végrehajtásra az eljárások, ezzel bemutatva azok működését és szemléltetve az eljárások hatékonyságát az elérhető javulás terén. Az alapmátrix melyen elvégeztem mindhárom módszer lépéseit:

$$A = \begin{pmatrix} 0.0005 & 3.000 & 0.340 & 234.000 & 34.000 \\ 2.0000 & 4.000 & 345.000 & 1234.000 & 234.000 \\ 30000.0000 & 5.000 & 4565643.000 & 34.000 & 234.000 \\ 9.0000 & 6.000 & 0.001 & 567.000 & 4.000 \\ 567.0000 & 7.000 & 234.000 & 24.000 & 234.000 \\ 56.0000 & 8.000 & 345.000 & 0.001 & 3.000 \\ 45000.0000 & 9.000 & 4.000 & 3.000 & 123.000 \end{pmatrix}$$

Erre a mátrixra azt mondhatjuk, hogy rosszul skálázott, mert

$$\max_{i,j\in J_+}\left(\left|a_{ij}\right|\right) = a_{33} = 4565643.000 = 4.565643E+06,$$

$$\min_{i,j\in J_+}\left(\left|a_{ij}\right|\right) = a_{11} = 0.0005 = 5.000000E-04;$$

$$\sigma(A) = \frac{\max_{i,j\in J_+}\left(\left|a_{ij}\right|\right)}{\min_{i,j\in J_+}\left(\left|a_{ij}\right|\right)} = \frac{4.565643E+06}{5.000000E-04} = 9.13E+09$$

VII.1. A középérték módszer lépései az adott példán

Az A, eredeti mátrix soraihoz tartozó skálázási faktorokból álló ρ^r vektor a következő:

$$\rho_1^r = \sqrt{\min_{j;a_{1j}\neq 0}(|a_{1j}|) * \max_{j;a_{1j}\neq 0}(|a_{1j}|)} = \sqrt{0.0005 * 234} = 0.3421$$

$$\rho_2^r = \sqrt{2 * 1234} = 49.6790$$

$$\vdots$$

$$\rho_7^r = \sqrt{3 * 45000} = 367.4235$$

$$\rho^r = (0.3421,\ 49.6790,\ 4777.8881,\ 0.7530,\ 63.0000,\ 0.5874,\ 367.4235)^T.$$

Elvégezve a sor skálázását:

$$A^1 = \begin{pmatrix} 0.0015 & 8.7706 & 0.994 & 684.1052 & 99.3999 \\ 0.0403 & 0.0805 & 6.9446 & 24.8395 & 4.7102 \\ 6.2789 & 0.0010 & 955.5776 & 0.0071 & 0.0490 \\ 11.9523 & 7.9682 & 0.0013 & 752.994 & 5.3121 \\ 9 & 0.1111 & 3.7143 & 0.3810 & 3.7143 \\ 95.3407 & 13.6201 & 587.367 & 0.0017 & 5.1075 \\ 122.4745 & 0.0245 & 0.0109 & 0.0082 & 0.3348 \end{pmatrix}$$

Az így kapott mátrix szigmája:

$$\sigma(A^1) = \frac{9.56E+02}{1.05E-03} = 9.13E+05.$$

Hasonlóan az előzőkhöz, következő lépésben megadom az A^1 oszlopaihoz tartozó faktorokat:

$$\rho^c = (0.4231, 0.1194, 1.1265, 1.1322, 2.2064).$$

Ezzel a faktorokkal kapott A^2 mátrix szigmája a következő:

$$\sigma(A^2) = 7.2E+05.$$

A következő sor skálázási faktorokból álló vektor:

$$\rho^r = (1.4448, 1.4448, 2.3090, 0.8854, 2.6752, 0.8854, 1.4448)^T.$$

Elvégezve a skálázást, a szigmában történő változás:

$$\sigma(A^3) = 5.45E+05.$$

Második oszlopskálázási faktorok:

$$\rho^c = (0.7801, 0.6994, 0.8854, 1.1294, 0.5475).$$

Elvégezve a második oszlopskálázást:

$$\sigma(A^4) = 4.42E+05.$$

Harmadik sorskálázási faktorok:

$$\rho^r = (1.0654, 1.0654, 1.0000, 1.0000, 1.0654, 1.0000, 1.0654)^T.$$

Elvégezve a harmadik sorskálázást:

$$\sigma(A^5) = 4.42E + 05.$$

Harmadik oszlopskálázási faktorok:

$$\rho^c = (0.9688, 1.0000, 1.0000, 1.0000, 0.9688).$$

A többszörös egymást követő oszlop és sor skálázási műveletet végrehajtva, olyan értékekkel rendelkező skálázási faktorokat kapunk, mind a sorokra mind az oszlopokra, melyek az 1-hez vannak közel. Ezért nincs értelme folytatni ezt a folyamatot, mert a további javítása a $\sigma(A)$, az A mátrix rosszul skálázottság mértékének egyre drágább és drágább.

Így, az eredeti A mátrixból indulva, melynek $\sigma(A)$=9.13E + 09, a skálázási módosítás által a következőt kapjuk $\sigma(A)$=4.42E + 05. Ahogy láthatjuk, a terjedelem javítása körülbelül 5=10- 5.

VII.2. A geometriai módszer lépései az adott példán

Az első sorskálázáshoz tartozó faktorok vektora a következő:

$$\begin{aligned} \rho_1^r &= \sqrt[5]{0.0005 \cdot 3 \cdot 0.34 \cdot 234 \cdot 34} = 1.3233 \\ \rho_2^r &= \sqrt[5]{2 \cdot 4 \cdot 345 \cdot 1234 \cdot 234} = 60.2959 \\ &\vdots \\ \rho_7^r &= 56.9257 \end{aligned}$$

$$\rho^r = (1.3233, 60.2959, 1403.6460, 2.6158, 87.7940, 3.4138, 56.9257)^T.$$

Az első sor skálázás után a kapott mátrix:

$$A^1 = \begin{pmatrix} 0.0004 & 2.2671 & 0.2569 & 176.8328 & 25.6937 \\ 0.0332 & 0.0663 & 5.7218 & 20.4657 & 3.8809 \\ 21.3729 & 0.0036 & 3252.7026 & 0.0242 & 0.1667 \\ 3.4406 & 2.2937 & 0.0004 & 216.7583 & 1.5292 \\ 6.4583 & 0.0797 & 2.6653 & 0.2734 & 2.6653 \\ 16.4038 & 2.3434 & 101.0591 & 0.0003 & 0.8788 \\ 790.5043 & 0.1581 & 0.0703 & 0.0527 & 2.1607 \end{pmatrix}$$

Első oszlopskálázási faktorok vektora:

$$\rho^c = (1.8606, 0.2321, 1.6591, 0.6973, 2.0014).$$

Első oszlop skálázás után kapott mátrix szigmája:

$$\sigma(A^2) = 9.56E + 06.$$

Második sorskálázási faktorok vektora:

$$\rho^r = (1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000)^T.$$

Oszlopskálázási faktorok:

$$\rho^c = (1.0000, 1.0000, 1.0000, 1.0000, 1.0000).$$

Mind a sor mind az oszlopskálázási faktorok beálltak egyre, csupán két iteráció után. Ezért nincs értelme folytatni ezt a folyamatot, mert a további javítása az A mátrix, $\sigma(A)$ rosszul skálázottságának ezen szabályt használva már nem lehetséges.

Tehát, az eredeti $\sigma(A)$ = 9.13E + 09 értékkel rendelkező A mátrixszal indulva, a skálázási módosítással a $\sigma(A)$ = 9.65E + 06 értéket kaptuk. Amint láthatjuk, a terjedelem javítása körülbelül 4 = 10 - 6.

VII.3. A Min-Max módszer lépései az adott példán

Ennél a módszer egyszerre nem lehet módosítani az összes sort vagy oszlopot. Egy alkalommal csak egy sort vagy oszlopot lehet skálázni, melyben vagy a maximális vagy a minimális elem található. A váltakozást itt is alkalmazom. A sorrend legyen minimum elem sora, majd oszlopa, utána a maximum elemre ugyanígy.

A minimális elem az $A[0][0] = 0.0005$, a maximális elem a $A[2][2] = 4565643$. A minimális elem sorában, azaz az első sorban a legnagyobb elem a $A[0][3] = 234$. Ezen adatokból kiszámolhatjuk az első sor faktorát.

$$\rho_r' = \frac{\max_j \{| a_{i'j} |\} + \max elem}{2 * \max_j \{| a_{i'j} |\}} = \frac{234 + 4565643}{2 \cdot 234} = 9756.1475$$

A minimum elem sorának skálázása után a mátrix:

$$A^1 = \begin{pmatrix} 4.8781 & 29268.4414 & 3317.0901 & 2282938.5 & 331709 \\ 2.0000 & 4.000 & 345.000 & 1234.000 & 234.000 \\ 30000.0000 & 5.000 & 4565643.000 & 34.000 & 234.000 \\ 9.0000 & 6.000 & 0.001 & 567.000 & 4.000 \\ 567.0000 & 7.000 & 234.000 & 24.000 & 234.000 \\ 56.0000 & 8.000 & 345.000 & 0.001 & 3.000 \\ 45000.0000 & 9.000 & 4.000 & 3.000 & 123.000 \end{pmatrix}$$

Ahol

$$\sigma(A^1) = \frac{4565643}{0.001} = 4.56E + 09 .$$

Ezután következne a minimum elem oszlopának skálázása, ami most az $A[3][3] = 0.001$, de mivel ez az elem egy oszlopban van a maximális elemmel, így a faktor egy lenne, tehát továbblépek és a maximális elem sorát skálázom.

A maximális elem sorának faktora:

$$\rho_r'' = 0.5001$$

Elvégezve a skálázást a kapott mátrix szigmája:

$$\sigma(A^2) = 2.28E + 09$$

A maximális elem oszlopának faktora szintén egy, hiszen a minimális elem is ebben az oszlopban van, így a következő faktor:

$$\rho'_r = 2013.9729$$

Elvégezve a skálázást:

$$\sigma(A^3) = 2.28E + 09$$

Elvégezve még kilenc skálázást a szigmára a következőt kapjuk: 491935.782406

$$\sigma(A^2) = 4.92E + 05$$

Természetesen ez a kilenc skálázás nem olyan költséges, mint a másik két módszer egy iterációja, hisz ott minden sort illetve minden oszlopot kell skálázni egyszerre, itt pedig csak egyet, azaz összesen kilenc sor vagy oszlop skálázása történik meg, míg a másik módszereknél egy lépésnél felmerül majdnem ekkora költség (7 sor vagy 5 oszlop).

Az iterációt lehetne folytatni tovább is, de nem érdemes mert innen már a $\sigma(A)$ -ban vett javulás egyre kisebb mértékű. Fel kell állítani egy határt, ahol megmondjuk, hogy a számolási költség már nem fedezi a vele elérhető javulást.

Tehát, az eredeti $\sigma(A)$ = 9.13E + 09 értékkel rendelkező A mátrixszal indulva, a skálázási módosításokkal a $\sigma(A)$ = 4.92E + 05 értéket kaptam. Amint láthatjuk, a terjedelem javítása körülbelül 5 = 10 - 5.

VIII. A tesztelő program

Egy skálázási eljárás fontosabb tulajdonságai, hogy

- Mennyi ideig tart, amíg előállít egy optimális szigmával rendelkező mátrixot
- El tudja e érni minden esetben ezt a javulást
- Egy iterációval, milyen mértékű javulást ér el
- Egy értékrendbeli változást mennyi idő alatt ér, illetve minden esetben elér-e ekkora javulást.

Ezen tulajdonságok összehasonlítására készítettem egy programot, mely a fenti három eljárást teszteli számos mátrixra. A programot C++ nyelven írtam Borland C++ Builder fejlesztőkörnyezetben, grafikus felhasználó felülettel rendelkezik, a könnyebb kezelhetőség érdekében. A felületen választható ki a számos opció, melyek mindegyikén teszteltem az eljárásokat.

VIII.1. Implementációs kérdések

Az implementáció során néhány trükköt is kellett alkalmazni. A geometriai módszer szembetűnő hátránya, hogy a skálázandó sor vagy oszlop összes elemét összeszorozza. Ez egy nagy méretű mátrix esetén igen nagy szám lehet, ami akár túlcsorduláshoz is vezethet és nagyon sok időt vesz igénybe. A szorzatból végül *k*. gyököt von. Ezen két műveletet kis átalakítással hatékonyabban is lehet alkalmazni, hogy elkerüljük a nagyon nagy számok létrehozását, ezzel a túlcsordulás esélyét. Tekintsük $(a_1, a_2, ..., a_k)$ nem nulla elemek halmazát, melyeket skálázni szeretnénk. A faktor kiszámolása a geometriai módszer szerint a következő: $\rho = \sqrt[k]{a_1 \cdot a_2 \cdot ... \cdot a_k}$. Ennek implementálása, abban a sorrendben, hogy először bejárjuk az elemeket és összeszorozzuk őket, majd ebből a szorzatból gyököt vonunk, igen költséges művelet. Ezért a következő eljárást alkalmaztam: $\rho = \sqrt[k]{a_1} \cdot \sqrt[k]{a_2} \cdot ... \cdot \sqrt[k]{a_k}$. Azaz, első lépésben megszámoljuk a nem nulla elemeket, hogy megtudjuk a *k* értékét, majd az egyes elemek *k.* gyökét szorozzuk össze, ami így kisebb számot eredményez a részlet szorzatokban is. Egy kódrészlet, ami ezzel a módszerrel számolja ki a mátrix z. sorához tartozó faktort:

```
for(j=0; j<col; j++)
{
        if(fabs(matrix[z][j])!=0)
        num++;
}
num=1/num;

for(j=0; j<col; j++)
{
        if(fabs(matrix[z][j])!=0)
        a=a*pow(fabs(matrix[z][j]),num);
}
return a;
```

Mint már láttuk, a geometriai és a középérték módszer hajlamos arra, hogy egy idő után a faktorok megközelítik az egy értéket majd el is érik azt. Ez a folyamat egyes teszteseteknél hamarabb, máskor később jelentkezik. Ezért szükséges egy vizsgálat, amely megnézi, hogy a kiszámolt faktorok milyen közel vannak az egyhez. És ha túl közel, akkor nem végzi el a mátrixon az átalakítást, hanem véget ér a teszt.

A minimum-maximum módszernél is szükséges egy leállási feltétel. Meg kell vizsgálni, hogy az ebben a lépésben elért javulás mértéke mekkora. Valamint egy maximális iterációszámot is beépítettem, melyen tovább már nem dolgozik a módszer.

A programban a teszteléshez használt mátrixok véletlen számokból épülnek fel. A mátrixok előállításánál, ügyelni kell arra, hogy a beállított sűrűségnek megfelelő legyen benne a nulla elemek száma, illetve ne forduljon elő csupa nulla sor, illetve oszlop. A csupa nulla sor egy értelmetlen feltételt jelentene $0 \cdot x_1 + 0 \cdot x_2 + \ldots + 0 \cdot x_m = 0$. A csupa nulla oszlop pedig egy változó feleslegességére utal.

$$
\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \ldots + 0 \cdot x_i + \ldots + a_{1m}x_m = b_1 \\
&a_{21}x_1 + a_{22}x_2 + \ldots + 0 \cdot x_i + \ldots + a_{2m}x_m = b_2 \\
&\vdots \\
&a_{m1}x_1 + a_{m2}x_2 + \ldots + 0 \cdot x_i + \ldots + a_{mm}x_m = b_m
\end{aligned}
$$

A százaléknak megfelelő nulla elem számot az alábbi ciklus végzi el.

```
for(p=1; p<=(m*m*sprfrom)/100 && p<=m*m-m; p++)
{
        i=random(m);
        j=random(m);
        while(!(matrix[i][j]!=0 && ismorenotnull(matrix,i,j,m,m)))
        {
                matrix[i][j]=0;
                i=random(m);
                j=random(m);
        }
}
```

A p jelöli a ciklus lefutása után a nem nulla elemek számát a mátrixban. Ha nem szeretnénk egy nulla elemet sem, azaz a sűrűség beállításnál 0% szerepelt (ezt az sprfrom változó jelöli), akkor nem fut le egyszer sem a ciklus. A mátrixban elhelyezhető nulla elemek száma korlátos, mert minden sorban és oszlopban kell, hogy szerepeljen nullától különböző elem. A mátrix elemszáma négyzetes mátrixokról lévén szó m*m. Ahhoz, hogy a mátrixszal értelmes feltételrendszert képezhessünk maximum m*m-m darab nulla elemet helyezhetünk el benne. Ha a felhasználói felületen a sűrűség beállításnál ennél több nulla elemet kérünk, a program nem veszi figyelembe. Ahhoz, hogy a maximálisan elhelyezhető nulla elemek száma (m*m-m) az össz elemszámnak legalább 80 százaléka legyen a mátrixnak legalább 5*5-nek kell lennie.

$$\frac{m \cdot m - m}{m \cdot m} \geq 0.8 \quad \Rightarrow \quad \frac{m \cdot (m-1)}{m \cdot m} \geq 0.8 \quad \Rightarrow \quad \frac{m-1}{m} \geq 0.8 \quad \Rightarrow \quad 0.2m \geq 1$$

$$m \geq 5$$

A tesztelés során, mivel 10*10 mátrixoktól indultam, minden mátrix esetén elérhető volt a 80%-os nulla elemszám, ami felső határnak be volt állítva.

Az *ismorenotnull* függvény végzi az ellenőrzést, miszerint a véletlenül kiválasztott sorban (*i* változó által jelölt koordináta) és oszlopban (a *j* változó által jelölt koordináta) van nullán kívüli elem és az nem az *(i,j)* helyre esik. Ha ez igaz, akkor az *(i,j)* koordinátájú helyre nulla elemet lehet helyezni. Ezen függvény működését az alábbi kódrészlet írja le:

```
int ismorenotnull(float** matrix, int z, int x, int row, int col)
{
        int i,j;
        int forrow=0,forcol=0;
//in z row
        for(j=0;  j<col; j++)
```

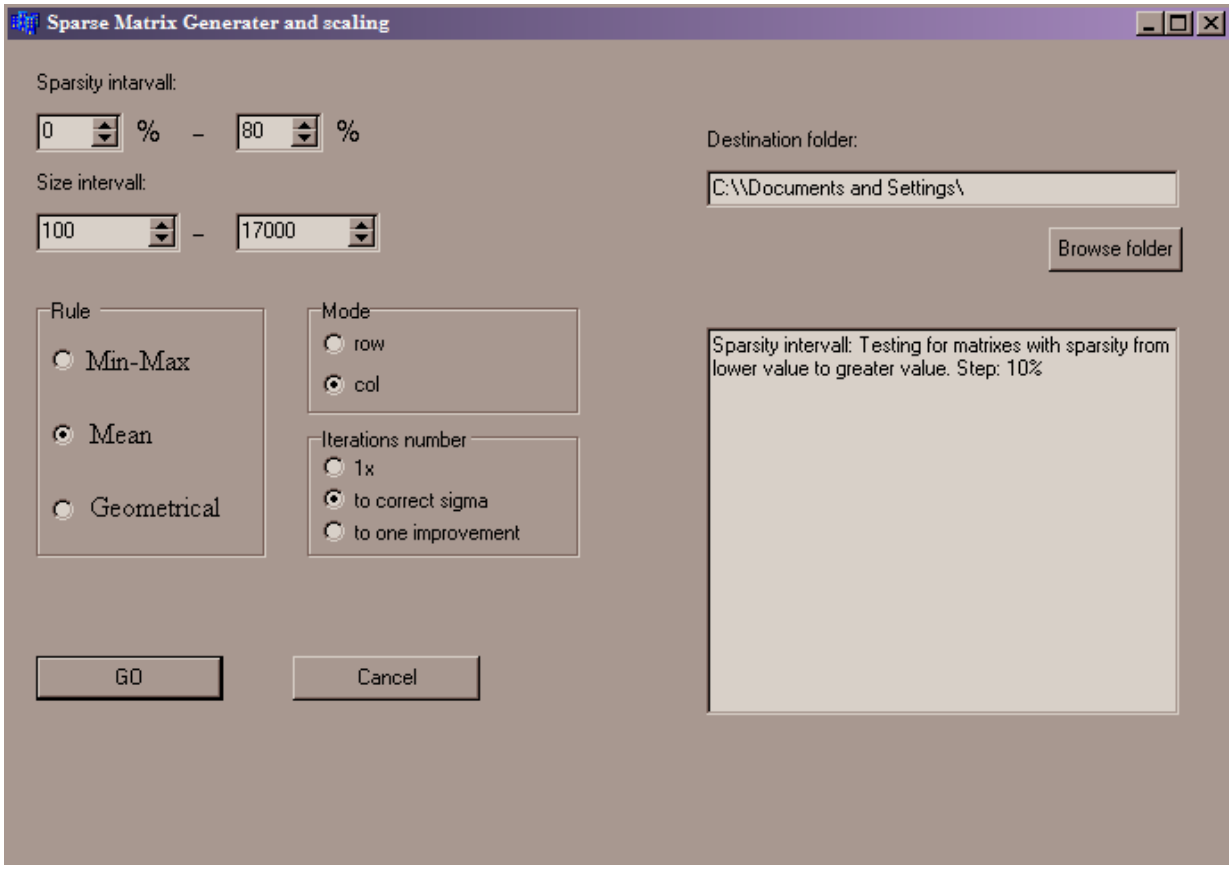
```
        {
                if(j!=x && matrix[z][j]!=0)
                        forrow=1;
        }
// in col x
        for(i=0; i<row; i++)
        {
                if(i!=z && matrix[i][x]!=0)
                        forcol=1;
        }

return forrow && forcol;
}
```

VIII.2. A felhasználói felület

Az alábbi részben bemutatom a felhasználói felületet, melynek komponenseit és azok használatát részletesen ismertetem.

A felhasználói felület:



A programban ki kell választani a tesztelni kívánt eljárást. Be kell állítani egy méret, illetve sűrűség intervallumot. A méretintervallumban megadott alsó érték egy elemszámot fog jelenteni, a program ilyen elemszámhoz legközelebb eső négyzetes mátrixot generál, majd lépésenként növeli a méretet amíg az meg nem haladja a felső határban megadott értéket. A „sparsity intervall" alatt a sűrűség intervallum, azaz a mátrixban szereplő nulla elemek száma adható meg, mely a megadott alsó határtól kezdve 10 százalékos lépésközzel halad a felső határ felé. Egy-egy ilyen százalékos értékkel minden méretben generál a program mátrixot. Így végül a megadott méret és nulla elem szám intervallum között minden méretű és minden sűrűségű mátrixra el fogja végezni a tesztelést.

Ezen kívül a tesztelésnek különböző módjai vannak:

- tesztelhetünk egy előre megadott szigma érték eléréséig, illetve, ha erre nem képes az eljárás, akkor egy egyéb leállási feltételig (*to correct sigma*)
- vizsgálhatjuk, azt az esetet, amikor csupán egy iterációt hajtunk végre (*1x*)
- valamint egy nagyságrend javulásig engedjük iterálni a módszert. (*to one improvement*)

Az előre megadott optimális szigma alatt a 10^3 érték alatti számokat értjük, melyet egy nevesített konstansban tárolunk, ezzel megkönnyítve annak módosítását.

A középérték és geometriai módszer esetén megválaszthatjuk, hogy a módszer a sorok vagy az oszlopok skálázásával kezdje az iterációt.

VIII.3. A program outputja

Végül az outputról néhány szó. A program egy *.txt* fájlba menti el az eredményeket, mely importálható Excel dokumentummá. A felületen megadható a mappának az elérési útvonala, ahova az eredmény fájl kerüljön. Ha már végeztünk előzőleg teszteléseket, akkor a már létező fájlt kiegészíti az újabb eredményekkel. Egy eredmény fájl Excel programmal megnyitva:

	A	B	C	D	E	F	G	H	I
1	Testing of the method Min-Max :								
2	Sparsity	Probl size	Orig. sigma	ItType	Result sigma	Improvment	It number	Time	
3	0%	100	2.59e+08	to correct szigma	1.11e+06	2.32e+02	300	0.016	
4	0%	121	1.71e+08	to correct szigma	5.62e+06	3.04e+01	300	0.016	
5	0%	144	7.00e+08	to correct szigma	5.83e+06	1.20e+02	300	0.015	
6	0%	169	5.35e+08	to correct szigma	2.00e+07	2.67e+01	300	0.016	
7	0%	196	6.94e+09	to correct szigma	5.91e+07	1.17e+02	300	0.031	
8	10%	100	3.22e+08	to correct szigma	4.51e+06	7.13e+01	300	0.016	
9	10%	121	7.57e+08	to correct szigma	1.19e+07	6.34e+01	300	0.015	
10	10%	144	2.55e+08	to correct szigma	3.69e+06	6.89e+01	300	0.016	
11	10%	169	2.15e+09	to correct szigma	1.33e+07	1.61e+02	300	0.015	
12	10%	196	4.55e+09	to correct szigma	4.20e+07	1.08e+02	300	0.031	
13	20%	100	2.64e+08	to correct szigma	1.10e+07	2.39e+01	300	0.016	
14	20%	121	2.35e+09	to correct szigma	3.73e+07	6.30e+01	300	0.016	
15	20%	144	7.21e+08	to correct szigma	3.95e+07	1.83e+01	300	0.015	
16	20%	169	2.98e+08	to correct szigma	5.28e+06	5.65e+01	300	0.016	
17	20%	196	5.94e+08	to correct szigma	1.47e+07	4.04e+01	300	0.031	
18									
19									
20									

Első sorban olvasható a tesztelt eljárás neve. (*Testing of the method ...*) Első oszlopban a nulla elemek százalékos száma jelenik meg (*Sparsity*), mellette a mátrix mérete (*Probl size*). Az eredeti mátrixról szóló információkat a harmadik oszlop, a kiinduló szigma érték zárja (*Orig. sigma*). Majd a tesztelés módja következik (*It Type*), mely háromféle lehet (*to correct sigma, one iteration, one improvement*). Végül a tesztelés során előállt eredmények következnek. Az elért szigma érték (*result sigma*), az eredeti és az elért közti javulás mértéke (*improvement*), az iteráció száma (*it number*) illetve, hogy mennyi időre volt szükség (*Time*).

Olyan output előállítására is van lehetőség, ahol lépésről lépésre kapunk információt az átalakításokról. Ez akkor célszerű, amikor egy konkrét mátrixon szeretnénk megvizsgálni az adott módszer átalakításait. Az ilyen tesztelések során a teljes kiinduló mátrix kiírásra kerül, valamint annak szigma értéke és a kiszámolt faktorok. Majd ez ismétlődik minden lépésben a leállási feltételig, illetve nagyon nagy mátrixok esetében egy bizonyos fájl méretig, hogy elkerüljük a túl nagy méretű kimenet létrehozását, mellyel már nehezen boldogulnának a szerkesztő szoftverek. A VII. fejezetben használt példában szereplő mátrixot inputnak véve, majd arra a középérték módszert lefuttatva az alábbi eredményt kapjuk.

(Részlet:)

```
     0.0005    3.0000    0.3400  234.0000   34.0000
     2.0000    4.0000  345.0000 1234.0000  234.0000
 30000.0000    5.0000 4565643.0000   34.0000  234.0000
     9.0000    6.0000    0.0010  567.0000    4.0000
   567.0000    7.0000  234.0000   24.0000  234.0000
    56.0000    8.0000  345.0000    0.0010    3.0000
 45000.0000    9.0000    4.0000    3.0000  123.0000

9131285566.287209
============================================================
  0.3421   49.6790 4777.8882   0.7530  63.0000   0.5874  367.4235
============================================================
     0.0015    8.7706    0.9940  684.1052   99.3999
     0.0403    0.0805    6.9446   24.8395    4.7102
     6.2789    0.0010  955.5776    0.0071    0.0490
    11.9523    7.9682    0.0013  752.9940    5.3121
     9.0000    0.1111    3.7143    0.3810    3.7143
    95.3407   13.6201  587.3670    0.0017    5.1075
   122.4745    0.0245    0.0109    0.0082    0.3348

913128.590131
============================================================
  0.4231   0.1194   1.1265   1.1322   2.2064
============================================================
```

Először megtekinthetjük magát a tesztelendő mátrixot, majd a hozzá tartozó szigma értéket írja ki a program. Ezután az elválasztó jelek között szerepelnek az aktuális faktorok. Ezekkel elvégzett átalakítás után kapott mátrix kerül ismét kiírásra és így tovább. Ezen output segítségével szemmel követhető a faktorok alakulása, hogy milyen hamar kezdenek közeledni az egyhez. Azt is megfigyelhetjük, hogy az egyes lépések mennyivel javítják a szigmát, hogy a jelentős javulás megtörténik már a ciklizálás elején, vagy egyenletesen csökken a szigma, esetleg a kezdeti gyenge javulás után ugrásszerűen ér el eredményt.

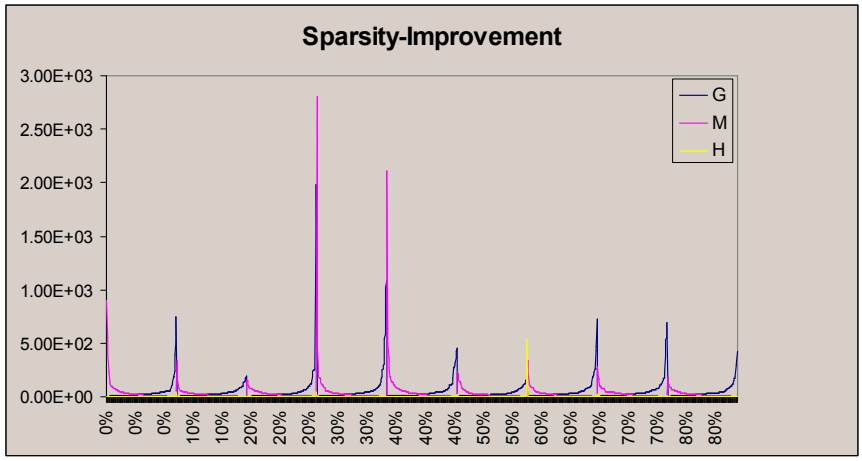
IX. Teszteredmények

IX.1. A korrekt szigma eléréséig történő tesztelés

Ebben a részben azon teszteredmények kerülnek összehasonlításra, melyeknél a megadott $\sigma(A)<10^3$ elérése volt a cél. A legnagyobb mértékű javulást a Min-Max módszer érte el, amikor is az elért $\sigma(A')$, az eredeti $\sigma(A)$ 2810 –ed része lett.

Az alábbi ábra az elért javulások nagyságrendjének a ritkaság szempontjából vett függvénye:

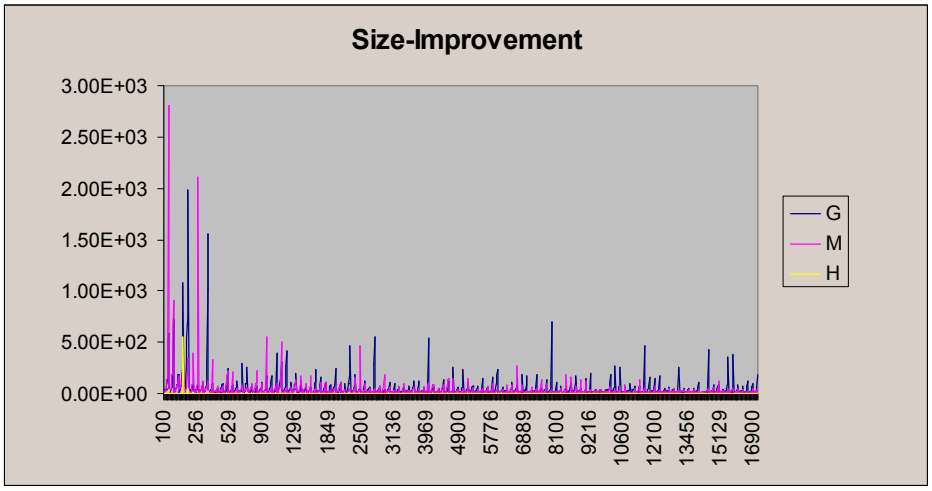
- M – Min-max módszer
- G – Középérték módszer
- H – Geometriai módszer



Grafikon 1.

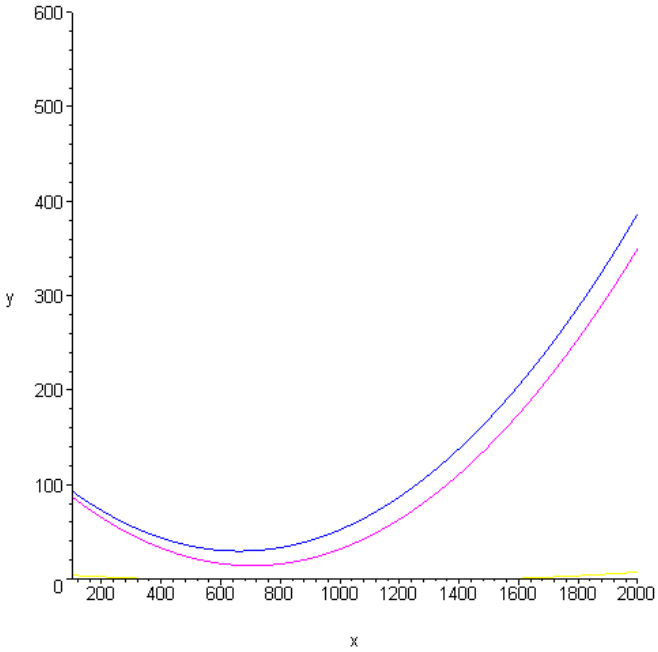
Az ábráról leolvasható, hogy mind a középérték mind a Min-max módszer a közepesen ritka mátrixoknál érnek el jelentősebb javulást, ez kb. 20 és 40 százalék között jelentkezik. A geometriai módszer a másik kettőhöz képest nem ért el jelentős javulást. Valamint az is megfigyelhető ezen az intervallumon, hogy a Min-max módszer görbéje magasabb értékeket ér el.

A következő ábra a méret függvényében vizsgálja az elért javulásokat:



Grafikon 2.

Az ábrán látható, hogy a módszerek a kisebb méretű mátrixoknál érnek el jelentősebb javulás. Itt is megfigyelhető egy erős ingadozás az adott méretű mátrixok között. A kapott adatokra illesztett görbék:

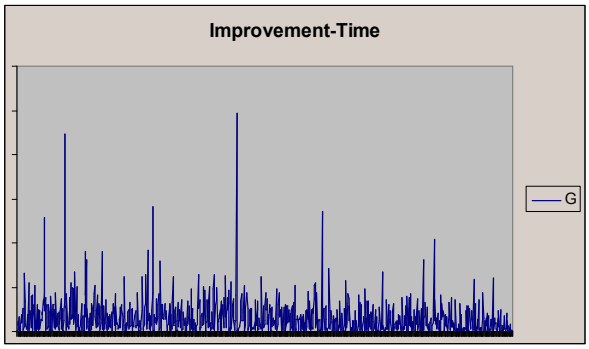


Grafikon 3.

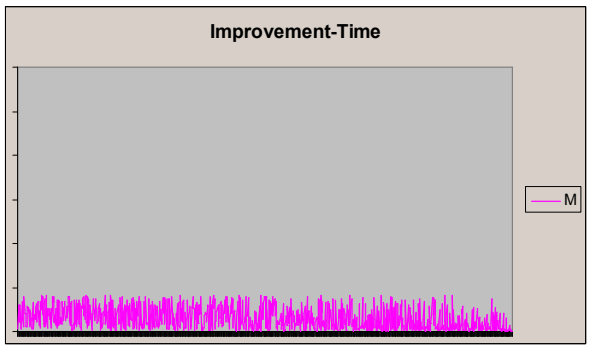
Ami az előző ábrából nem volt kivehető, az itt már tisztán látszik, hogy ezek a görbék parabolák, itt szintén nagy hasonlóságot mutat a G és az M módszer lényeges különbség nem figyelhető meg köztük, a görbe szerint, ha a méretet egy bizonyos érték fölé emeljük, akkor nagyobb javulást képesek elérni. Ez elég meglepő eredmény, hisz azt gondolnánk, hogy ha több az elemszám, akkor azok jobban szét vannak szórva, nehezebb azokat egymáshoz

közelíteni. Ekkora elemszámmal már nem teszteltem a hardver korlátok miatt, szóval ezek csak következtetések.

A következő nagyon lényeges dolog az idő, itt már jelentős különbségeket figyelhetünk meg. Míg a középérték módszer jelentősen ingadozik az időt tekintve, addig a Min-max módszer megbízhatóbb, egyenletesebb tendenciát mutat.



Grafikon 4.



Grafikon 5.

Az idő nagyon lényeges, mert eme eljárásokat a szimplex módszer, illetve más számítási algoritmusok előkészítéseként alkalmazzák. Természetes, hogy mivel maguk a számítási eljárások is elég idő és erőforrás-igényesek, így nem szeretnénk az előkészítéssel sok időt elpazarolni. Ez az a pont ahol az első jelentős különbséget felfedezhetjük a Min-max és a középérték módszer között. Míg a Min-max módszer minden tesztmátrixot közel ugyanolyan rövid idő alatt skálázott, addig a középérték módszer néhány mátrixnál elég sok időt eltöltött és ezeknél esetleg nem is ért el az idővel arányos javulást. Az ábrán a mátrixokkal eltöltött időket az elért javulás sorrendjében jelenítjük meg, azaz x tengely az elért javulás az y pedig hogy ezt mennyi idő alatt érte el.

A geometriai módszer hibái már az első teszteléseknél kiütköztek, miszerint a nagyon sok szorzás és gyökvonás felettébb időigényes műveletek, ami nem lenne baj, ha az iterációszám kevesebb lenne, de nagy iterációszám esetén sem ért el a másik két módszerhez mért eredményeket.

A teszteredmények nem mutattak olyan eredményt, miszerint az elért javulást nagyban befolyásolná az eredeti mátrixon értelmezett szigma érték.

IX.2. Egy nagyságrendbeli javulás eléréséig való iteráció

Ennél a beállításnál addig kell iterálnia az adott módszernek, amíg el nem ér egy nagyságrendbeli csökkenést a $\sigma(A)$-n. Természetesen voltak olyan mátrixok, melyeken nem érte el ezt a megadott értéket. Fontos tényezőnek találtam, hogy a megkapott mátrixok hány százalékánál érte el az adott javulást. Erre a következő eredményeket kaptam:

- Min-max módszer : 67 %
- Középérték módszer : 79 %

A középérték módszer egy bizonyos mértékű javulásnál jobbat nem tud elérni. Tapasztaltam, hogy minden esetben a skálázási faktorok az 1 felé közelítenek, és el is érik azt. Vannak mátrixok, melyeknél viszonylag hamar, viszont vannak olyanok is, ahol csak sok lépés után következik ez be. Az a tapasztalat, hogy ez nem áll összefüggésben a javulás mértékével. Találkoztam olyan tesztesetekkel is, melyekkel elég sokáig iterált a középérték módszer mire a faktorok beálltak az 1 értékre, de mégsem ért el nagy javulást.

Ezzel szemben az Min-max módszer elég sokáig képes úgy iterálni, hogy még ér el javulást, a módszer jellegéből adódik, hogy ez a javulás egyre kisebb. Felmerül a kérdés, hogy akkor meddig érdemes hagyni dolgozni. A teszteléskor maximum 300 iterációt engedtem meg, illetve beépítettem egy olyan vizsgálatot is, mely azt nézi, hogy az adott lépésben mekkora javulás történt a szigma értékében, és ha ez már elhanyagolhatóan kicsi, akkor a program befejezi a tesztelést.

Végeztem tesztelést a két nagyságrendbeli javulás elérésére is, itt még közelebb eset a középérték és a Min-max módszer esetén azon mátrixok száma melyeknél elérték ezt az értéket. Ez százalékokat tekintve jelentősen kevesebb volt, mint az előző.

X. Következtetések és további ötletek

A tesztelések során kiderült, hogy mire kell ügyelni egy skálázási eljárásnál. Az egyik legfontosabb tényező az idő. Fontos, hogy a szükséges idő minden esetben megtérüljön az elért javulásban. Nagy hátránya egy módszernél, ha az előkészítésnél jelentős időt vesz igénybe, és mégsem küszöböli ki a rosszul skálázottságot. Ahhoz hogy egy eljárás processzor

igénye kicsi legyen lényeges az egyszerűség. Ha többnyire elemi műveletekből épül fel, és ezekhez kevés elemet használ fel, akkor meglehetősen kicsi lesz az időigénye. A középérték és a minimum-maximum módszerek előnye itt is megmutatkozik, hiszen csak kettő illetve három mátrixelemet használnak fel a faktor kiszámításához, míg a geometriai módszer a mátrix méretének növekedésével egyre több elemet használ.

A másik nagyon fontos dolog a módszer hatékonysága. Ez magától az algoritmustól függ. Nem lehet receptet adni, arra hogy mitől lesz hatékony egy módszer. A munkám nagy részét tette ki, hogy olyan algoritmust találjak, amely hatékonyság terén felveszi a versenyt az eddigiekkel. A minimum-maximum módszer egy egyszerű eljárás, persze ebben a témában épp az a legnehezebb, hogy egy egyszerű eljárást találjunk, ami hatékony is.

Egy skálázási eljárásnál fontos, hogy könnyen implementálható legyen, így esetlegesen egy gépközeli nyelven megírva további gyorsítást érhetünk el. Például, ha a kiszámolt faktor helyett a hozzá legközelebb eső kettő hatványt használjuk fel, és így magát a skálázást nem egy szorzás vagy osztás segítségével végezzük el, hanem egy jóval kevesebb időt igénylő eltolással. Persze ez valószínű, hogy csak akkor tehető meg, ha a kiszámolt faktor elég közel esik egy kettő hatványhoz, különben elrontaná a módszer hatékonyságát. A skálázás gyakorlati alkalmazásánál fontos, hogy letároljuk a faktorokat. Az ilyen változtatás ezt is megkönnyíti, hiszen elég csak a hatványkitevőt letárolni, ami lényegesen kisebb helyen elfér. Ezt a megtakarítást a végeredmény visszanyerésénél is alkalmazhatjuk, hisz az egymás után alkalmazott faktorok szorzata is kettőnek valamely hatványa lesz és így az eredmény visszanyerése is megoldható egy eltolással.

XI. Tesztelés körülményei

A Borland C++ builder környezetben fejlesztett tesztelő program futtatható változata és forráskódja elérhető a http://delfin.unideb.hu/~ra0022 weboldalon. A tesztelést minden módszer és minden lefutási mód esetén kb. 1100 mátrixon végeztem el, melyek mérete 10x10 –től a 130x130-ig terjedt. Minden méret esetén előállt 9 különböző sűrűségű mátrix, melyekben a nulla elemek száma a 0%-tól a 80%-ig terjedt.

Tesztkörnyezet:

- Operációs rendszer: Microsoft Windows XP Home Edition +SP2
- CPU: Mobile AMD Turion 64 MT-32, 1800 MHz
- Rendszermemória: 512 Mb

XII. Függelék

Az alábbi ábrák tartalmazzák a grafikonok elkészítéséül szolgált teszt adatok egy részét tartalmazzák. A grafikonokat több száz sor adatból készítettem, így ezeknek csak egy részét mutatom itt be.

XII.1. A grafikon 1 elkészítésénél használt adatok:

A program által tesztelt esetek átlagát tekintve a következő eredményeket kaptam a sűrűség és az elért javulást nagyságrendje között.

Középérték módszer		Minimun-Maximum módszer	
Sparsity	Improvement	Sparsity	Improvement
0%	4,24E+01	0%	4,42E+01
10%	3,34E+01	10%	2,33E+01
20%	6,76E+01	20%	2,65E+01
30%	6,62E+01	30%	5,82E+01
40%	4,79E+01	40%	6,56E+01
50%	3,11E+01	50%	2,67E+01
60%	5,15E+01	60%	2,76E+01
70%	5,32E+01	70%	2,86E+01
80%	4,15E+01	80%	2,55E+01

XII.2. A grafikon 2 elkészítésénél használt adatok

A teszt a 10x10 mérettől a 130x130 méretig történt, és minden méret esetén kilenc különböző sűrűségű mátrixra. Ezen adatok egy része:

Középérték módszer		Minimum-maximum módszer	
Probl size	Improvement	Probl size	Improvement
100	1,65E+02	100	4,03E+02
256	3,26E+01	256	2,68E+02
484	5,37E+01	484	7,67E+01
784	2,74E+01	784	6,71E+01
1156	1,02E+02	1156	2,16E+01
1600	5,94E+01	1600	2,21E+01
2116	2,70E+01	2116	1,99E+01
2704	1,07E+02	2704	2,04E+01
3364	2,67E+01	3364	3,12E+01
4096	2,86E+01	4096	3,95E+01
4900	5,01E+01	4900	1,75E+01
5776	3,36E+01	5776	2,15E+01
6724	3,15E+01	6724	2,08E+01
7744	9,86E+01	7744	1,32E+01
8836	2,03E+01	8836	3,35E+01
10000	5,36E+01	10000	2,67E+01
11664	4,14E+01	11664	1,20E+01
13456	2,16E+01	13456	1,58E+01
15376	6,88E+01	15376	1,45E+01
16900	3,49E+01	16900	2,05E+01

XII.3. A grafikon 4 és grafikon 5 elkészítésénél használt adatok

Az eredeti tesztadatok 50-es csoportokban vett átlaga.

Középérték módszer		Minimum-maximum módszer	
Improvement	Time	Improvement	Time
4,21E+00	0,31	3,38E+00	0,36
6,10E+00	0,39	4,72E+00	0,40
7,21E+00	0,42	5,50E+00	0,40
8,49E+00	0,41	6,31E+00	0,39
9,60E+00	0,29	7,27E+00	0,35
1,08E+01	0,37	8,22E+00	0,35
1,22E+01	0,35	9,24E+00	0,36
1,38E+01	0,26	1,03E+01	0,37
1,54E+01	0,38	1,17E+01	0,35
1,68E+01	0,47	1,31E+01	0,35
1,85E+01	0,30	1,50E+01	0,32
2,13E+01	0,30	1,69E+01	0,30
2,45E+01	0,25	1,90E+01	0,30
2,81E+01	0,34	2,15E+01	0,28
3,24E+01	0,21	2,46E+01	0,28
3,70E+01	0,24	2,81E+01	0,21
4,29E+01	0,19	3,34E+01	0,31
5,26E+01	0,27	4,02E+01	0,20
6,58E+01	0,28	4,89E+01	0,23
9,05E+01	0,16	6,71E+01	0,17
1,47E+02	0,22	9,76E+01	0,16
4,85E+02	0,10	3,73E+02	0,10
Legkisebb idő	0,10		0,10
Legnagyobb idő (fenti átlagokban)	0,47		0,40
Legnagyobb idő a teszt során	4,92		0,83

XIII. Irodalom jegyzék

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

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

- Curtis,A.R., Reid,J.K., ”On the Automatic Scaling of Matrices for Gaussian Elimination”,
  J. Inst. Maths. Applics., Vol.10, pp.118-124, 1972.

- Duff,I.S., Erisman,A.M., Reid,J.K., ”Direct Methods for Sparse Matrices”, Clarendon Press,
  Oxford, 1986.

- Pissanetzky,S., ”Sparse Matrix Technology”, Academic Press, 1984.

- Skeel,R.D., ”Scaling for Stability in Gaussian Elimination”, J. Assoc. Comput. Mach., Vol.26, pp.494-526, 1979.

- Maros István, ”Computational Techniques of the Simplex Method”, Kluwer Academic Publishers 2003.

- Nicholas J. Higham, ”Accuracy and stability of numerical Algorithms”, Society for Industrial and Applied Mathematics, 1996

- Wayne L. Winston ”Operációkutatás Módszerek és alkalmazások I., II.”, International Thomson Publishing, Duxbury Press 1994.

XIV. Köszönetnyilvánítás

Sokan vannak, akik tapasztalataik átadásával segítették ezen munka elkészítését. Köszönettel tartozom Dr. Juhász Istvánnak a programozási ismeretek elsajátításáért, Dr. Vertse Tamásnak és Dr. Stoyan Gisbertnek a numerikus módszerek implementálása terén szerzett tapasztalatokért, Dr. Fazekas Attilának a formai és tartalmi kialakításhoz adott tanácsaiért. Külön köszönet témavezetőmnek Dr. Bajalinov Eriknek, aki nagy szerepet játszott abban, hogy nagy érdeklődéssel fordultam az operációkutatás különféle területeinek kutatása felé, és a sok segítségért, amit diplomamunkám elkészítéséhez nyújtott.