3. Lineáris egyenletrendszerek megoldása

2019. február 18.

Lineáris egyenletrendszer

M darab egyenlet N változóval, az a_{ij} és b_j értékek ismertek:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1N}x_N &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2N}x_N &= b_2 \\ &\vdots \\ a_{M1}x_1 + a_{M2}x_2 + \ldots + a_{MN}x_N &= b_M \end{aligned}$$

Felírhatjuk indexes[1] és mátrix alakban is:

$$\begin{aligned} a_{ij} \cdot x_j &= b_i \\ \mathbf{A} \cdot \mathbf{x} &= \mathbf{b} \end{aligned}$$

Cél: az x_i ismeretlenek meghatározása.

[1] azonos indexekre automatikus Σ

Lineáris egyenletrendszer meghatározottsága

$M = N$ esetén

- Jó esély van konkrét megoldás megtalálására, kivéve ha
- van olyan sor, ami más sorok lineárkombinációjaként előáll
  $\longrightarrow$ ekkor a mátrix *sorok szerint degenerált*, vagy
- a mátrix két oszlopa megegyezik[2],
  $\longrightarrow$ ekkor a mátrix *oszlopok szerint degenerált*
- Négyzetes mátrix esetében:
  sorok szerint degenerált $\Leftrightarrow$ oszlopok szerint degenerált
  $\Leftrightarrow$ $det(\mathbf{A}) = 0$
  $\longrightarrow$ ekkor a mátrix *szinguláris*

[2] pontosabban: ha az egyenletrendszer bizonyos változókat csak teljesen azonos lineárkombinációkban tartalmaz

Numerikusan szinguláris mátrixok

A számítógép a valós számokat véges precizitással ábrázolja, emiatt kerekítési hibák adódnak.

Ha két sor nem teljesen azonos, de nagyon hasonlóak ($\epsilon \simeq 10^{-10}$)

- Ekkor a kerekítési hibák miatt előfordulhat 0-val osztás $\longrightarrow$ a mátrix *numerikusan szinguláris*
- Ilyen esetekben a 0-val való osztás miatt hibát kapunk [3].

$N \gg 1 \Rightarrow$ az egyenletrendszer megoldása sok műveletet igényel

- A kerekítési hibák összeadódnak
- A program lefut, de a végeredmény hibás lesz
- Behelyettesítéssel kell ellenőrizni

[3] Újabb C implementációk hiba helyett Infinity vagy NaN értéket használnak.

Numerikusan szinguláris mátrixok kezelése

Az előbbi két probléma kiküszöbölésére léteznek algoritmusok

Speciális algoritmus alkalmazása nélkül

- Néhány 10 változós egyenlet még megoldható
- Néhány 100 egyenlethez duplapontosságú aritmetika kell: double típusú változók használata
- 1000 egyenlet fölött már biztosan jelentkeznek a problémák

Lineáris egyenletrendszerek megoldásának sebessége

A feladat gépigénye:

- a memóriaigény N^2-tel skálázik
- a számításigény N^3-bel skálázik

Egy megoldási lehetőség: a feladat párhuzamosítható

Speciális alakú mátrixok esetében van gyorsabb megoldás.

- tridiagonális (ld. spline interpoláció)
- Vandermonde-típusú (ld. interpolált polinom együtthatói)

Gyakori megoldandó problémák

- Az $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ lineáris egyenletrendszer megoldása
- Az $\mathbf{A} \cdot \mathbf{x}_k = \mathbf{b}_k$ egyenletrendszerek megoldása
  - Egyszerre érdemes a k darab egyenletrendszert megoldani
  - Az $\mathbf{A}$ mátrix invertálása, és $\mathbf{b}_k$ beszorozgatása a kerekítési hibák miatt nem célszerű
- Az $\mathbf{A}$ mátrix inverzének meghatározása
  - ez azonos az előzővel, ha $\mathbf{b}_k$ a triviális bázisvektorok
- Az $\mathbf{A}$ mátrix determinánsának kiszámítása, stb.

Alulhatározott egyenletrendszerek

Ha $M < N$, vagy $M = N$, de az egyenletrendszer degenerált, azaz kevesebb egyenlet van, mint ahány ismeretlen; az egyenletrendszer *alulhatározott*

- Egyáltalán nincsen megoldás, vagy
- A megoldás egy egész altér:
  $\mathbf{x}_p$ egy lehetséges megoldás, és ehhez jön még $N - M$ vektor tetszőleges lineárkombinációja

A megoldást ilyenkor ún. *szinguláris*érték-dekompozícióval keressük (singular value decomposition = SVD)

Túlhatározott egyenletrendszerek

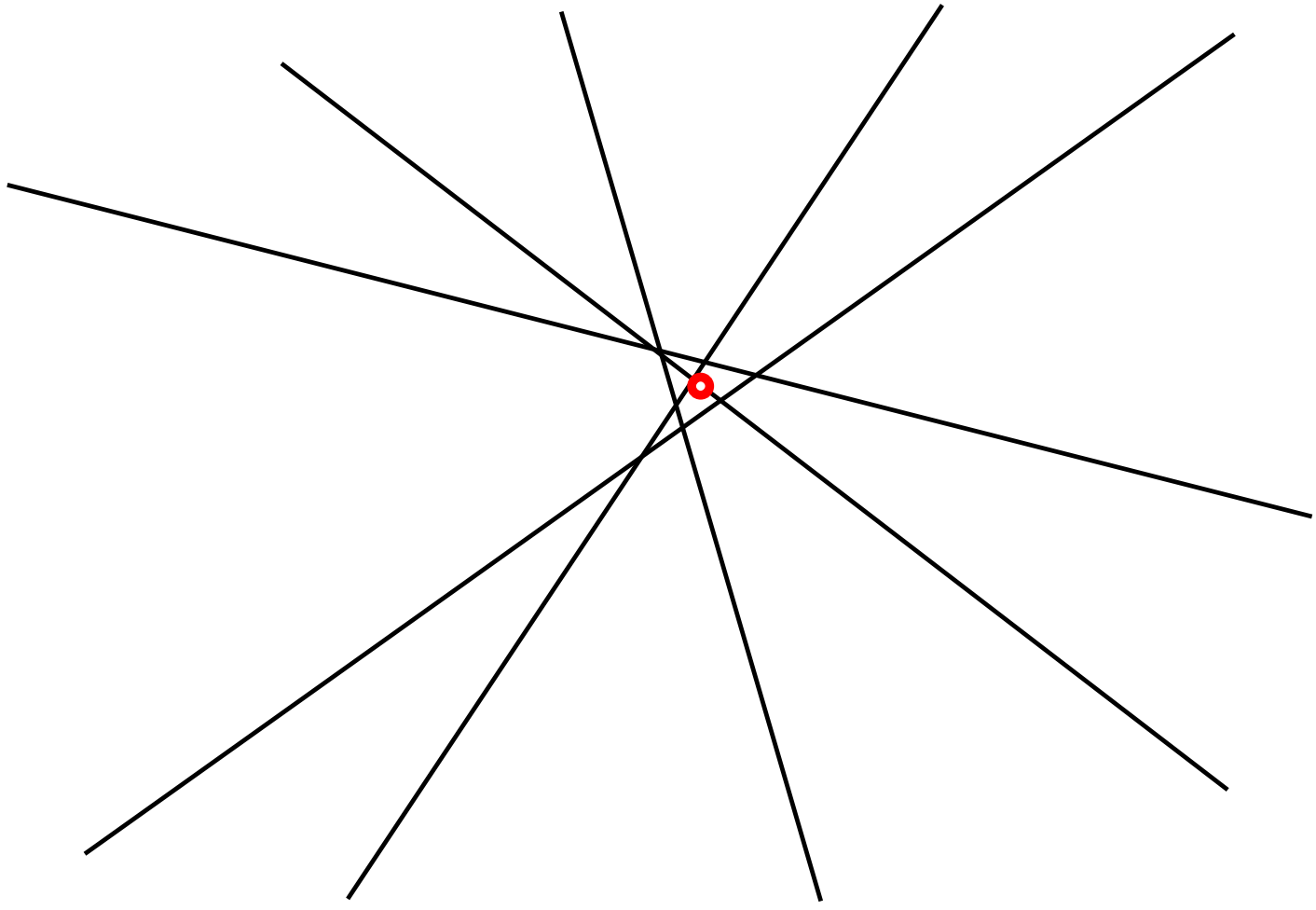
Ha $M > N$, akkor az egyenletrendszer *túlhatározott*

- Általában nincsen megoldás, de
- Van értelme egy lehetséges legjobb megoldásról beszélni.

Például: Mi az a pont, ami a lehető legjobban kielégíti az egyenletrendszert olyan értelemben, hogy az egyenletek jobb és baloldalai közötti eltérések négyzetösszege minimális?

$$\arg\min_{x_j} \sum_i \left(b_i - a_{ij}x_j\right)^2$$

Túlhatározott egyenletrendszer “lehető legjobb” megoldása



Házi feladat

Lássuk be, hogy a szinguláris egyenletek legkisebb négyzetek alapján definiált lehető legjobb megoldása előáll a következő egyenletrendszer megoldásaként:

$$\mathbf{A}^\mathsf{T}\mathbf{A}\,\mathbf{x} = \mathbf{A}^\mathsf{T}\mathbf{b},$$

ahol $\mathbf{A}^\mathsf{T}$ az $\mathbf{A}$ mátrix transzponáltja!

- A fenti egyenletrendszert az eredeti egyenletrendszer *normálegyenleteinek* nevezzük.
- Általában ezeket is SVD-vel célszerű megoldani direkt megoldás helyett.

Programcsomagok lineáris egyenletek megoldására

Nagyon általános probléma

- Sok ember dolgozik rajta
- Optimalizált programcsomagok léteznek

Például: LINPACK, LAPACK stb.

- Párhuzamosított változat: ScaLAPACK
- Intel processzorokra optimalizált: MKL

Támogatják speciális mátrixok optimális tárolását is, pl.:

- Szimmetrikus mátrix, háromszög-mátrix
- Sávmátrixok
- Ritka mátrixok (majdnem minden elem 0)

Gauss–Jordan-elimináció

Feladat: megoldani az alábbi egyenletrendszert:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b},$$

ahol $\mathbf{A}$ négyzetes mátrix.

A Gauss–Jordan-elimináció tulajdonképpen az általános iskolában tanult módszer lineáris egyenletrendszerek megoldására.

- Egyszerre állítja elő az egyenletrendszer megoldását, és
- adja meg az $\mathbf{A}$ mátrix inverzét.
- Numerikusan legalább annyira stabil, mint más eljárás,
- főleg teljes *pivotolás*[4] esetén

[4] pivot = csuklópont, de a *pivoting* és *pivot element* szavaknak nincsen bevett fordítása

A Gauss–Jordan-elimináció tulajdonságai

Memóriaigény:

- tárolni kell a mátrixot, és a megoldásvektort: $N^2 + N$
- a mátrix inverzét elvileg lehet tárolni a bemeneti mátrix helyén

Számítási igény:

- szükséges műveletek száma $O(N^3)$
- de ha olyan algoritmusokat nézünk, amik hasonlóan megoldják az egyenletrendszer, és ugyanakkor elő is állítják a mátrix inverzét, akkor ez háromszor lassabb, mint a legjobb módszer

A megoldási módszer elemi lépései

A következő műveletek nem változtatják meg egy lineáris egyenletrendszer megoldását:

- *két sor felcserélése*
  ez nyilvánvaló, hiszen az egyenletek sorrendje teljesen tetszőleges, feltéve persze, hogy a jobboldal megfelelő sorait is megcseréljük
- *más sorok lineárkombinációjának hozzáadása bármely sorhoz*
  ebbe belefér az is, hogy egy sort számmal szorzunk, ha az a szám nem nulla; természetesen mindkét oldalon el kell végezni
- *a változók felcserélése*
  ez lényegileg nem változtat az egyenleteken, ha emlékszünk, hogy a végén a változókat megfelelő permutáció szerint vissza kell cserélgetni; ez az $\mathbf{A}$ mátrix oszlopainak felcserélését jelenti.

Kiindulás: kibővített mátrix

A megoldandó egyenletrendszer:

$$\begin{pmatrix} 2 & 6 & 2 \\ 1 & 1 & -1 \\ 3 & 9 & 5 \end{pmatrix} \cdot \mathbf{x} = \begin{pmatrix} 18 \\ 1 \\ 35 \end{pmatrix}$$

Felírjuk a mátrixot és a jobb oldalt a következő alakban:

$$\left(\begin{array}{ccc|c} 2 & 6 & 2 & 18 \\ 1 & 1 & -1 & 1 \\ 3 & 9 & 5 & 35 \end{array}\right)$$

Ha az inverz mátrixot keressük, akkor pedig:

$$\left(\begin{array}{ccc|ccc} 2 & 6 & 2 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 1 & 0 \\ 3 & 9 & 5 & 0 & 0 & 1 \end{array}\right)$$

(A jobb oldalon tulajdonképpen akárhány oszlop állhat, amennyiben egyszerre több egyenletet akarunk megoldani.)

Az elimináció menete

Cél: a korábban ismertetett elemi műveletek segítségével a baloldalt egységmátrix alakra hozzuk úgy, hogy közben a műveleteket a jobb oldalon is elvégezzük. Ekkor jobboldalt előáll az egyenletrendszer megoldása (illetve a mátrix inverze).

Elindulunk a főátló első elemétől.

1. Ha az elem nem 1, akkor az egész sort elosztjuk a főátlóbeli elemmel, hogy a főátlóban 1 legyen

$$\left(\begin{array}{ccc|c} \mathbf{2} & 6 & 2 & 18 \\ 1 & 1 & -1 & 1 \\ 3 & 9 & 5 & 35 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} \mathbf{1} & \mathbf{3} & \mathbf{1} & \mathbf{9} \\ 1 & 1 & -1 & 1 \\ 3 & 9 & 5 & 35 \end{array}\right)$$

2. Minden alatta levő sorból kivonjuk az első sor valahányszorosát úgy, hogy a főátló alatt végig 0 legyen

$$\left(\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 1 & 1 & -1 & 1 \\ 3 & 9 & 5 & 35 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ \mathbf{0} & -2 & -2 & -8 \\ \mathbf{0} & 0 & 2 & 8 \end{array}\right)$$

3. Ha főátlóbeli elem alatt mindent kinulláztunk, akkor folytatjuk a főátló következő elemével:

$$\left(\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 0 & \mathbf{-2} & -2 & -8 \\ 0 & 0 & 2 & 8 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ \mathbf{0} & \mathbf{1} & \mathbf{1} & 4 \\ 0 & 0 & 2 & 8 \end{array}\right)$$

4. Majd kivonjuk a sor valahányszorosát az összes többiből úgy, hogy a főátlót kivéve mindenütt 0 legyen:

$$\left(\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 0 & 1 & 1 & 4 \\ 0 & 0 & 2 & 8 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 1 & \mathbf{0} & -2 & -3 \\ 0 & 1 & 1 & 4 \\ 0 & \mathbf{0} & 2 & 8 \end{array}\right)$$

5. Az eljárást tovább folytatjuk a főátló elemeire

$$\left(\begin{array}{ccc|c} 1 & 0 & -2 & -3 \\ 0 & 1 & 1 & 4 \\ 0 & 0 & \mathbf{2} & 8 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 1 & 0 & -2 & -3 \\ 0 & 1 & 1 & 4 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{4} \end{array}\right)$$

$$\longrightarrow \left(\begin{array}{ccc|c} 1 & 0 & \mathbf{0} & 5 \\ 0 & 1 & \mathbf{0} & 0 \\ 0 & 0 & 1 & 4 \end{array}\right)$$

6. Az eljárás akkor ér végét, ha a baloldali részmátrix az egységmátrix alakját veszi fel. Ekkor jobboldalon vagy a megoldást kapjuk, vagy pedig a mátrix inverzét, attól függően miből indultunk ki.

$$\left(\begin{array}{ccc|c} \mathbf{1} & 0 & 0 & 5 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 4 \end{array}\right)$$

1. probléma: Zérus elem a főátlóban

Képzeljük el a következő szituációt:

$$\left(\begin{array}{ccc|c|c} 1 & 0 & 5 & & \\ 0 & 1 & 6 & & \\ 0 & 0 & \mathbf{0} & \cdots & \cdots \\ & & \vdots & & \end{array}\right)$$

Ilyenkor az adott sorral nem tudjuk az alatta és fölötte levő elemeket eliminálni.

Megoldás: sorcsere! Keressünk a főátló aktuális eleme alatti oszlopban olyan elemet, ami nem nulla (ez lesz az ún. *pivot-elem*), cseréljük meg a két sort (a jobboldalt is!), majd folytassuk az eliminációt, mintha mi se történt volna.

Az eljárás neve: *részleges pivotolás*.

2. probléma: Nagyon pici elem a főátlóban

Ha ezzel próbálnánk eliminálni, akkor nagy szorzótényezők miatt nagyon nagy számok jelennének meg a mátrixban, ami numerikus instabilitási problémákhoz vezet.

Megoldás: válasszuk az adott oszlop főátló alatti abszolút értékben legnagyobb elemét, a megfelelő sorokat cseréljük meg, és folytassuk így az eliminációt.

3. probléma: a mátrix sorai tetszőlegesen normálhatók

A sorokat tetszőleges számmal szorozva végül is bármelyik aktuális oszlopbeli, főátló alatti elem lehet maximális.

Megoldás: Úgy keressük a legnagyobb elemet, hogy a sorokat az eredeti mátrix sorainak legnagyobb elemével normáljuk.

- Ez a gyakorlatban azt jelenti, hogy a soronkénti legnagyobb elemet tárolni kell.
- Vagy a mátrix sorait a legelején leosztjuk minden sor abszolút értékben maximális elemével.

4. probléma: Az oszlop főátló alatti része csupa 0

$$\left(\begin{array}{ccc|c|c} 1 & 0 & 5 & & \\ 0 & 1 & 6 & & \\ 0 & 0 & \mathbf{0} & \cdots & \cdots \\ & & \mathbf{0} & & \\ & & \mathbf{0} & & \\ & & \vdots & & \\ & & \mathbf{0} & & \end{array}\right)$$

Ez könnyű: a mátrix szinguláris, megállunk.

További javítási lehetőség: oszlopcsere

Ilyenkor a főátlóbeli aktuális elem alatti, és tőle jobbra levő almátrixban keressük az abszolút értékben legnagyobb elemet, ez lesz a *pivot-elem*.

A megfelelő sorokat *és* oszlopokat megcseréljük (feljegyezve, hogy melyik két változót kell az eljárás végén visszacserélni!), majd a pivot-elemmel eliminálunk.

Ezt nevezzük *teljes pivotolásnak*.

Nem bizonyított állítás, csak sejtés, hogy az algoritmus stabil, ha pivotnak mindig az almátrix abszolút értékben legnagyobb elemét választjuk.

Gauss-elimináció visszahelyettesítéssel

Ez nagyjából azonos az előbbiekkel, de

- Eliminálás kor csak a főátló alatti elemeket nullázuk le
- A főátlóban minden elemet 1-esre hozunk
- A felső háromszöget csak ezután, a jobb alsó sarokból kiindulva nullázzuk ki
- Ez némileg kevesebb műveletet igényel, ezért numerikusan stabilabb

$$\left(\begin{array}{ccc|c} 1 & 3 & 1 & 9 \\ 0 & 1 & 1 & 4 \\ 0 & 0 & 2 & 8 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} \mathbf{1} & 3 & 1 & 9 \\ \mathbf{0} & \mathbf{1} & 1 & 4 \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & 4 \end{array}\right) \longrightarrow$$

$$\left(\begin{array}{ccc|c} 1 & 3 & \mathbf{0} & \mathbf{5} \\ 0 & 1 & \mathbf{0} & \mathbf{0} \\ 0 & 0 & 1 & 4 \end{array}\right) \longrightarrow \left(\begin{array}{ccc|c} 1 & \mathbf{0} & 0 & \mathbf{5} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 4 \end{array}\right)$$

Az algoritmus építőkövei

Az implementálást sosem a fő algoritmussal kezdjük, hanem azonosítjuk a fő részalgortimusokat, amiket külön-külön könnyű megírni.

- Eldöntjük, hogy milyen formában tároljuk a mátrixokat (sorok vagy oszlopok szerint)
- Azonosítjuk az alapvető műveleteket:
  - Legnagyobb abszolút értékű elem megtalálása sorban
  - Sor szorzása számmal
  - Legnagyobb abszolút értékű elem megtalálása oszlopban, főátló alatt
  - Legnagyobb abszolút értékű elem megtalálása almátrixban
  - Két sor cseréje
  - Két oszlop cseréje (oszlopcsere könyvelése)
  - Két sor különbségének képzése

Az algoritmus implementálása

- Az építőelemeket külön-külön megvalósítjuk és
- Külön-külön *teszteljük*!

Soha ne írjunk úgy programot, hogy előbb "nagyjából" kész legyen, aztán majd javítgatjuk. Soha sem fog működni. Mindig alulról felfelé, az egyes függvényeket alaposan kipróbálva kell haladni!

- Ezek után jöhet a fő algoritmus leprogramozása
- Többfajta mátrixon teszteljük

Direkt kell olyan mátrixokat gyártani, ami próbára teszi az algoritmust. Például: van a főátlóban 0 elem, szinguláris mátrix stb. A numerikus stabilitás tesztelése ugyanakkor nehéz feladat.

Alsó-felső háromszög dekompozíció

Invertáláshoz a mátrixot gyakran érdemes *faktorizálni*, azaz kettő vagy több speciális tulajdonságú mátrix szorzataként előállítani.

A legegyszerűbb faktorizáció: LU-dekompozíció.

Az $\mathbf{A}$ mátrixot $\mathbf{A} = \mathbf{L} \cdot \mathbf{U}$ alakban keressük, ahol

- $\mathbf{L}$ csak főátlóbeli, és főátló alatti elemeket tartalmaz
- $\mathbf{U}$ csak főátlóbeli, és főátló fölötti elemeket tartalmaz

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

Az LU-dekompozíció alapötlete

Vegyük észre, hogy ha $\mathbf{A} = \mathbf{L} \cdot \mathbf{U}$, akkor az $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ egyenlet átírható

$$\mathbf{A} \cdot \mathbf{x} = (\mathbf{L} \cdot \mathbf{U}) \cdot \mathbf{x} = \mathbf{L} \cdot (\mathbf{U} \cdot \mathbf{x}) = \mathbf{b}$$

alakra, azaz a probléma két lépésre bontható:

$$\mathbf{L} \cdot \mathbf{y} = \mathbf{b}$$

$$\mathbf{U} \cdot \mathbf{x} = \mathbf{y}$$

A módszer előnye, hogy mivel $\mathbf{L}$ és $\mathbf{U}$ háromszög-mátrixok, a korábbi visszahelyettesítéses módszerrel közvetlenül meghatározhatók:

- $\mathbf{L}$ esetében a bal felső sarokból kell indulni
- $\mathbf{U}$ esetében a jobb alsóból

L és **U** meghatározása

L és **U** összesen $N^2 + N$ ismeretlen elemet tartalmaz az

$$(\mathbf{L} \cdot \mathbf{U}) \cdot \mathbf{x} = \mathbf{b}$$

egyenletrendszer viszont csak N^2 egyenletet ad:

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

Kössük, ki hogy az **L** mátrix főátlójában csupa 1-es szerepel, azaz $l_{ii} = 1$. Ez az általánosság megszorítása nélkül megtehető. Ez további N egyenletet ad.

Az $N^2 + N$ egyenlet megoldása

Az egyenletrendszer a Crout-algoritmussal megoldható:

for $j = 1, 2, \ldots, N$ **do**
  for $i = 1, 2, \ldots, j$ **do**
    az első két egyenlet felhasználásával:
    $u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}$
  end
  for $i = j + 1, j + 2, \ldots, N$ **do**
    a harmadik egyenlet felhasználásával:
    $l_{ij} = (1/u_{jj})(a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}$
  end
end

Az algoritmus második belső ciklusában az u_{jj} elem a nevezőben szerepel. Elkerülendő, hogy ez ne legyen 0, a Gauss-eliminációhoz hasonlóan itt is szükség lehet átrendezésre.

Az LU-dekompozíció előnyei

Az **A** mátrix determinánsa előáll a következő alakban:

$$\det \mathbf{A} = \prod_{j=1}^{N} u_{jj}$$

A dekompozíciót elegendő egyszer elvégezni, és utána már a visszahelyettesítéses módszerrel gyorsan megkapható az egyenletrendszer megoldása bármilyen jobb oldal esetére. Ugyanígy a mátrix inverze is könnyen számolható.

Iteratív módszerek

Iteratív lépés

- Általában egy egyszerű műveletet ismételgetünk
- A művelet egy függvény, aminek bemenete x_k, kimenete x_{k+1}
- A következő lépés bemenete az előző lépés kimenete

Tekintsük a lépések által előállított x_k értékek sorozatát

- Tart-e az x_k sorozat valahova?
- Ha igen, akkor az iteráció konvergens
- A konvergenciát vagy elméletileg kell belátni, vagy
- Lehet programmal is vizsgálni: $|x_{k+1} - x_k| \stackrel{?}{\to} 0$
- Ha tudjuk, hogy konvergens, akkor megállhatunk, amikor $|x_{k+1} - x_k| < \epsilon$, ahol ϵ elő van írva

Honnan induljon az iteráció?

Természetesen valami x_0 értékből kell elindulni

- Ennek jó megválasztása gyakran kérdés
- Ha bárhonnan indulva konvergens az iteráció, akkor a konvergencia *globális*
- Ha csak bizonyos helyekről konvergens, akkor a konvergencia *lokális*
- Előfordul, hogy máshonnan indulva máshova konvergál az iteráció

A lineáris iterációs módszerek, ha konvergensek, akkor globálisan konvergensek.

Iteratív módszerek egyenletrendszerek megoldására

A megoldást iteratívan érdemes előállítani, ha

- az egyenletrendszer nagyon nagy
- közel szinguláris, és numerikusan nehezen kezelhető

Tegyük fel, hogy ismerünk egy $\mathbf{x} + \delta\mathbf{x}$ közelítő megoldást.
(Csak az összeg ismert, de mi $\mathbf{x}$ értékét keressük.)

Behelyettesítve az egyenletbe a jobb oldal el fog térni az eredetitől:

$$\mathbf{A} \cdot (\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$$

Ebből kivonva az eredeti $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ egyenletet:

$$\mathbf{A} \cdot \delta\mathbf{x} = \delta\mathbf{b}$$

Iterációs lépés

Viszont az eredetit átrendezve:

$$\delta\mathbf{b} = \mathbf{A} \cdot (\mathbf{x} + \delta\mathbf{x}) - \mathbf{b},$$

aminek a jobb oldalát ismerjük, tehát megoldhatjuk $\delta\mathbf{x}$-re a

$$\mathbf{A} \cdot \delta\mathbf{x} = \mathbf{A} \cdot (\mathbf{x} + \delta\mathbf{x}) - \mathbf{b}$$

egyenletet.

Ha $\mathbf{A}$ LU-faktorizációját ismerjük, akkor csak behelyettesítgetni kell. Várhatóan pontosabb megoldást kapunk, és az eljárást ismételve egyre konvergálunk az egyenlet megoldásához.

Iterációs módszerek általánosabban

Írjuk át az $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ egyenletet:

$$\mathbf{x} = (\mathbf{I} - \mathbf{A}) \cdot \mathbf{x} + \mathbf{b},$$

ahol $\mathbf{I}$ az egységmátrix.

Itt $\mathbf{x}$ kivételével minden ismert, és ha ismerünk egy kellően jó $\mathbf{x}_k$ megoldást, akkor képezhetjük a következő iterációt:

$$\mathbf{x}_{k+1} = (\mathbf{I} - \mathbf{A}) \cdot \mathbf{x}_k + \mathbf{b}$$

- Belátható, hogy ha $\mathbf{I} - \mathbf{A}$ mátrix-normája kisebb egynél, akkor az iteráció konvergens.
- Itt $\mathbf{A}$-t nem kell invertálni (vagy faktorizálni)