4. Lineáris egyenletrendszerek közelítő megoldása iterációval

A lineáris algebrai egyenletrendszerek megoldhatók direkt és iterációs módszerekkel is.
A **direkt módszerek**nél olyan algoritmust adunk meg, mely véges sok lépésben kiszámolja a **pontos megoldás**t, pontos kiindulási adatokat feltételezve és a kerekítési hibáktól eltekintve (Gauss-Jordan elimináció, Cramer szabály). Gyakorlati feladatoknál a kapott eredmény hibával terhelt lesz, mivel már a kezdeti adatok sem pontosak (mérési, modellezési hibák miatt).
A direkt módszereket főleg kis és közepes méretű egyenletrendszerek esetén használják. A gyakorlatban sok nagyméretű, ritka mátrixú egyenletrendszer (amelyekben sok együttható értéke 0) keletkezik, ezeknél a direkt módszerek hátránya a nagy tárigény, ezért ilyen típusú mátrixokra iterációs módszert alkalmaznak.
Az **iterációs módszer**ek egy (általunk megadott) kiindulási vektorból (a változók kezdőértékeiből megadott rendezett számokból) egy **vektorsorozat**ot állítanak elő, mely (megfelelő tulajdonságokkal bíró A együtthatómátrix esetén) az egyenletrendszer **megoldásához konvergál**. A gyakorlatban az iterációs lépések számára meg kell adni egy leállási kritériumot, melynek meghatározása újabb problémát vet fel. (Az egyik lehetséges módszer, hogy akkor állítjuk meg az iterációt, amikor az egymást követő vektorok különbsége már kisebb egy általunk megadott hibánál, egy másik pedig az, hogy akkor állunk meg, ha az $r_k = A x_k$ -b úgynevezett reziduum elég kicsi, ahol x_k a k. küzelítő megoldás.)
Az iterációs módszereknél két fontos szempont van:
- a vektorsorozat elemeit könnyen, "olcsón" tudjuk kiszámolni (hiszen pont ez az előnye a direkt módszerekkel szemben),
- az iteráció gyorsan konvergáljon a megoldáshoz (minél gyorsabb a konvergencia, annál kevesebb lépésre lesz szükség).

Az iterációs módszereknél az egyenletrendszer együtthatómátrixát olyan mátrixok összegére (ill. különbségére) bontjuk, melyekkel könnyebben és gyorsabban tudunk számolni, mint az eredeti mátrixszal. Ezt a módszerek levezetésénél használjuk, a tényleges számolás általában már az eredeti A mátrix elemeit használja.
Nagy méretű lineáris egyenletrendszerek megoldására több módszer ismeretes.
A megoldási módszer kiválasztásának kritériumai:
a) Mennyire gyorsan vezet eredményre? (Mennyi számítást kell elvégezni?)
b) Mennyire pontos a kiszámított megoldás?
A problémák eredete és jellegük az egyenletrendszer együtthatómátrixa szerint, ha az együtthatómátrix négyzetes mátrix:
a) "Kitöltött" (a zérus értékű elemek száma kicsi) és a rendszáma (a sorok száma) kicsi (<30)
Alkalmazás: statisztika, fizika problémák megoldása során.
b) "Ritka" (kevés zérustól különböző elem van, és azok leginkább a főátlóban és amellett helyezkednek el, valamint rendszámuk igen nagy (>30).
Alkalmazás: parciális differenciálegyenletek numerikus megoldásánál.

Műszaki problémák leírásánál mindkét tipus előfordul.

4. 1. A lineáris egyenletrendszer megoldásának hibaforrásai

Használt Maple parancsok: MatrixNorm, ConditionNumber

Az egyenletrendszerek vizsgálatára alkalmasak a következő definíciók illetve a következő tétel.

4. 1. 1. Definíció

Vektor összeg normája $\|x\|_1 = \sum_{i=1}^{n} |x_i|$, **vektor euklideszi normája** $\left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}}$ (ez a hagyományos vektor hosszúság)

A **mátrix sorösszeg normája**: $\|\mathbf{A}\|_\infty = \max_{1 \leqslant j \leqslant n} \sum_{i=1}^{m} |a_{ij}|$, A **mátrix oszlopösszeg normája:**

$\|\mathbf{A}\|_1 = \max_{1 \leqslant i \leqslant n} \sum_{j=1}^{n} |a_{ij}|$. □

Ha egy mátrix normája **1**, akkor a mátrixot **normált**nak nevezzük. □

4. 1. 2. Definíció

A **mátrix kondícionáltsága** (a kezdeti adatoktól való függés): $\mathrm{cond}(\mathbf{A}) = \|\mathbf{A}\| \, \|\mathbf{A}^{-1}\|$ □ (ha A nem szinguláris)

Ha a kondíciószám 1-hez közeli, az egyenletrendszer **jól kondicionált,** különben rosszul kondícionált.

4. 1. 3. Tétel

Érzékenységvizsgálat: Az $\mathbf{Ax} = \mathbf{b}$ illetve $(\mathbf{A} + \Delta\mathbf{A})(\mathbf{x} + \Delta\mathbf{x}) = (\mathbf{b} + \Delta\mathbf{b})$ egyenletrendszerek esetén a hibák eredményre gyakorolt hatását a következő tétel alapján vizsgálhatjuk.

4. 1. 4. Öröklött hiba

1. Példa Figyeljük meg az alábbi egyenletrendszereket, mi történik, ha az együtthatókat kismértékben megváltoztatjuk (a mért eredmények kissé megváltoznak)!

i.) $2x + 6y = 8,$ $2x + 6.00001\,y = 8.00001$ *ii.*) $2x + 6y = 8,$ $2x + 5.99999\,y = 8.00002$

Megoldás

```
> restart; with(LinearAlgebra) :
> rendszer1 := [2 x + 6 y = 8, 2 x + 6.00001 y = 8.00001 ]
> A1, b1 := GenerateMatrix(rendszer1, [x, y]) :
> megoldas1 := MatrixInverse(A1).b1
```

$$rendszer1 := [2x + 6y = 8, 2x + 6.00001\,y = 8.00001]$$

$$megoldas1 := \begin{bmatrix} 1. \\ 1. \end{bmatrix}$$

Kismértékben változtassuk meg az egyenletrendszer konstansait !

> $rendszer2 := [2x + 6y = 8, 2x + 5.99999y = 8.00002]$
> $A2, b2 := GenerateMatrix(rendszer2, [x, y])$:
> $megoldas2 := MatrixInverse(A2).b2$

$$rendszer2 := [2x + 6y = 8, 2x + 5.99999y = 8.00002]$$

$$megoldas2 := \begin{bmatrix} 10. \\ -2. \end{bmatrix}$$

Kis változás az együtthatókban az ereményben óriási változást eredményez! Mi okozhatta ezt a nagymértékű változást? Vizsgáljuk meg az együtthatómátrixokat és inverzüket!

> $A1 := A1; inverzA1 := MatrixInverse(A1)$
> $A2 := A2; inverzA2 := MatrixInverse(A2)$

$$A1 := \begin{bmatrix} 2 & 6 \\ 2 & 6.00001 \end{bmatrix}$$

$$inverzA1 := \begin{bmatrix} 3.00000500011357\ 10^5 & -3.00000000011357\ 10^5 \\ -1.00000000003786\ 10^5 & 1.00000000003786\ 10^5 \end{bmatrix}$$

$$A2 := \begin{bmatrix} 2 & 6 \\ 2 & 5.99999 \end{bmatrix}$$

$$inverzA2 := \begin{bmatrix} -2.99999500011357\ 10^5 & 3.00000000011357\ 10^5 \\ 1.00000000003786\ 10^5 & -1.00000000003786\ 10^5 \end{bmatrix}$$

Figyeljük meg, az **A1** és **A2** inverz mátrixok elemei igen különböző, nagy számok! Vizsgáljuk meg, mi történik, ha az egyenletek mindkét oldalán ugyanazzal a nagy számmal megszorozzuk:

> $rendszer11 := [2\ 10^5 x + 6\ 10^5 y = 8\ 10^5, 2\ 10^5 x + 6.00001\ 10^5 y = 8.00001\ 10^5]$
> $A11, b11 := GenerateMatrix(rendszer11, [x, y])$:
> $megoldas11 := MatrixInverse(A11).b11$

$$rendszer11 := [200000x + 600000y = 800000, 200000x + 6.000010000\ 10^5 y = 8.000010000\ 10^5]$$

$$megoldas11 := \begin{bmatrix} 0.999999999534339 \\ 1. \end{bmatrix}$$

> $rendszer22 := [2\ 10^5 x + 6\ 10^5 y = 8\ 10^5, 2\ 10^5 x + 5.99999\ 10^5 y = 8.00002\ 10^5]$
> $A22, b22 := GenerateMatrix(rendszer22, [x, y])$:
> $megoldas22 := MatrixInverse(A22).b22$
> $inverzA22 := MatrixInverse(A22)$
> $inverzA11 := MatrixInverse(A11)$

$$rendszer22 := [200000\,x + 600000\,y = 800000,\ 200000\,x + 5.999990000\ 10^5\,y = 8.000020000\ 10^5]$$

$$megoldas22 := \begin{bmatrix} 10. \\ -2. \end{bmatrix}$$

$$inverzA22 := \begin{bmatrix} -2.99999500000000 & 3.00000000000000 \\ 1. & -1. \end{bmatrix}$$

$$inverzA11 := \begin{bmatrix} 3.00000500000000 & -3.00000000000000 \\ -1. & 1. \end{bmatrix}$$

Természeresen a megoldások nem változtak, pedig az inverz mátrix elemei kicsik. Ha az együtthatók értékei mérési eredmények, **kis mérési hiba az eredményben durva eltérést eredményezhet**!

Mivel az együtthatók közel azonosak (determinánsuk közel 0), azt mondjuk, hogy az **egyenletrendszer gyengén kondícionált (**meghatározott).

> $determinans\_A1 := Determinant(A1);\ determinans\_A2 := Determinant(A2)$

$$determinans\_A1 := 0.00002$$

$$determinans\_A2 := -0.00002$$

> $sor\_osszeg\_norma1 := MatrixNorm(A1, 1);$
$oszlop\_osszeg\_norma1 := MatrixNorm(A1, \infty);$
> $sor\_osszeg\_norma2 := MatrixNorm(A2, 1);$
$oszlop\_osszeg\_norma1 := MatrixNorm(A2, \infty);$

$$sor\_osszeg\_norma1 := 12.0000099999999996$$

$$oszlop\_osszeg\_norma1 := 8.00001$$

$$sor\_osszeg\_norma2 := 11.9999900000000004$$

$$oszlop\_osszeg\_norma1 := 8$$

> $kondicioszam1 := ConditionNumber(A1);\ kondicioszam2 := ConditionNumber(A2);$

$$kondicioszam1 := 4.800010000\ 10^6$$

$$kondicioszam2 := 4.799996001\ 10^6$$

Vizsgáljuk meg, teljesülnek-e a 4.1.3 tétel feltételei.

> $ConditionNumber(A1, 1)\ \frac{MatrixNorm(A2 - A1, 1)}{MatrixNorm(A1, 1)}$

$$8.000010000$$

A tétel 2. feltétele nem teljesül, így a hiba becslésére a tétel nem alkalmas. A tétel állításában szereplő egyenlőtlenség jobb oldalában, a $\frac{cond(\mathbf{A})\left(\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|} + \frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|}\right)}{1 - cond(\mathbf{A}).\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}}$ kifejezésben szereplő

$cond(\mathbf{A}).\frac{\|\Delta\mathbf{A}\|}{\|\mathbf{A}\|}$ értéke 1-nél nagyobb,

> $\frac{ConditionNumber(A1)\ MatrixNorm(A2-A1,\ \infty)}{MatrixNorm(A1,\ \infty)}$

$$12.00001000$$

így a teljes tört értéke negatív, ami nem adhat két norma hányadosát.

2. Példa Vizsgáljunk egy jól kondícionált rendszert:
$$\begin{aligned} 2x_1 - x_2 &= 3,\\ -x_1 + 2x_2 - x_3 &= -4,\\ -x_2 + 2x_3 &= 3 \end{aligned}$$

> $rendszer := [2x_1 - x_2 = 3,\ -x_1 + 2x_2 - x_3 = -4,\ -x_2 + 2x_3 = 3]$
> $A, b := GenerateMatrix(rendszer, [x_1, x_2, x_3])$:
> $sor\_osszeg\_norma := MatrixNorm(A, 1)$; $oszlop\_osszeg\_norma := MatrixNorm(A, \infty)$; $kondicioszam := ConditionNumber(A)$;

$$rendszer := [2x_1 - x_2 = 3,\ -x_1 + 2x_2 - x_3 = -4,\ -x_2 + 2x_3 = 3]$$

$$sor\_osszeg\_norma := 4$$

$$oszlop\_osszeg\_norma := 4$$

$$kondicioszam := 8$$

> $megoldas := MatrixInverse(A).b$

$$megoldas := \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

Ilyen esetekben az együtthatómátrix kis változása nem befolyásolja jelentősen a megoldást.

> $rendszer\_modositott := [2.01x_1 - x_2 = 3,\ -1.06x_1 + 2x_2 - x_3 = -4,\ -x_2 + 2x_3 = 3]$
> $A\_modositott, b\_modositott := GenerateMatrix(rendszer\_modositott, [x_1, x_2, x_3])$:
> $megoldas := MatrixInverse(A).b$
> $megoldas\_modositott := MatrixInverse(A\_modositott).b$

$$rendszer\_modositott := [2.01x_1 - x_2 = 3,\ -1.06x_1 + 2x_2 - x_3 = -4,\ -x_2 + 2x_3 = 3]$$

$$megoldas := \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix}$$

$$megoldas\_modositott := \begin{bmatrix} 1.02301790281330 \\ -0.943734015345268 \\ 1.02813299232737 \end{bmatrix}$$

Erre a jól kondícionált egyenletrendszerrre az érzékenységvizsgálat elvégezhető, hiszen:

> $Determinant(A)$; $ConditionNumber(A)\ \frac{MatrixNorm(A\_modositott - A, infinity)}{MatrixNorm(A, infinity)}$

4

0.1200000000

vagyis a tétel feltételei teljesülnek. A tétel állításának bal oldala:

> $bal := simplify\left(\frac{MatrixNorm(megoldas\_modositott - megoldas,\ \infty)}{MatrixNorm(megoldas,\ \infty)} \right)$

$bal := 0.05626598465$

A jobb oldal:

> $jobb := \left(ConditionNumber(A) \left(\frac{MatrixNorm(A\_modositott - A,\ \infty)}{MatrixNorm(A,\ \infty)} + \frac{MatrixNorm(b\_modositott - b,\ \infty)}{MatrixNorm(b,\ \infty)} \right) \right) \Big/ \left(1 - \frac{ConditionNumber(A)\ MatrixNorm(A\_modositott - A,\ \infty)}{MatrixNorm(A,\ \infty)} \right)$

$jobb := 0.1363636364$

Tehát a mérési hibából eredő számítási hiba 14%-nál kisebb.

4. 1. 5. Kerekítési hiba

A gyakorlatban előálló egyenletrendszerek esetén az együtthatók is csak bizonyos pontossággal adottak, tehát fölösleges arra törekedni, hogy az ismeretlenek értékét nagyobb pontossággal számítsuk ki, mint az együtthatók pontossága egyáltalán lehetővé teszi. A kerekítési hiba az alkalmazott számítógépek véges pontosságú számábrázolásából, az alapműveletek eredményein végzett kerekítésből adódnak.

A direkt módszer csak elvileg ad pontos megoldást, a gyakorlatban a kerekítési hibák halmozódása nagyon nehezen megbecsülhető módon eltéríthet a pontos megoldástól. Az iterációnál a kerekítési hibák nem halmozódnak, és amennyiben az eljárás konvergens, a hiba könnyen becsülhető.

4. 1. 6. Képlethiba

Képlethiba akkor lép fel, amikor egy végtelen eljárást véges számú lépés után leállítunk, közelítő algoritmusokat alkalmazunk.
Direkt módszereknél nem lép fel, iterációs eljárásoknál végtelen számú lépéssel kaphatnánk pontos értéket, a közelítést véges számú lépéssel tesszük, ezért képlethiba mindig fellép.

4. 2. Szabályos lineáris egyenletrendszerek megoldása iterációval (Reguláris szétvágás módszere)

4. 2. 1. Definíció

Egy iteráció **konvergens**, ha valamely normában igaz, hogy $\lim_{n \to \infty} \| \mathbf{x}^* - \mathbf{x}^{(n)} \| = 0$, ahol $\mathbf{x}^*$ a valódi, $\mathbf{x}^{(n)}$ az n-edik közelítő megoldás.

Amint azt a

$$\mathbf{Ax = b}$$

szabályos lineáris egyenletrendszer direkt megoldásánál láttuk a megoldás

$$\mathbf{x=A^{-1}b}$$

alakú, vagyis a megoldáshoz az **A** együtthatómátrix inverzét kell megkeresni. Ez bonyolult egyenletrendszereknél hosszú és fáradtságos folyamat.

Az **A** együttható mátrixát bontjuk fel két mátrix különbségére. Legyen

$$\mathbf{A = P - N},$$

ahol **P** - t prekondicionáló mátrixnak bevezzük.

Ekkor az egyenletrendszer mátrixos alakja átrendezés után

$$\mathbf{Px=Nx+b}$$

alakú.

Ha a **P** mátrix invertálható (determinánsa nem 0), az egyenletrendszer megoldásvektora

$$\mathbf{x=P^{-1}(Nx + b)}$$

alakú.

Az iteráció azt jelenti, hogy valamely $\mathbf{x}^{(0)}$ megoldásvektorból kiindulva, azt az egyenlet jobb oldalába beírva megkaphatjuk az első közelítő gyököt, majd az eljárást folytatva, közelítő megoldások sorozatát kapjuk, ami (ha konvergens) a pontos megoldáshoz tart. Az *i*-edik közelítő megoldásvektor tehát

$$\mathbf{x}^{(i)} = \mathbf{P^{-1}}\left(\mathbf{Nx}^{(i-1)} + \mathbf{b}\right)$$

alakú.

Az eljárás alkalmazhatósága a **P** mátrix jó megválasztásától függ. Minél közelebb van **A**-hoz az **P** mátrix, az iteráció annál gyorsabban konvergál a valódi megoldáshoz, ugyanakkor annál költségesebb is kiszámolni a vektorsorozat elemeit. Ha például **P = A**, akkor csupán egyetlen lépésre van szükség a megoldáshoz. A megoldáshoz viszont az inverezeket kell "kiszámolnunk", miközben pont ennek a nehézsége indolkolja az iterációs módszer alkalmazását. A cél tehát az, hogy egy adott mátrixnál olyan felbontásokat találjunk, melyekben a **P** inverze lényegesen könnyebben számolható mint **A** inverze és sok iterációs lépés esetén is kisebb legyen a műveletigény, mint ha az eredeti egyenletrendszert oldanánk meg. Ugyanakkor teljesülnie kell a konvergenciának is, és persze annak gyorsasága sem elhanyagolható.

Jacobi módszer:

Legyen a **P**=**D** prekondícionáló mátrix diagonál mátrix, vagyis **A** főátlóbeli elemeit tartalmazza csak! A prekondícionálás most olyan, hogy az egyenleteket sorra x_1, x_2,....,x_n ismeretlenekre rendezi, és így közelít.

A Gauss-Seidel iteráció:

A **P = D + U** prekondicionáló mátrix az együtthatómátrix főátlója alatti elemeket is tartalmazza. Itt az egyenletek úgy rendezeződnek át, hogy az első egyenlet bal oldalán x_1-es, a második egyenlet

bal oldalán x_1, és x_2 -es, a harmadik sorában x_1, x_2,és x_3-as tagok az utolsóban pedig az összes x_1 , x_2,. .. x_n tagok maradjanak, így az egyenletek bal oldalára már a kiszámított közelítő értékek kerülnek.

4. 2. 1. Definíció

A $\mathbf{D}^{-1}\mathbf{N}$ mátrixot **Jacobi mátrixnak, a** $(\mathbf{D}+\mathbf{U})^{-1}\mathbf{N}$ mátrixot **Gauss-Seidel mátrix**nak nevezzük. □

4. 2. 2. Tétel

Ha az **A** együtthatómátrix elemeiből képezett összegre teljesül, hogy

$$\sum_{j=1}^{n} |a_{i,j}| \le |a_{i,i}| \quad \text{de} \quad i \neq j$$

vagyis az egyes sorokban a nem azonos sor- és oszlopindexű elemek összege kisebb vagy egyenlő, mint az azonos sor- és oszlopindexű elem, akkor a Jacobi és a Gauss-Seidel iteráció **konvergens**.

4. 2. 3. Tétel

Ha a Jacobi mátrix $b_{i,j}$ $i = 1 \ldots n$, j=1..n elemeiből képzett összegre teljsül, hogy:

$$\sum_{j=1}^{n} |b_{i,j}| \le 1$$

vagyis az egyes sorokban szereplő együtthatók összege kisebb vagy egyenlő mint 1, akkor a Jacobi és a Gauss-Seidel iteráció **konvergens**.

4. 3. Mintapéldák megoldása lépésenként

Használt Maple parancsok: implicitplot, is,seq, pointplot, LinearSolve

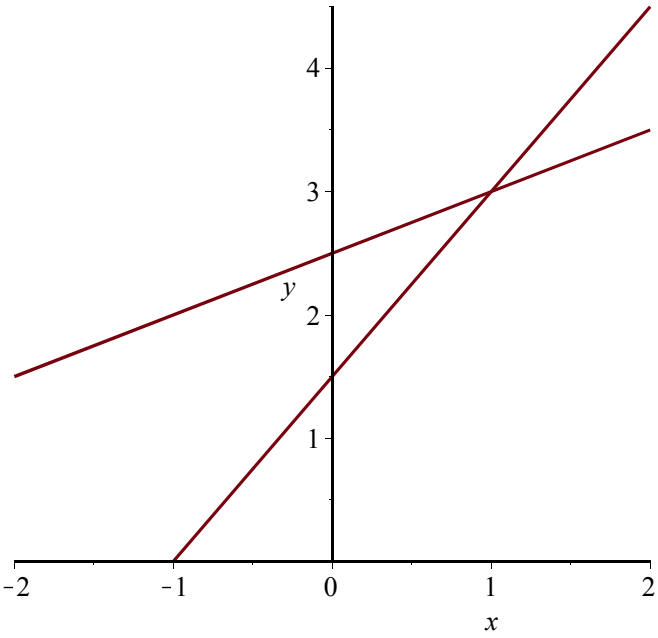
3. Példa Oldjuk meg a $3 \cdot x - 2 \cdot y = -3$, $-x + 2 \cdot y = 5$ egyenletrendszert iterációval, ábrázoljuk a közelítő megoldásokat, ha

a; **P** prekondicionáló mátrix diagonál mátrix (**Jacobi módszer**)
b; **P** alsó háromszög mátrix (**Gauss-Seidel módszer**)

Megoldás

Mivel az egyenletrendszer 2 egyenletet és két ismeretlent tartalmaz,ezért szemléletesen mindegyik egyenletnek egy egyenes felel meg. A 2 egyenes metszéspontját keressük.

```
> restart; rendszer := [3 x - 2 y = -3, -x + 2 y = 5, ] : with(plots) :
  abra1 := implicitplot( {rendszer[1], rendszer[2]}, x = -2 ..2, y = 0 ..5 ) : abra1;
```



Az egyenletrendszer megoldása természetesen az ábrából is leolvasható, illetve a solve utasítással direkt módon is megkapható.

> $solve(rendszer, \{x, y\})$

$$\{x = 1, y = 3\}$$

> $with(LinearAlgebra)$:

> $A, b := GenerateMatrix(rendszer, [x, y])$:

Az iterációs eljárások alkalmazhatók, hiszen

> $is(|A_{1,2}| \le |A_{1,1}| \textbf{ and } |A_{2,1}| \le |A_{2,2}|)$

$$true$$

vagyis a 4. 2. 2. tétel feltételei teljesülnek.

Megoldás a;

Állítsuk elő egy egyszerű eljárással a **P** prekondícionáló mátrixot. Az *id* nevű eljárás eljárás az azonos sor- és oszlopindexű elemekhez *A[i,j]* -t, a különböző sor- és oszlopindexű elemekhez 0-t rendel.

> $id :=\textbf{proc}(i, j)\ \textbf{if}\ i = j\ \textbf{then}\ A[i, j]\ \textbf{else}\ 0\ \textbf{end if}\ \textbf{end proc}$:

> $P := Matrix(2, 2, id)$

$$P := \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

A **N** mátrix

> $N := P - A$

$$N := \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix}$$

Így a Jacobi mátrix

> $J := MatrixInverse(P).N$

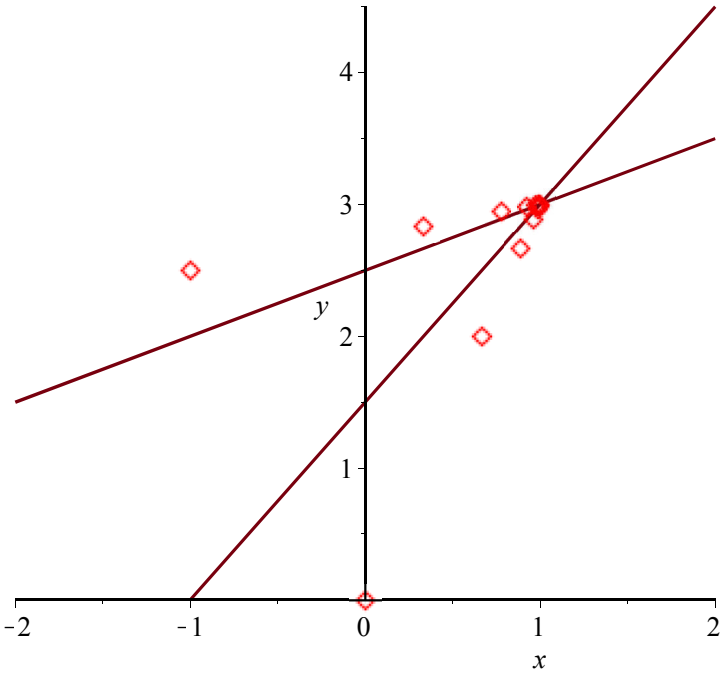
$$J := \begin{bmatrix} 0 & \frac{2}{3} \\ \frac{1}{2} & 0 \end{bmatrix}$$

Végezzük el az iterációt addíg, amíg az egymást követő két megoldás eltérése nem lesz kisebb, mint az előre megadott hiba! Meg kell adni a maximális lépésszámot, és a hiba kezdőértékét.

	Megvalósítás Maple -ben megadott hibahatárig
Input : $\mathbf{P}$, $\mathbf{N}$, $\boldsymbol{m}_0$, $\boldsymbol{b}$, ε $(\varepsilon > 0)$.	$\mathbf{P}$, $\mathbf{N}$, $\boldsymbol{m}_0$, $\boldsymbol{b}$, k, $hiba$
	> $m[0] := Vector([0, 0])$:
	> $k := 50$: $hiba := 100$:
	> **for** i **to** k **while** $10^{-2} < hiba$ **do** $m[i] := evalf(MatrixInverse(P).N.m[i-1] + MatrixInverse(P).b)$; $hiba := evalf(\lvert Norm(m[i-1] - m[i], 2)\rvert)$ **end do**:
$i=1,\ \mathbf{x}^{(i)} = \mathbf{P}^{-1}(\mathbf{N}\mathbf{x}^{(i-1)} + \mathbf{b})$	
$i = i + 1$	
amíg $\lvert\mathbf{x}^{(i)} - \mathbf{x}^{(i-1)}\rvert < \varepsilon$	
	> $print(lépésszám:, i-1)$; $print(`hiba:`, hiba)$; $print(megoldás: , m[i-1])$
	lépésszám:, 12
	hiba:, 0.007160704053
Output : $\mathbf{x}^{(i)}$	*megoldás:* , $\begin{bmatrix} 0.998628257887517 \\ 2.99588477366255 \end{bmatrix}$

Ábrázoljuk az egyeneseket és a kapott pontsorozatot egy közös ábrán

> $abra2 := pointplot(\{seq(m[j], j = 0..i-1)\}, color = red, symbolsize = 20)$:
$display([abra1, abra2])$;



Megoldás b;

P prekondicionáló az együtthatómátrix főátlója alatti elemeket is tartalmazza:

> $id := \mathbf{proc}(i,j)\ \mathbf{if}\ j <= i\ \mathbf{then}\ A[i,j]\ \mathbf{else}\ 0\ \mathbf{end\ if}\ \mathbf{end\ proc}$:

> $P := Matrix(2, 2, id)$

$$P := \begin{bmatrix} 3 & 0 \\ -1 & 2 \end{bmatrix}$$

> $N := P - A$

$$N := \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix}$$

> $m[0] := Vector([0, 0])$

$$m_0 := \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

> $k := 50 : hiba := 100 :$

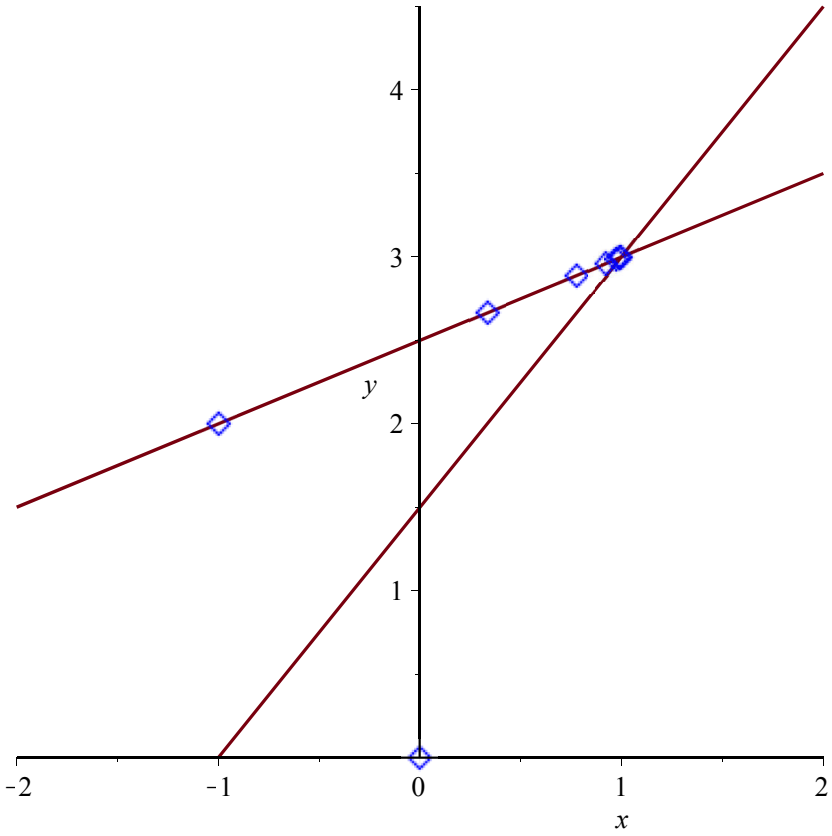
> **for** i **to** k **while** $10^{-2} < hiba$ **do**
$m[i] := evalf(MatrixInverse(P).N.m[i-1] + MatrixInverse(P).b);$
$hiba := evalf(|Norm(m[i-1] - m[i], 2)|)$
end do:
$print(lépésszám:, i-1);$
$print(`hiba:`, hiba);$
$print(megoldás:\ ,\ m[i-1])$

lépésszám:, 7

$$\text{hiba:}, 0.006134617222$$

$$\text{megoldás:}, \begin{bmatrix} 0.997256515775034 \\ 2.99862825788752 \end{bmatrix}$$

> $abra3 := pointplot(\{seq(m[j], j = 0..i-1)\}, color = \text{blue}, symbolsize = 20)$:
> $display([abra1, abra3])$



Megfigyelhető, hogy ugyanabból a kezdőértékből kiindulva, a Gauss-Seidel iteráció gyorsabban konvergál, már 7 lépésben elérjük a kívánt pontosságot, míg a Jacobi iterációnál ehhez 12 lépés kellett.

4. Példa Mutassuk meg, hogy a Jacobi iteráció nem alkalmas a
$3x_1 + 4x_3 = -1,\ 7x_1 + 4x_2 + 2x_3 = 17,\ -x_1 + x_2 + 2x_3 = 0$ egyenletrendszer megoldására.

Megoldás

Az egyenletrendszer direkt módon megoldható

> $restart; with(LinearAlgebra)$: $rendszer := [3x_1 + 4x_3 = -1, 7x_1 + 4x_2 + 2x_3 = 17, -x_1 + x_2 + 2x_3 = 0]$:
> $A, b := GenerateMatrix(rendszer, [x_1, x_2, x_3])$:
> $solve(rendszer, \{x_1, x_2, x_3\})$

$$\{x_1 = 1, x_2 = 3, x_3 = -1\}$$

> $id := \mathbf{proc}(i,j)\ \mathbf{if}\ i = j\ \mathbf{then}\ A[i,j]\ \mathbf{else}\ 0\ \mathbf{end\ if}\ \mathbf{end\ proc}$:

```
> P := Matrix(3, 3, id)
> N := P − A;
> J := MatrixInverse(P).N
> n := 3 :
```

$$P := \begin{bmatrix} 3 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

$$N := \begin{bmatrix} 0 & 0 & -4 \\ -7 & 0 & -2 \\ 1 & -1 & 0 \end{bmatrix}$$

$$J := \begin{bmatrix} 0 & 0 & -\frac{4}{3} \\ -\frac{7}{4} & 0 & -\frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & 0 \end{bmatrix}$$

Vizsgáljuk meg a 4.2.3. és 4.2.4. tételek feltételeinek teljesülését.

A következő eljárás $|a_{i,i}| - \sum_{j=1}^{n} |a_{i,j}| \quad (i \neq j)$ különbséget vizsgálja. Ha ez negatív, a feltétel nem teljesül.

```
> for i to n do
      osszeg := 0;
      for j to n do
          if i ≠ j then osszeg := osszeg + |A[i, j]| end if; kulonbseg := |A[i, i]| − osszeg
      end do;
      print(cat(`Az `, i,
      . sorban az azonos sor- és oszlopindexű elem és a nem azonos sor- és oszlopindexű
      elemek összegének különbsége:, kulonbseg) );
      if kulonbseg < 0 then
          print(Az egyenletrendszer nem oldható meg Jacobi iterációval)
      end if
  end do :
```

Az 1. sorban az azonos sor- és oszlopindexű elem és a nem azonos sor- és oszlopindexű elemek összegének különbsége:-1

Az egyenletrendszer nem oldható meg Jacobi iterációval

Az 2. sorban az azonos sor- és oszlopindexű elem és a nem azonos sor- és oszlopindexű elemek összegének különbsége:-5

Az egyenletrendszer nem oldható meg Jacobi iterációval

Az 3. sorban az azonos sor- és oszlopindexű elem és a nem azonos sor- és oszlopindexű elemek összegének különbsége:0

A $\sum_{j=1}^{n} |b_{i,j}| \leq 1$ feltétel teljesülésének vizsgálatára alkalmas a következő eljárás.

```
> for i to n do
      osszeg := 0;
      for j to n do osszeg := osszeg + |J[i,j]| end do;
      print(cat(A Jacobi mátrix , i, . sor elemeinek összege:), osszeg);
      if 1 < osszeg then print(Az egyenletrendszer nem oldható meg Jacobi iterációval)
      end if
  end do :
```

$$\textit{A Jacobi mátrix 1. sor elemeinek összege:},\ \frac{4}{3}$$

Az egyenletrendszer nem oldható meg Jacobi iterációval

$$\textit{A Jacobi mátrix 2. sor elemeinek összege:},\ \frac{9}{4}$$

Az egyenletrendszer nem oldható meg Jacobi iterációval

$$\textit{A Jacobi mátrix 3. sor elemeinek összege:},\ 1$$

Tehát egyik feltétel sem teljesül. Ennek ellenére próbáljuk ki a közelítést.

```
> m[0] := Vector([0, 0, 0]) :
> k := 100 : hiba := 100 :
> for i to k while 10^(-2) < hiba do
      m[i] := evalf(MatrixInverse(P).N.m[i - 1] + MatrixInverse(P).b);
      hiba := evalf(|Norm(m[i - 1] - m[i], 2)|)
  end do:
> print(cat("m[", i-1, "]="), m[i - 1]); print(`hiba=`, hiba)
```

$$\text{"m[100]="},\ \begin{bmatrix} -1.74659832399523\ 10^{5} \\ 2.03705311494839\ 10^{5} \\ -25334.7448292696 \end{bmatrix}$$

$$\textit{hiba=},\ 3.450188941\ 10^{5}$$

```
> megoldas := solve(rendszer, {x1, x2, x3})
```

$$\textit{megoldas} := \{x_1 = 1, x_2 = 3, x_3 = -1\}$$

Látszik, hogy az eljárás során a hiba egyre növekszik, a valódi megoldáshoz semmi köze.

4. 4. Eljárások iterációra

A következő eljárások segítségével - amennyiben a Jacobi módszer alkalmazható- az egyenletrendszer **A** együtthatómátrix, **b** konstansmátrixa, elemszáma, a hibakorlát és a kiindulási vektor ismeretében tetszőleges, szabályos egyenletrendszer, megoldható.

4. 4. 1. Jacobi eljárás

```
> restart; with(LinearAlgebra) :
> Jacobi := proc(A, b, n, hibakorlat, x0)
      local id, P, N, J, osszeg, k, i, j, hiba, x, f;
      id := proc(i, j) if i = j then A[i, j] else 0 end if end proc;
      P := Matrix(n, n, id);
      N := P - A;
      J := MatrixInverse(P).N;
      f := cat(A Jacobi mátrix:);
      print(f, J);
```

```
    for i to n do
        osszeg := 0;
        for j to n do osszeg := osszeg + abs(J[i,j]) end do;
        print(cat(A Jacobi mátrix , i, . sor elemeinek összege:), osszeg)
    end do;
    for i to n do
        osszeg := 0;
        for j to n do osszeg := osszeg + abs(J[i,j]) end do;
        x[0] := Vector(x0);
        if 1 < osszeg then
            print("Az egyenletrendzser nem oldható meg Jacobi iterációval")
        else
            k := 50;
            hiba := 100;
            for i to k while hibakorlat < hiba do
                x[i] := evalf(MatrixInverse(P).N.x[i - 1] + MatrixInverse(P).b);
                hiba := evalf(abs(norm(x[i - 1] - x[i], 2)))
            end do;
            print(cat(`x[`, i - 1, `]=`), x[i - 1]);
            print(cat(`hiba=`), hiba)
        end if
    end do
end proc :
```

Az eljárást a Jacobi(**A,b**,elemszám, hibakorlát, **x0**) módon lehet megadni. Vigyázat! **x0** szögletes zárójelbe tett rendezett szám n-es, vesszővel elválasztva.

```
> rendszer := [2 x1 - x2 = 3, -x1 + 2 x2 - x3 = -4, -x2 + 2 x3 = 3]
> A, b := GenerateMatrix(rendszer, [x1, x2, x3]) :
> Jacobi(A, b, 3, 10^-1, [0, 0, 0])
```

$$rendszer := [2x_1 - x_2 = 3,\ -x_1 + 2x_2 - x_3 = -4,\ -x_2 + 2x_3 = 3]$$

$$A\ Jacobi\ mátrix:,\ \begin{bmatrix} 0 & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \end{bmatrix}$$

$$A\ Jacobi\ mátrix\ 1.\ sor\ elemeinek\ összege:,\ \frac{1}{2}$$

$$A\ Jacobi\ mátrix\ 2.\ sor\ elemeinek\ összege:,\ 1$$

$$A\ Jacobi\ mátrix\ 3.\ sor\ elemeinek\ összege:,\ \frac{1}{2}$$

$$x[11]=,\ \begin{bmatrix} 1.01562500000000 \\ -1.03125000000000 \\ 1.01562500000000 \end{bmatrix}$$

$$hiba=,\ 0.09110862336$$

4. 4. 2. Gauss-Seidel eljárás

```
> Seidel := proc(A, b, n, hibakorlat, x0)
    local id, P, N, J, osszeg, k, i, j, hiba, x;
    id := proc(i,j) if j <= i then A[i,j] else 0 end if end proc;
    P := Matrix(n, n, id);
    N := P - A;
    J := MatrixInverse(P).N;
    print(cat(A Jacobi mátrix:), J);
    for i to n do
        osszeg := 0;
        for j to n do osszeg := osszeg + abs(J[i,j]) end do;
        print(cat(A Jacobi mátrix , i, . sor elemeinek összege:), osszeg)
    end do;
    for i to n do
        osszeg := 0;
        for j to n do osszeg := osszeg + abs(J[i,j]) end do;
        x[0] := Vector(x0);
        if 1 < osszeg then
            print("Az egyenletrendszer nem oldható meg Gauss -Seidel iterációval")
        else
            k := 50;
            hiba := 100;
            for i to k while hibakorlat < hiba do
                x[i] := evalf(MatrixInverse(P).N.x[i - 1] + MatrixInverse(P).b);
                hiba := evalf(abs(norm(x[i - 1] - x[i], 2)))
            end do;
            print(cat(`x[`, i - 1, `]=`), x[i - 1]);
            print(cat(`hiba=`), hiba)
        end if
    end do
end proc :
```

```
> rendszer := [2 x1 - x2 = 3, -x1 + 2 x2 - x3 = -4, -x2 + 2 x3 = 3]
```

$$rendszer := \left[2x_1 - x_2 = 3,\ -x_1 + 2x_2 - x_3 = -4,\ -x_2 + 2x_3 = 3\right]$$

```
> A, b := GenerateMatrix(rendszer, [x1, x2, x3]) :
```

```
> Seidel(A, b, 3, 10^(-2), [0, 0, 0])
```

$$rendszer := \left[2x_1 - x_2 = 3,\ -x_1 + 2x_2 - x_3 = -4,\ -x_2 + 2x_3 = 3\right]$$

$$A\ Jacobi\ mátrix:, \begin{bmatrix} 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{4} & \frac{1}{2} \\ 0 & \frac{1}{8} & \frac{1}{4} \end{bmatrix}$$

$$A\ Jacobi\ mátrix\ 1.\ sor\ elemeinek\ összege:, \frac{1}{2}$$

$$A\ Jacobi\ mátrix\ 2.\ sor\ elemeinek\ összege:, \frac{3}{4}$$

$$A\ Jacobi\ mátrix\ 3.\ sor\ elemeinek\ összege:, \frac{3}{8}$$

$$x[7]=, \begin{bmatrix} 0.996093750000000 \\ -1.00390625000000 \\ 0.998046875000000 \end{bmatrix}$$

$hiba=, 0.005859375000$

4. 5. Gyakorlati alkalmazás

5. Példa Oldjuk meg a 3. fejezet 11. példáját Jacobi és Gauss-Seidel iterációval is.

> $rendszer := -4\,T_1 + T_2 + T_4 = -50,\ T_1 - 4\,T_2 + T_3 + T_5 = -50,\ T_2 - 4\,T_3 + T_6 = -150,\ T_1 - 4\,T_4 + T_5 + T_7 = 0,\ T_2 + T_4 - 4\,T_5 + T_6 + T_8 = 0,\ T_3 + T_5 - 4\,T_6 + T_9 = -100,\ T_4 - 4\,T_7 + T_8 = -50,\ T_5 + T_7 - 4\,T_8 + T_9 = -50,\ T_6 + T_8 - 4\,T_9 = -150 :$

> $A, b := GenerateMatrix([rendszer], [T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9]) :$

> $M := GenerateMatrix([rendszer], [T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9], augmented = true) :$

> $Jacobi(A, b, 9, 10^{-1}, [0, 0, 0, 0, 0, 0, 0, 0, 0])$

$$A\ Jacobi\ mátrix:, \begin{bmatrix} 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 \end{bmatrix}$$

$$A\ Jacobi\ mátrix\ 1.\ sor\ elemeinek\ összege:, \frac{1}{2}$$

$$A\ Jacobi\ mátrix\ 2.\ sor\ elemeinek\ összege:, \frac{3}{4}$$

$$A\ Jacobi\ mátrix\ 3.\ sor\ elemeinek\ összege:, \frac{1}{2}$$

$$A\ Jacobi\ mátrix\ 4.\ sor\ elemeinek\ összege:, \frac{3}{4}$$

$$A\ Jacobi\ mátrix\ 5.\ sor\ elemeinek\ összege:, 1$$

$$A\ Jacobi\ mátrix\ 6.\ sor\ elemeinek\ összege:, \frac{3}{4}$$

$$A\ Jacobi\ mátrix\ 7.\ sor\ elemeinek\ összege:, \frac{1}{2}$$

$$A\ Jacobi\ mátrix\ 8.\ sor\ elemeinek\ összege:, \frac{3}{4}$$

A Jacobi mátrix 9. sor elemeinek összege:, $\frac{1}{2}$

$$x[19]=, \begin{bmatrix} 32.0940290577710 \\ 49.9267578125000 \\ 67.8083146922290 \\ 28.4981864504516 \\ 49.9023437500000 \\ 71.3553291745484 \\ 32.0940290577710 \\ 49.9267578125000 \\ 67.8083146922290 \end{bmatrix}$$

hiba=, 0.08457279334

> $Seidel(A, b, 9, 10^{-1}, [0, 0, 0, 0, 0, 0, 0, 0, 0])$

$$A\ Jacobi\ mátrix:, \begin{bmatrix} 0 & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{16} & \frac{1}{4} & \frac{1}{16} & \frac{1}{4} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{64} & \frac{1}{16} & \frac{1}{64} & \frac{1}{16} & \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{16} & 0 & \frac{1}{16} & \frac{1}{4} & 0 & \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{32} & \frac{1}{16} & \frac{1}{32} & \frac{1}{8} & \frac{1}{4} & \frac{1}{16} & \frac{1}{4} & 0 \\ 0 & \frac{3}{256} & \frac{1}{32} & \frac{3}{256} & \frac{3}{64} & \frac{1}{8} & \frac{1}{64} & \frac{1}{16} & \frac{1}{4} \\ 0 & \frac{1}{64} & 0 & \frac{1}{64} & \frac{1}{16} & 0 & \frac{1}{16} & \frac{1}{4} & 0 \\ 0 & \frac{3}{256} & \frac{1}{64} & \frac{3}{256} & \frac{3}{64} & \frac{1}{16} & \frac{1}{32} & \frac{1}{8} & \frac{1}{4} \\ 0 & \frac{3}{512} & \frac{3}{256} & \frac{3}{512} & \frac{3}{128} & \frac{3}{64} & \frac{3}{256} & \frac{3}{64} & \frac{1}{8} \end{bmatrix}$$

A Jacobi mátrix 1. sor elemeinek összege:, $\frac{1}{2}$

A Jacobi mátrix 2. sor elemeinek összege:, $\frac{5}{8}$

A Jacobi mátrix 3. sor elemeinek összege:, $\frac{13}{32}$

A Jacobi mátrix 4. sor elemeinek összege:, $\frac{5}{8}$

A Jacobi mátrix 5. sor elemeinek összege:, $\frac{13}{16}$

$$\text{A Jacobi mátrix 6. sor elemeinek összege:}, \frac{71}{128}$$

$$\text{A Jacobi mátrix 7. sor elemeinek összege:}, \frac{13}{32}$$

$$\text{A Jacobi mátrix 8. sor elemeinek összege:}, \frac{71}{128}$$

$$\text{A Jacobi mátrix 9. sor elemeinek összege:}, \frac{71}{256}$$

$$x[12]= \begin{bmatrix} 32.1161542866321 \\ 49.9732971244157 \\ 67.8437914162714 \\ 28.5447256976113 \\ 49.9732971191406 \\ 71.4152199872615 \\ 32.1295057028692 \\ 49.9866485589109 \\ 67.8504671365431 \end{bmatrix}$$

$$hiba=, 0.06008136967$$

```
> T := evalf(LinearSolve(A, b, method = 'LU'))
```

$$T := \begin{bmatrix} 32.14285714 \\ 50. \\ 67.85714286 \\ 28.57142857 \\ 50. \\ 71.42857143 \\ 32.14285714 \\ 50. \\ 67.85714286 \end{bmatrix}$$

4. 6. Feladatok

1. Feladat A **H** Hilbert mátrix egy olyan mátrix, melynek elemei $H_{i,j}=\frac{1}{i+j-1}$, amely klasszikus esete a rosszul kondícionál mátrixoknak. Legyen $\mathbf{b}^T = \left[\frac{25}{12}, \frac{77}{60}, \frac{57}{60}, \frac{319}{420}\right]$,

a; Adjuk meg a 4×4-es Hilbert mátrixot, és a kondíciószám és a determináns értékének meghatározásával bizonyítsa be, hogy az egyenletrendszer gyengén kondícionált.

b; Oldjuk meg a **Hx=b** egyenletrendszert az inverz mátrix segítségével.

c; Vizsgáljuk meg az egyenletrendszer megoldását, ha a **b** mátrix értékeit 3,4,5 jegyre kerekítve adja meg.

2. Feladat Oldjuk meg az alábbi egyenletrendszereket Jacobi és Gauss-Seidel iterációval is. Mi történik a konvergenciával, ha más kezdőértéket választunk, illetve az egyenletek sorrendjét megváltoztatjuk?.

a;
$$\begin{aligned} 5\cdot x_1 - x_2 &= 9,\\ -x_1 + 5\cdot x_2 - x_3 &= 4,\\ -x_2 + 5\cdot x_3 &= -6, \end{aligned} \qquad x[0] = [0, 0, 0]$$

b;
$$\begin{aligned} 8\cdot x_1 + x_2 - x_3 &= 8,\\ 2\cdot x_1 + x_2 + 9\cdot x_3 &= 12,\\ x_1 - 7\cdot x_2 + 2\cdot x_3 &= -4, \end{aligned} \qquad x[0] = [0, 0, 0]$$

c;
$$\begin{aligned} 4\cdot x_1 - x_2 - x_4 &= 0\\ -x_1 + 4\cdot x_2 - x_3 - x_5 &= 5,\\ -x_2 + 4\cdot x_3 - x_6 &= 0,\\ -x_1 + 4\cdot x_4 - x_5 &= 6,\\ -x_3 - x_5 + 4\cdot x_6 &= 6 \end{aligned}$$

3. Feladat Megoldhatók-e a megismert numerikus módszerekkel az alábbi egyenletrendszerek?

a;
$$\begin{aligned} x_1 + 2\cdot x_2 + 3\cdot x_3 &= 1,\\ x_1 + 3\cdot x_2 + 5\cdot x_3 &= 1,\\ 3\cdot x_1 - x_2 - 4\cdot x_3 &= 1, \end{aligned}$$

b ;
$$\begin{aligned} x_1 + x_2 + x_3 - x_4 &= 4,\\ x_1 - x_2 + x_3 + x_4 &= 8,\\ 3\cdot x_1 + x_2 + 3\cdot x_3 - x_4 &= 16, \end{aligned}$$

4. 7. Ellenőrző kérdések

1. Mitől függ a lineáris egyenletrendszer megoldási módjának kiválasztása?
2. Milyen hibaforrások lépnek fel lineáris egyenletrendszer numerikus megoldása során?
3.Milyen feltételek mellett alkalmazható a Jacobi iteráció? Mi az eljárás lényege?
4. A Gauss elimináció lépései. Mi a feltétele annak, hogy az egyenletrendszernek egyértelmű megoldása legyen? Ez hogy került ide?
5. Mi a Gauss-Seidel iteráció?