

Referat 2

Tehnici și metode numerice în securitatea informației

Cod sursă: <https://github.com/danielw98/numerical-methods/>

Vizualizare site web: <https://metode-numerice.danielwagner.ro/>

Profesor universitar dr. habil. Mircea Merca
Student masterand: Ștefan-Daniel Wagner

Cuprins

1. Metoda Jacobi.....	4
Teorie.....	4
Idea metodei.....	4
Matricea de iterare.....	5
Convergență (condiții suficiente).....	5
Criterii de oprire	5
Legătura cu implementarea din proiect	5
Exemplu (sistemul 4).....	6
Iterația 1.....	6
Iterația 2.....	7
Tabel (iterațiile 0–2)	7
2. Metoda Gauss–Seidel	8
Teorie.....	8
Idea metodei.....	8
Matricea de iterare.....	8
Convergență (condiții suficiente).....	9
Criterii de oprire	9
Legătura cu implementarea din proiect	9
Exemplu (sistemul 4).....	9
Iterația 1.....	10
Iterația 2.....	10
Tabel (iterațiile 0–2)	10
3. Newton pentru sisteme (neliniare).....	11
Teorie.....	11
Metoda lui Newton pentru sisteme	11
Jacobianul	11
Criterii de oprire	11
Observații despre convergență	12
Newton pentru sisteme (exemplu)	12
Soluții (verificare rapidă)	12
Funcția vectorială și Jacobianul	12
Iterații (pas cu pas)	12

Iterația 0.....	13
Iterația 1.....	13
Iterația 2 (încă un pas)	14
Legătura cu implementarea	14

1. Metoda Jacobi

Teorie

Ideea metodei

Pentru sistemul liniar

$$Ax = b,$$

scriem descompunerea

$$A = D - L - U,$$

unde D este partea diagonală, iar L și U sunt părțile strict triunghiulare (definite cu semn schimbat), astfel încât într-adevăr $A = D - L - U$.

Atunci ecuația

$$Ax = b$$

se rescrie ca

$$(D - L - U)x = b \Rightarrow Dx = (L + U)x + b.$$

Dacă D^{-1} există (adică $a_{ii} \neq 0$ pentru orice i), rezultă forma matricială a iterației Jacobi:

$$x^{(k)} = D^{-1}(L + U)x^{(k-1)} + D^{-1}b, \quad k = 1, 2, \dots$$

Metoda **Jacobi** definește o succesiune de aproximății $\{x^{(k)}\}$ prin:

$$x^{(k)} = D^{-1}(L + U)x^{(k-1)} + D^{-1}b.$$

Pe componente (pentru $i = 1, \dots, n$):

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad a_{ii} \neq 0.$$

Observația importantă: toate componentele lui $x^{(k+1)}$ se calculează folosind doar valorile din $x^{(k)}$.

Matricea de iterație

Formula se poate scrie în forma standard

$$x^{(k+1)} = T_J x^{(k)} + c_J,$$

unde

$$T_J = D^{-1}(L + U), \quad c_J = D^{-1}b.$$

Convergență (condiții suficiente)

Metoda Jacobi converge dacă raza spectrală $\rho(T_J) < 1$.

O condiție suficientă practică (ușor de verificat) este ca A să fie **strict diagonal dominantă** pe linii:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}| \quad \text{pentru orice } i.$$

Criterii de oprire

În practică se folosește un criteriu pe diferența dintre iterații (normă infinit):

$$\frac{\|x^{(k)} - x^{(k-1)}\|_\infty}{\|x^{(k)}\|_\infty} < \varepsilon,$$

și/sau un criteriu pe reziduu:

$$\|Ax^{(k)} - b\|_\infty < \varepsilon.$$

Legătura cu implementarea din proiect

- Implementare: `JacobiSolver::iterate` în `nm-lib/src/linear/Jacobi.cpp`.
- Driver/test: `nm-lib/tests/tema3_iterative.cpp`.

Exemplu (sistemul 4)

Calculăm primele două iterații pentru metoda **Jacobi**, pornind de la

$$x^{(0)} = 0,$$

pentru sistemul (4):

$$\begin{cases} 4x_1 + x_2 - x_3 + x_4 = -2 \\ x_1 + 4x_2 - x_3 - x_4 = -1 \\ -x_1 - x_2 + 5x_3 + x_4 = 0 \\ x_1 - x_2 + x_3 + 3x_4 = 1 \end{cases}$$

Formulele de actualizare (ecuație cu ecuație, rezolvând după variabila de pe diagonală):

$$\begin{aligned} x_1 &= \frac{-2 - x_2 + x_3 - x_4}{4} \\ x_2 &= \frac{-1 - x_1 + x_3 + x_4}{4} \\ x_3 &= \frac{x_1 + x_2 - x_4}{5} \\ x_4 &= \frac{1 - x_1 + x_2 - x_3}{3} \end{aligned}$$

Valorile numerice de mai jos sunt generate cu implementarea din proiect: `nm-lib/tests/tema3_iterative.cpp` (JacobiSolver), rulat cu `--json --trace 4`.

Iterația 1

$$\begin{aligned} x_1^{(1)} &= \frac{-2}{4} = -1/2 \\ x_2^{(1)} &= \frac{-1}{4} = -1/4 \\ x_3^{(1)} &= 0 \\ x_4^{(1)} &= \frac{1}{3} \end{aligned}$$

Iterația 2

$$\begin{aligned}x_1^{(2)} &= \frac{-2 - x_2^{(1)} + x_3^{(1)} - x_4^{(1)}}{4} = -25/48 \\x_2^{(2)} &= \frac{-1 - x_1^{(1)} + x_3^{(1)} + x_4^{(1)}}{4} = -1/24 \\x_3^{(2)} &= \frac{x_1^{(1)} + x_2^{(1)} - x_4^{(1)}}{5} = -13/60 \\x_4^{(2)} &= \frac{1 - x_1^{(1)} + x_2^{(1)} - x_3^{(1)}}{3} = 5/12\end{aligned}$$

Tabel (iterațiile 0–2)

iter	x_1	x_2	x_3	x_4
0	0	0	0	0
1	-1/2	-1/4	0	1/3
2	-25/48	-1/24	-13/60	5/12

2. Metoda Gauss–Seidel

Teorie

Ideea metodei

Pentru sistemul liniar

$$Ax = b,$$

scriem

$$A = D - L - U,$$

unde D este diagonală, iar L și U sunt părțile strict triunghiulare (definite cu semn schimbat), astfel încât $A = D - L - U$.

Din

$$(D - L - U)x = b$$

obținem

$$(D - L)x = Ux + b.$$

Metoda **Gauss–Seidel** definește iterația:

$$(D - L)x^{(k)} = Ux^{(k-1)} + b, \quad x^{(k)} = (D - L)^{-1}(Ux^{(k-1)} + b), \quad k = 1, 2, \dots$$

Pe componente (pentru $i = 1, \dots, n$):

$$x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j < i} a_{ij} x_j^{(k)} - \sum_{j > i} a_{ij} x_j^{(k-1)} \right), \quad a_{ii} \neq 0.$$

Observația importantă: când calculăm $x_i^{(k+1)}$, folosim imediat valorile deja actualizate $x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}$.

Matricea de iterație

În forma standard:

$$x^{(k)} = B_{GS}x^{(k-1)} + c_{GS},$$

unde

$$B_{GS} = (D - L)^{-1}U, \quad c_{GS} = (D - L)^{-1}b.$$

Convergență (condiții suficiente)

Gauss–Seidel converge dacă $\rho(B_{GS}) < 1$.

Condiții suficiente utilizate frecvent:

- A este **strict diagonal dominantă** pe linii;
- sau A este **simetrică pozitiv definită**.

Criterii de oprire

Tipic, folosim aceleași criterii ca la Jacobi:

$$\frac{\|x^{(k)} - x^{(k-1)}\|_\infty}{\|x^{(k)}\|_\infty} < \varepsilon, \quad \|Ax^{(k)} - b\|_\infty < \varepsilon.$$

Legătura cu implementarea din proiect

- Implementare: [GaussSeidelSolver::iterate](#) în [nm-lib/src/linear/GaussSeidel.cpp](#).
- Driver/test: [nm-lib/tests/tema3_iterative.cpp](#).

Exemplu (sistemul 4)

Calculăm primele două iterații pentru metoda **Gauss–Seidel**, pornind de la

$$x^{(0)} = 0,$$

pentru sistemul (4):

$$\begin{cases} 4x_1 + x_2 - x_3 + x_4 = -2 \\ x_1 + 4x_2 - x_3 - x_4 = -1 \\ -x_1 - x_2 + 5x_3 + x_4 = 0 \\ x_1 - x_2 + x_3 + 3x_4 = 1 \end{cases}$$

Formulele de actualizare (ecuație cu ecuație, rezolvând după variabila de pe diagonală):

$$\begin{aligned} x_1 &= \frac{-2 - x_2 + x_3 - x_4}{4} \\ x_2 &= \frac{-1 - x_1 + x_3 + x_4}{4} \\ x_3 &= \frac{x_1 + x_2 - x_4}{5} \\ x_4 &= \frac{1 - x_1 + x_2 - x_3}{3} \end{aligned}$$

În Gauss–Seidel, în cadrul aceleiași iterații, folosim imediat valorile deja actualizate.

Valorile numerice de mai jos sunt generate cu implementarea din proiect: [nm-lib/tests/tema3_iterative.cpp](#) (`GaussSeidelSolver`), rulat cu `--json --trace 4`.

Iterația 1

$$\begin{aligned}x_1^{(1)} &= -1/2 \\x_2^{(1)} &= \frac{-1 - x_1^{(1)} + x_3^{(0)} + x_4^{(0)}}{4} = -1/8 \\x_3^{(1)} &= \frac{x_1^{(1)} + x_2^{(1)} - x_4^{(0)}}{5} = -1/8 \\x_4^{(1)} &= \frac{1 - x_1^{(1)} + x_2^{(1)} - x_3^{(1)}}{3} = 1/2\end{aligned}$$

Iterația 2

$$\begin{aligned}x_1^{(2)} &= \frac{-2 - x_2^{(1)} + x_3^{(1)} - x_4^{(1)}}{4} = -5/8 \\x_2^{(2)} &= \frac{-1 - x_1^{(2)} + x_3^{(1)} + x_4^{(1)}}{4} = 0 \\x_3^{(2)} &= \frac{x_1^{(2)} + x_2^{(2)} - x_4^{(1)}}{5} = -9/40 \\x_4^{(2)} &= \frac{1 - x_1^{(2)} + x_2^{(2)} - x_3^{(2)}}{3} = 37/60\end{aligned}$$

Tabel (iterațiile 0–2)

iter	x_1	x_2	x_3	x_4
0	0	0	0	0
1	-1/2	-1/8	-1/8	1/2
2	-5/8	0	-9/40	37/60

3. Newton pentru sisteme (neliniare)

Teorie

Un **sistem de ecuații neliniare** poate fi scris în forma

$$F(x) = 0,$$

unde $F: \mathbb{R}^n \rightarrow \mathbb{R}^n$ este o funcție vectorială, iar $x \in \mathbb{R}^n$ este necunoscută.

Metoda lui Newton pentru sisteme

Ideea este aceeași ca la cazul scalar, dar liniarizăm sistemul în jurul lui $x^{(k)}$ folosind Jacobianul:

$$F(x^{(k)} + \Delta x) \approx F(x^{(k)}) + J(x^{(k)}) \Delta x.$$

Impunem $F(x^{(k)} + \Delta x) = 0$ în aproximare și obținem problema liniară:

$$J(x^{(k)}) \Delta x^{(k)} = -F(x^{(k)}),$$

apoi actualizarea:

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)}.$$

Jacobianul

Jacobianul este matricea derivatelor parțiale:

$$J(x) = \left[\frac{\partial f_i}{\partial x_j}(x) \right]_{i,j=1..n}.$$

În implementare, fie îl calculăm analitic (preferabil), fie aproximăm numeric (diferențe finite) când nu avem o formulă convenabilă.

Criterii de oprire

În practică sunt folosite una sau mai multe condiții:

- **Schimbare mică a iterațiilor** (norma infinit):

$$\frac{\|x^{(k+1)} - x^{(k)}\|_\infty}{\|x^{(k+1)}\|_\infty} < \varepsilon.$$

- **Rezidual mic:**

$$\|F(x^{(k+1)})\|_\infty < \varepsilon.$$

Observații despre convergență

- Convergența este, în general, **locală**: trebuie un $x^{(0)}$ suficient de aproape de soluție.
- Dacă $J(x^{(k)})$ este (aproape) singular, metoda poate eşua sau poate deveni instabilă numeric.
- În fiecare pas trebuie rezolvat un sistem liniar cu matricea $J(x^{(k)})$.

Newton pentru sisteme (exemplu)

Luăm **sistemul (4)** (2 ecuații, 2 necunoscute):

$$\begin{aligned}3x_1^2 - x_2^2 &= 0, \\3x_1x_2^2 - x_1^3 - 1 &= 0.\end{aligned}$$

Soluții (verificare rapidă)

Din prima ecuație rezultă $x_2^2 = 3x_1^2$. Înlocuim în a doua:

$$3x_1(3x_1^2) - x_1^3 - 1 = 0 \Rightarrow 8x_1^3 - 1 = 0 \Rightarrow x_1 = \frac{1}{2}.$$

Apoi $x_2 = \pm \frac{\sqrt{3}}{2}$.

Funcția vectorială și Jacobianul

Definim $F(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix}$ cu

$$f_1(x_1, x_2) = 3x_1^2 - x_2^2, \quad f_2(x_1, x_2) = 3x_1x_2^2 - x_1^3 - 1.$$

Jacobianul este:

$$J(x_1, x_2) = \begin{bmatrix} 6x_1 & -2x_2 \\ 3x_2^2 - 3x_1^2 & 6x_1x_2 \end{bmatrix}.$$

La fiecare pas Newton rezolvă:

$$J(x^{(k)}) \Delta x^{(k)} = -F(x^{(k)}), \quad x^{(k+1)} = x^{(k)} + \Delta x^{(k)}.$$

Iterații (pas cu pas)

Pentru sistemul (4) alegem aproximarea inițială

$$x^{(0)} = (1, 2).$$

La fiecare pas calculăm $F(x^{(k)})$, $J(x^{(k)})$, rezolvăm sistemul liniar

$$J(x^{(k)}) \Delta x^{(k)} = -F(x^{(k)})$$

și apoi actualizăm $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$.

Valorile numerice de mai jos sunt cele generate de implementarea din proiect ([nm-lib/tests/tema4 newton systems.cpp](#), rulat cu `--json --trace 4`).

Iterăția 0

Punct de plecare:

$$x^{(0)} = (1, 2).$$

Evaluăm funcția și Jacobianul:

$$F(x^{(0)}) = \begin{bmatrix} -1 \\ 10 \end{bmatrix}, \quad J(x^{(0)}) = \begin{bmatrix} 6 & -4 \\ 9 & 12 \end{bmatrix}.$$

Rezolvăm sistemul liniar

$$J(x^{(0)}) \Delta x^{(0)} = -F(x^{(0)})$$

și obținem aproximativ

$$\Delta x^{(0)} \approx (-0.25926, -0.63889).$$

Prin urmare

$$x^{(1)} = x^{(0)} + \Delta x^{(0)} \approx (0.74074, 1.36111).$$

Iterăția 1

Luăm acum $x^{(1)}$ drept noua aproximație:

$$x^{(1)} \approx (0.74074, 1.36111).$$

Evaluăm din nou F și Jacobianul:

$$F(x^{(1)}) \approx \begin{bmatrix} -0.20653 \\ 2.71050 \end{bmatrix}, \quad J(x^{(1)}) \approx \begin{bmatrix} 4.44444 & -2.72222 \\ 3.91178 & 6.04938 \end{bmatrix}.$$

Rezolvăm

$$J(x^{(1)}) \Delta x^{(1)} = -F(x^{(1)})$$

și obținem

$$\Delta x^{(1)} \approx (-0.16329, -0.34247),$$

de unde

$$x^{(2)} = x^{(1)} + \Delta x^{(1)} \approx (0.57745, 1.01864).$$

Iterația 2 (încă un pas)

Repetăm procedura pentru $x^{(2)}$:

$$x^{(2)} \approx (0.57745, 1.01864).$$

Din datele generate de program:

$$F(x^{(2)}) \approx \begin{bmatrix} -0.03729 \\ 0.60498 \end{bmatrix}, \quad J(x^{(2)}) \approx \begin{bmatrix} 3.46469 & -2.03728 \\ 2.11255 & 3.52927 \end{bmatrix},$$

iar soluția sistemului liniar dă

$$\Delta x^{(2)} \approx (-0.06659, -0.13156), \quad x^{(3)} = x^{(2)} + \Delta x^{(2)} \approx (0.51085, 0.88708).$$

După câteva iterații, metoda ajunge rapid foarte aproape de soluția teoretică

$$x = (1/2, \sqrt{3}/2) \approx (0.5, 0.86603).$$

Legătura cu implementarea

Valorile numerice sunt cele obținute de implementarea din proiect.

- Program de test (definirea sistemelor + apel Newton): [nm-lib/tests/tema4_newton_systems.cpp](#)
- Solver Newton pentru sisteme: [nm-lib/include/nonlinear/Newton.h](#) și [nm-lib/src/nonlinear/Newton.cpp](#)
- Tipuri pentru sistem neliniar (F și Jacobian): [nm-lib/include/nonlinear/NonlinearSystem.h](#) și [nm-lib/src/nonlinear/NonlinearSystem.cpp](#)

Pentru sistemul (4), în cod este folosită de exemplu aproximarea inițială:

$$x^{(0)} = (1, 2).$$

Soluția numerică este apropiată de:

$$x \approx (1/2, \sqrt{3}/2) \approx (0.5, 0.8660254),$$

cu rezidual $\| F(x) \|_{\infty}$ foarte mic (sub ε).