Laborator 2

Rezolvarea sistemelor liniare determinate. Sisteme triunghiulare. Eliminarea gaussiană

2.1 Tema

Rezolvarea numerică a sistemelor liniare inferior şi superior triunghiulare prin metode de substituţie directă precum şi rezolvarea sistemelor liniare determinate dense prin metode de transformare cum este eliminarea gaussiană cu diferite strategii de pivotare.

2.2 Preliminarii

Un sistem de n ecuaţii liniare cu n necunoscute, scris în formă matriceală

$$Ax = b, \tag{2.1}$$

unde matricea $A = (a_{ij})_{i=1:m,j=1:n} \in \mathbf{R}^{n\times n}$ şi vectorul $b = (b_i)_{i=1:n} \in \mathbf{R}^n$ sunt date, iar $x = (x_j)_{j=1:n} \in \mathbf{R}^n$ este vectorul necunoscutelor se numeşte sistem liniar determinat.

Sistemul (2.1) admite a soluţie unică pentru toţi $b \in \mathbf{R}^n$ dacă şi numai dacă matricea A este nesingulară, i.e. inversabilă. Într-un astfel de caz unica soluţie a sistemului (2.1) se scrie

$$x = A^{-1}b. \tag{2.2}$$

Trebuie să subliniem că formula (2.2) *nu este recomandată* pentru calculul soluţiei sistemului $Ax = b$, după cum nu sunt recomandate şi alte metode "bine cunoscute" cum este regula lui Cramer etc.

Vom prezenta o categorie de metode fiabile şi eficiente de rezolvare a sistemelor determinate şi anume aşa numitele *metode directe*, bazate pe reducerea, printr-o secvenţă finită de transformări elementare, la rezolvarea, prin simplă substituţie, a unuia sau a două sisteme triunghiulare. În această categorie intră metode precum eliminarea gaussiană sau factorizărea LU.

2.2.1 Rezolvarea sistemelor triunghiulare

O matrice $L \in \mathbf{R}^{n\times n}$, cu $l_{ij} = 0$ pentru toţi $i < j$ se numeşte *inferior triunghiulară*. O matrice $U \in \mathbf{R}^{n\times n}$, cu $u_{ij} = 0$ pentru toţi $i > j$ se numeşte *superior triunghiulară*. Sistemul $Ax = b$ se numeşte inferior (superior) triunghiular dacă matricea A este inferior (superior) triunghiulară. O matrice triunghiulară is nesingulară dacă şi numai dacă toate elementele sale diagonale sunt nenule.

Algoritmii pentru rezolvarea sistemelor triunghiulare sunt foarte simpli, întrucât necunoscutele pot fi calculate, într-o ordine definită, prin substituţie numerică directă. Considerăm, mai întâi, un sistem inferior triunghiular

$$Lx = b,$$

unde $L \in \mathbf{R}^{n\times n}$, cu $l_{ij} = 0$ pentru $i < j$, $l_{ii} \neq 0$, $i = 1 : n$, şi $b \in \mathbf{R}^n$. Prima ecuaţie este

$$l_{11}x_1 = b_1,$$

de unde

$$x_1 = b_1/l_{11}. \tag{2.3}$$

În general, dacă ştim x_1, x_2, ..., x_{i-1}, putem rezolva ecuaţia i

$$\sum_{k=1}^{i-1} l_{ik}x_k + l_{ii}x_i = b_i$$

şi obţinem

$$x_i = \left(b_i - \sum_{k=1}^{i-1} l_{ik}x_k\right)/l_{ii}. \tag{2.4}$$

Formulele (2.3) şi (2.4) definesc un algoritm pentru calculul soluţiei sistemului $Lx = b$, cunoscut sub denumirea de metoda *substituţiei înainte*.

ALGORITM 2.1 *(S_INF_TR)* (Dată o matrice $L \in \mathbf{R}^{n\times n}$, nesingulară, inferior triunghiulară şi un vector $b \in \mathbf{R}^n$, algoritmul calculează soluţia x a sistemului $Lx = b$.)

1. **pentru** $i = 1 : n$
   1. $s \leftarrow b_i$
   2. **dacă** $i > 1$ **atunci**
      1. **pentru** $k = 1 : i - 1$
         1. $s \leftarrow s - l_{ik}x_k$
   3. $x_i \leftarrow s/l_{ii}$

Acest algoritm necesită $N_{S_INF_TR} \approx n^2$ flopi şi $M_{S\_INF\_TR} \approx n^2/2$ locaţii de memorie. Dacă vectorul b nu mai este necesar, soluţia x poate suprascrie vectorul b pe măsură ce este calculată.

Pentru a rezolva un sistem liniar superior triunghiular nesingular

$$Ux = b,$$

unde U este o matrice superior triunghiulară de ordinul n, cu $u_{ii} \neq 0$, $i = 1 : n$, şi $b \in \mathbf{R}^n$, să observăm că ultima ecuaţie este

$$u_{nn}x_n = b_n,$$

din care avem

$$x_n = b_n/u_{nn}. \tag{2.5}$$

Mai mult, dacă x_n, x_{n-1}, ..., x_{i+1}, sunt deja calculate, atunci, din ecuaţia i

$$u_{ii}x_i + \sum_{k=i+1}^{n} u_{ik}x_k = b_i$$

obţinem

$$x_i = \left(b_i - \sum_{k=i+1}^{n} u_{ik}x_k \right) / u_{ii}. \tag{2.6}$$

Din formulele (2.5) şi (2.6), care definesc aşa numita *substituţie înapoi*, obţinem următorul algoritm.

ALGORITM 2.2 *(S_SUP_TR)* (Dată o matrice $n \times n$ reală U, superior triunghiulară, nesingulară şi un vector $b \in \mathbf{R}^n$, acest algoritm calculează soluţia x a sistemului $Ux = b$.)

1. **pentru** $i = n : -1 : 1$
   1. $s \leftarrow b_i$
   2. **dacă** $i < n$ **atunci**
      1. **pentru** $k = i + 1 : n$
         1. $s \leftarrow s - u_{ik}x_k$
   3. $x_i \leftarrow s/u_{ii}$

Ca şi în cazul inferior triunghiular, efortul de calcul este dat de $N_{S\_SUP\_TR} = n^2$ flopi şi memoria necesară este $M_{S\_SUP\_TR} \approx n^2/2$ locaţii.

2.3 Eliminarea gaussiană

Eliminarea gaussiană este o tehnică pentru reducerea sistemului $Ax = b$ la un sistem superior triunghiular echivalent

$$MAx = Mb$$

, i.e. cu matricea $U = MA$ superior triunghiulară. Matricea de transformare M este o matrice inferior triunghiulară unitară obţinută ca o secvenţă (produs) de transformări (matrice) inferior triunghiulare elementare

$$M = M_{n-1}M_{n-2}\dots M_1.$$

O matrice inferior triunghiulară elementară de ordin n şi indice k este o matrice de forma

$$M_k = I_n - m_k e_k^T, \quad m_k = [0\ 0\ \cdots 0 \mu_{kk}\ \mu_{k+1,k}\cdots\mu_{nk}]^T.$$

Triangularizarea de mai sus există dacă toate submatricele lider principale $A^{[k]} = A(1:k, 1:k)$ sunt nesingulare, iar procesul de triangularizare are loc conform schemei de calcul

1. **pentru** $k = 1 : n-1$
   1. Se determină matricea ITE M_k, i.e. scalarii μ_{ik}, $i = k+1 : n$ astfel încât $(M_kA)_{ik} = 0$, $i = k+1 : n$
   2. $A \leftarrow M_kA$.

Scalarii μ_{ik}, numiţi *multiplicatori gaussieni*, care asigură satisfacerea condiţiei de la 1.1 au expresia

$$\mu_{ik} = a_{ik}/a_{kk}, \quad i = k+1 : n,$$

şi pot fi memoraţi în locul zerourilor create. Operaţia $A \leftarrow M_kA$ se execută eficient dacă ţinem seama că primele $k-1$ coloane ale lui A nu sunt afectate, coloana k este afectată într-un mod cunoscut, iar coloanele a_j, $j \in k+1 : n$ suferă transformarea

$$(M_ka_j)_i = ((I_n - m_ke_k^T)a_j)_i = (a_j - m_ka_{kj})_i = a_{ij} - \mu_{ik}a_{kj}, \quad i = k+1 : n.$$

Cele prezentate mai sus definesc algoritmul G de eliminare gaussiană descris mai jos.

ALGORITM 2.3 *(G)* (Dată o matrice $A \in \mathbf{R}^{n\times n}$ cu submatricele lider principale $A^{[k]}$, $k = 1 : n-1$, nesingulare, acest algoritm suprascrie triunghiul superior al lui A, inclusiv diagonala, cu matricea superior triunghiulară $U = M_{n-1}M_{n-2}\dots M_1A$. Triunghiul inferior al lui A este suprascris cu multiplicatorii gaussieni μ_{ik} care definesc matricele ITE M_k, $k = 1 : n-1$.)

1. **pentru** $k = 1 : n-1$
   1. **pentru** $i = k+1 : n$
      1. $a_{ik} \leftarrow \mu_{ik} = a_{ik}/a_{kk}$
   2. **pentru** $i = k+1 : n$
      1. **pentru** $j = k+1 : n$
         1. $a_{ij} \leftarrow a_{ij} - \mu_{ik}a_{kj}$

Efortul de calcul este $N_G \approx \frac{2n^3}{3}$ flopi, în timp ce memoria necesară este $M_G = n^2$ locaţii.

Observaţie. Nesingularitatea submatricelor lider principale ale matricei A nu este o condiţie necesară pentru existenţa şi unicitatea soluţiei unui sistem de forma $Ax = b$. Pentru a elimina această condiţie vom introduce pivotarea.

2.3.1 Strategii de pivotare

Singularitatea submatricelor lider principale $A^{[k]}$ este echivalentă cu anularea elementului a_{kk}, numit pivot, la pasul k al algoritmului G. Ca urmare, calculul multiplicatorilor gaussieni nu se poate face. Într-un astfel de caz, este necesară aducerea pe poziţia pivotului a unui element nenul, preferabil de modul cât mai mare, prin permutare de linii şi/sau coloane.

Întrucât interschimbarea a două linii ale unei matrice este echivalentă cu premultiplicarea acelei matrice cu o matrice de permutare elementară iar interschimbarea a două coloane este echivalentă cu postmultiplicarea matricei cu o matrice de permutare elementară, întregul proces de "pivotare" poate fi descris în termeni matriceali într-unul din două următoare moduri.

Pivotare parţială

Pivotarea parţială are loc prin permutarea numai a liniilor. La pasul k al algoritmului G se aduce în poziţia (k, k) a pivotului cel mai mare element în modul dintre elementele subdiagonale din coloana k, fie acesta $a_{i_k k} \neq 0$, prin permutarea liniilor k şi i_k. Acest lucru este echivalent cu premultiplicarea matricei A cu matricea de permutare elementară $P_{i_k k} \overset{\text{not}}{=} P_k$, astfel încât pasul k realizează

$$A \leftarrow M_k P_k A,$$

iar întregul algoritm realizează triangularizarea matricii A prin secvenţa

$$A \leftarrow U = M_{n-1}P_{n-1}M_{n-2}P_{n-2}\ldots M_1 P_1 A.$$

Algoritmul corespunzător este dat mai jos.

ALGORITM 2.4 *(GPP – eliminarea gaussiană cu pivotare parţială)* (Dată o matrice $A \in \mathbf{R}^{n\times n}$, algoritmul suprascrie triunghiul superior al lui A cu matricea superior triunghiulară $U = M_{n-1}P_{n-1}\dots M_1P_1A$. Triunghiul strict inferior al lui A este suprascris de multiplicatorii gaussieni μ_{ik} ce definesc matricele ITE M_k, $k = 1 : n-1$, iar întregii i_k, definind permutările de linii, sunt memoraţi într-un vector p, astfel încât $p(k) = i_k$, pentru $k = 1 : n-1$.)

1. **pentru** $k = 1 : n-1$
   1. Se determină primul $i_k \in k : n$ astfel încât $|a_{i_kk}| = \max_{i=k:n} |a_{ik}|$.
   2. $p(k) \leftarrow i_k$
   3. **pentru** $j = k : n$
      1. $a_{kj} \leftrightarrow a_{i_kj}$
   4. **pentru** $i = k+1 : n$
      1. $a_{ik} \leftarrow \mu_{ik} = a_{ik}/a_{kk}$
   5. **pentru** $i = k+1 : n$
      1. **pentru** $j = k+1 : n$
         1. $a_{ij} \leftarrow a_{ij} - \mu_{ik}a_{kj}$

Complexitatea algoritmului GPP este aceeaşi cu cea a algoritmului G.

Pivotare completă

Proprietăţi numerice mai bune se obţin dacă pivotul de la pasul k se alege drept cel mai mare element în modul dintre elementele a_{ij}, cu $i = k : n$, $j = k : n$. Acest cel mai mare element în modul, fie el $a_{i_kj_k}$, este adus în poziţia (k, k) a pivotului prin permutarea liniilor k şi i_k şi a coloanelor k şi j_k. Acest lucru este echivalent cu premultiplicarea matricei A cu matricea de permutare elementară $P_{i_kk} \overset{\text{not}}{=} P_k$ şi cu postmultiplicarea cu matricea de permutare elementară $Q_{j_kk} \overset{\text{not}}{=} Q_k$, astfel încât pasul k realizează

$$A \leftarrow M_kP_kAQ_k,$$

iar întregul algoritm realizează triangularizarea matricii A prin secvenţa

$$A \leftarrow U = M_{n-1}P_{n-1}\dots M_2P_2M_1P_1AQ_1Q_2\cdots Q_{n-1}.$$

Algoritmul corespunzător, de eliminare gaussiană cu pivotare completă, este dat mai jos.

ALGORITM 2.5 *(GPC – eliminarea gaussiană cu pivotare completă)* (Dată a matrice $A \in \mathbf{R}^{n\times n}$, acest algoritm suprascrie triunghiul superior al lui A cu matricea superior triunghiulară U. Triunghiul strict inferior al lui A este suprascris cu multiplicatorii gaussieni μ_{ik}, $k = 1 : n-1$, $i = k+1 : n$. Întregii i_k şi j_k, ce definesc permutările de linii, respectiv coloane, sunt memoraţi în doi vectori p şi q, astfel încât $p(k) = i_k$ şi $q(k) = j_k$, pentru $k = 1 : n-1$.)

1. **pentru** $k = 1 : n-1$
   1. Se determină $i_k \in k : n$ şi $j_k \in k : n$ astfel încât $|a_{i_k j_k}| = \max_{i=k:n, j=k:n} |a_{ij}|$.
   2. $p(k) \leftarrow i_k$
   3. $q(k) \leftarrow j_k$
   4. **pentru** $j = k : n$
      1. $a_{kj} \leftrightarrow a_{i_k j}$
   5. **pentru** $i = 1 : n$
      1. $a_{ik} \leftrightarrow a_{ij_k}$
   6. **pentru** $i = k+1 : n$
      1. $a_{ik} \leftarrow \mu_{ik} = a_{ik}/a_{kk}$
   7. **pentru** $i = k+1 : n$
      1. **pentru** $j = k+1 : n$
         1. $a_{ij} \leftarrow a_{ij} - \mu_{ik} a_{kj}$

Complexitatea algoritmului GPC este aceeaşi cu cea a algoritmului G.

2.4 Rezolvarea sistemelor liniare determinate

Aşa cum s-a precizat, algoritmii GPP şi GPC stau la baza unei metode de rezolvare a sistemelor liniare nesingulare $Ax = b$. Deşi din punct de vedere numeric algoritmul GPC este mai bun, GPP is suficient de bun pentru necesităţile practice curente şi, de aceea, este folosit pe scară largă.

Admiţând că desfacem matricea A rezultată din algoritmii GPP şi GPC în matricele componente: matricea superior triunghiulară U şi matricea strict inferior triunghiulară M a multiplicatorilor gaussieni, introducem următoarele sintaxe de utilizare a algoritmilor GPP şi GPC

$$[\, M, U, p \,] = \mathrm{GPP}(A) \qquad [\, M, U, p, q \,] = \mathrm{GPC}(A).$$

Considerăm sistemul liniar $Ax = b$, cu A nesingulară. Evident, sistemul $Ax = b$ este echivalent cu sistemul

$$M_{n-1} P_{n-1} \ldots M_1 P_1 A x = M_{n-1} P_{n-1} \ldots M_1 P_1 b,$$

unde M_k, P_k sunt matricele utilizate de GPP, i.e. cu sistemul superior triunghiular

$$Ux = M_{n-1} P_{n-1} \ldots M_1 P_1 b,$$

Deci, pentru a rezolva $Ax = b$, trebuie să calculăm vectorul

$$b \leftarrow d = M_{n-1} P_{n-1} \ldots M_1 P_1 b,$$

folosind următoarea schemă:

1. **pentru** $k = 1 : n-1$
   1. $b \leftarrow P_k b$
   2. $b \leftarrow M_k b$.

Algoritmul corespunzător este dat mai jos.

ALGORITM 2.6 *(SL_GPP – Rezolvarea sistemelor liniare cu GPP)* (Dată o matrice nesingulară $A \in \mathbf{R}^{n\times n}$ şi un vector $b \in \mathbf{R}^n$, acest algoritm calculează soluţia $x \in \mathbf{R}^n$ sistemului liniar $Ax = b$, folosind eliminarea gaussiană cu pivotare parţială.)

1. $[M, U, p] = \text{GPP}(A)$
2. **pentru** $k = 1 : n-1$
   1. $b_k \leftrightarrow b_{p(k)}$
   2. **pentru** $i = k+1 : n$
      1. $b_i \leftarrow b_i - \mu_{ik} b_k$
3. $x = \text{S\_SUP\_TR}(U, b)$

Complexitatea algoritmului este dată de

$$N_{SL\_GPP} = N_{GPP} + \sum_{k=1}^{n-1} 2(n-k) + N_{UTRIS} \approx \frac{2n^3}{3} + n^2 + n^2 \approx \frac{2n^3}{3} \approx N_{GPP}.$$

În cazul utilizării algoritmului GPC sistemul $Ax = b$ este echivalent cu sistemul

$$M_{n-1}P_{n-1} \dots M_1 P_1 A Q_1 \dots Q_{n-1} Q_{n-1} \dots Q_1 x = M_{n-1}P_{n-1} \dots M_1 P_1 b.$$

Notând

$$Q_{n-1} \dots Q_1 x = y,$$

rezultă că y poate fi calculat ca soluţie a sistemului superior triunghiular $Uy = d$, unde $d = M_{n-1}P_{n-1} \dots M_1 P_1 b$. Din expresia lui y avem

$$x = Q_1 Q_2 \dots Q_{n-1} y$$

care poate fi calculat cu schema

1. $x \leftarrow y$
2. **pentru** $k = n-1 : -1 : 1$
   1. $x \leftarrow Q_k x$.

Rezultă algoritmul.

ALGORITM 2.7 *(SL_GPC – rezolvarea sistemelor liniare cu GPC)* (dată o matrice nesingulară $A \in \mathbf{R}^{n\times n}$ şi un vector $b \in \mathbf{R}^n$, acest algoritm calculează soluţia $x \in \mathbf{R}^n$ a sistemului liniar $Ax = b$, folosind eliminarea gaussiană cu pivotare completă.)

1. $[\,M, U, p, q\,] = \mathrm{GPC}(A)$
2. **pentru** $k = 1 : n-1$
    1. $b_k \leftrightarrow b_{p(k)}$
    2. **pentru** $i = k+1 : n$
        1. $b_i \leftarrow b_i - \mu_{ik} b_k$
3. $x = \mathrm{S\_SUP\_TR}(U, b)$
4. **pentru** $k = n-1 : -1 : 1$
    1. $x_k \leftrightarrow x_{q(k)}$

2.5 Sarcini de lucru

2.5.1 A. In laborator

1. Se elaborează şi editează programe, sub forma unor funcţii MATLAB, pentru implementarea algoritmilor de rezolvare a sistemelor liniare inferior şi superior triunghiulare.

2. Se testează corectitudinea algoritmilor şi programelor pentru sisteme triunghiulare cu coeficienţi aleatori de 7 şi 25 de ecuaţii.

3. Se elaborează şi editează un program, sub forma unei funcţii MATLAB, pentru implementarea algoritmului G de triangularizare prin eliminare gaussiană şi se testează corectitudinea algoritmului şi programului pentru matrice cu elemente aleatoare de ordinul 7 şi 25. Ce se întâmplă dacă înainte de apelul funcţiei executaţi instrucţiunea `A(1,1:2)=A(2,1:2)`?

4. Se elaborează şi editează un program, sub forma unei funcţii MATLAB, pentru implementarea algoritmului GPP de triangularizare prin eliminare gaussiană cu pivotare parţială şi se testează corectitudinea algoritmului şi programului pentru matrice cu elemente aleatoare de ordinul 7 şi 25.

5. Se elaborează şi editează un program, sub forma unei funcţii MATLAB, pentru implementarea algoritmului de rezolvare a sistemelor liniare $Ax = b$ folosind programul GPP de triangularizare prin eliminare gaussiană cu pivotare parţială şi se testează corectitudinea algoritmului şi programului pentru sisteme liniare de 7 şi 25 de ecuaţii cu 7, respectiv 25 de necunoscute.

2.5.2 B. Acasă

1 Se elaborează şi editează programe, sub forma unor funcţii MATLAB, pentru implementarea algoritmilor de rezolvare a sistemelor liniare inferior şi superior bidiagonale şi se testează corectitudinea lor pentru sisteme bidiagonale cu coeficienţi aleatori de 100 şi 500 de ecuaţii.

2 Se elaborează şi editează un program, sub forma unei funcţii MATLAB, pentru implementarea algoritmului GPC de triangularizare prin eliminare gaussiană cu pivotare parţială şi se testează corectitudinea algoritmului şi programului pentru matrice cu elemente aleatoare de ordinul 100 şi 500.

3 Se elaborează şi editează un program, sub forma unei funcţii MATLAB, pentru implementarea algoritmului de rezolvare a sistemelor liniare $Ax = b$ folosind programul GPC de triangularizare prin eliminare gaussiană cu pivotare completă şi se testează corectitudinea algoritmului şi programului pentru sisteme liniare de 100 şi 500 de ecuaţii cu 100, respectiv 500 de necunoscute.