

Operații cu matrice în MATLAB. Rezolvarea eficientă a sistemelor de ecuații liniare. Factorizări LU.

Colaboratori: Andrei STAN, Mădălina Hristache

February 22, 2025

Cuprins

1	Obiective laborator	1
2	Noțiuni teoretice	1
2.1	Complexitatea regulii Cramer	1
2.2	Factorizarea LU	2
2.2.1	Descompunerea matricei în L și U	2
2.2.2	Rezolvarea sistemelor triunghiulare	5
3	Probleme	7

1 Obiective laborator

În urma parcurgerii acestui laborator, studentul va fi capabil să:

- factorizeze o matrice folosind una dintre metodele LU: Crout, Doolittle, Cholesky;
- rezolve recursiv un sistem triunghiular;

2 Noțiuni teoretice

2.1 Complexitatea regulii Cramer

La liceu sistemele de ecuații se rezolvau folosind *regula lui Cramer*. În continuare, demonstrăm de ce această abordare, este inefficientă computațional.

Fie un sistem de n ecuații cu n necunoscute, $Ax = b$, cu $\det(A) \neq 0$. Folosind regula lui Cramer urmează să calculăm n determinanți pentru fiecare necunoscută, înlocuind pe rând coloane din A cu b . În total, calculăm $n + 1$ determinanți.

Determinantul unei matrici se poate calcula în două moduri:

$$\det(A) = \sum_{i=1}^n (-1)^{i+j} a_{ij} \det(A_{ij}), \quad \forall j \in \{1, 2, \dots, n\} \quad (1)$$

$$\det(A) = \sum_{j=1}^n (-1)^{i+j} a_{ij} \det(A_{ij}), \quad \forall i \in \{1, 2, \dots, n\} \quad (2)$$

Cu A_{ij} am notat matricea de dimensiune $(n-1) \times (n-1)$ rezultată din suprimarea (eliminarea) liniei i și a coloanei j . Cu alte cuvinte, pentru calcularea unui determinant de ordin n trebuie să calculăm n determinanți de ordin $n-1$. În total, pentru calcularea determinantului de ordin n am avea:

$$n \cdot (n-1) \cdot (n-2) \cdot \dots \cdot 1 = n! \quad (3)$$

În cazul unui sistem de n ecuații cu n necunoscute, folosind ecuația 3, avem o complexitate de ordinul $O(n!)$. Pe mai departe, vom explora metode mai eficiente de calcul, de ordinul $O(n^3)$.

2.2 Factorizarea LU

Factorizarea (sau descomponerea) unei matrici are aplicabilitate foarte în analiza numerică. Pentru rezolvarea sistemelor liniare, vom discuta despre factorizarea LU și QR.

Factorizarea LU presupune descompunerea unei matrici pătratice A într-un produs de două matrici, L și U , unde L este o matrice inferior triunghiulară, iar U este o matrice superior triunghiulară. Astfel, putem scrie $A = LU$. Astfel sistemul de ecuații $Ax = b$ se transformă în două sisteme:

$$Ly = b$$

$$Ux = y$$

Aceste sisteme se numesc *triunghiulare* și se pot rezolva în $O(n^2)$. Astfel avem 2 pași de făcut:

1. Descompunerea matricii A în L și U ;
2. Rezolvarea sistemelor triunghiulare.

2.2.1 Descompunerea matricii în L și U

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix}$$

Prin efectuarea deescompunerii direct, ar rezulta un sistem cu n^2 ecuații și $n^2 + n$ necunoscute. Putem ”scăpa” de necunoscutele în plus folosind mai multe metode, cele mai cunoscute fiind *Crout*, *Doolittle* și *Cholesky*.

Metoda Crout

Metoda Crout presupune ca toate elementele de pe diagonala matricii U să fie egale cu 1. Astfel, pentru o matrice 3×3 , putem scrie sistemul de ecuații:

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

Algoritmul în MATLAB pentru Crout poate fi gândit astfel:

1. Folosesc un indice p cu care mă plimb pe coloane;
2. Observăm că pentru fiecare coloană avem două seturi de ecuații:
 - Din primele $p - 1$ ecuații putem calcula $u_{ip}, \forall j \in \{1, 2, \dots, p - 1\}$;
 - Din restul, putem calcula $l_{ip}, \forall j \in \{p, p + 1, \dots, n\}$.

```

A = [14 8 3; 8 5 2; 3 2 1];

n = size(A)(1);

L = zeros(size(A));
U = eye(size(A));

for p = 1 : n
    for i = 1 : p - 1
        U(i, p) = (A(i, p) - L(i, 1 : i) * U(1 : i, p)) / L(i, i);
    end

    for i = p : n
        L(i, p) = A(i, p) - L(i, 1 : i) * U(1 : i, p);
    end
end

disp(L);
disp(U);
disp(L * U);

```

Metoda Doolittle

Metoda Doolittle presupune ca toate elementele de pe diagonala matricei L să fie egale cu 1. Astfel, pentru o matrice 3×3 , putem scrie sistemul de ecuații:

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

Algoritmul în MATLAB pentru Doolittle poate fi gândit astfel:

1. Folosesc un indice p cu care mă plimb pe linii;
2. Observăm că pentru fiecare linie avem două seturi de ecuații:
 - Din primele $p - 1$ ecuații putem calcula $l_{pi}, \forall i \in \{1, 2, \dots, p - 1\}$;
 - Din restul, putem calcula $u_{pi}, \forall i \in \{p, p + 1, \dots, n\}$.

```

A = [14 8 3; 8 5 2; 3 2 1];

n = size(A)(1);

L = eye(size(A));
U = zeros(size(A));

for p = 1 : n
    for i = 1 : p - 1
        L(p, i) = (A(p, i) - L(p, 1 : (i - 1)) * U(1 : (i - 1), i)) / U(i, i);
    end

    for i = p : n
        U(p, i) = A(p, i) - L(p, 1 : (i - 1)) * U(1 : (i - 1), i);
    end
end

disp(L);
disp(U);
disp(L * U);

```

Metoda Cholesky

Descompunerea Cholesky se remarcă prin faptul că matricea U este setată ca fiind transpusa (sau hermitica) matricei L , adică $A = LL^*$.

Fie o matrice A simetrică de dimensiune $n \times n$.

$$A \text{ pozitiv - definită} \Leftrightarrow \mathbf{x}^T A \mathbf{x} > 0, \forall \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$$

În același mod se poate defini și conceptul de matrice *negativ-definită*, înlocuind semnul $>$ cu $<$.

Descompunerea Cholesky se poate aplica doar pe matrici simetrice, pozitiv-definite.

Demonstrație. Fie A o matrice oarecare.

$$A = LL^* \implies A^* = LL^* \implies A = A^*$$

Demonstrăm acum că A trebuie să fie pozitiv-definită. Fie $Ax = b$:

$$\begin{aligned}
 Ax &= b \\
 x^* Ax &= x^* b \\
 x^* LL^* x &= x^* b, \quad y := L^* x \\
 y^* y &= x^* b
 \end{aligned}$$

Cum $y^* y > 0 \quad \forall y \neq 0$, rezultă că $x^* b > 0$ și deci $x^* Ax > 0$. ■

Revenind, pentru o matrice 3×3 , putem scrie sistemul de ecuații:

$$\begin{array}{lll} l_{11}^2 = a_{11} & l_{11}l_{21} = a_{12} & l_{11}l_{31} = a_{13} \\ l_{11}l_{21} = a_{21} & l_{21}^2 + l_{22}^2 = a_{22} & l_{21}l_{31} + l_{22}l_{32} = a_{23} \\ l_{11}l_{31} = a_{31} & l_{21}l_{31} + l_{22}l_{32} = a_{32} & l_{31}^2 + l_{32}^2 + l_{33}^2 = a_{33} \end{array}$$

Algoritmul în MATLAB pentru Cholesky poate fi gândit astfel:

1. Folosesc un indice p cu care mă plimb pe linii și cu i pe coloane;
2. Observăm că avem 2 tipuri de ecuații:
 - $i = p$: Putem calcula $l_{pp} = \sqrt{a_{pp} - \sum_{j=1}^i l_{pj}^2}$;
 - $i \neq p$: Putem calcula $l_{pi} = \frac{a_{pi} - \sum_{j=1}^i l_{pj}l_{ij}}{l_{ii}}$.
3. Pentru că matricea A este simetrică, putem ignora partea de deasupra diagonalei principale a sistemului.
4. Se observă că cele două sume sunt echivalente atunci când $p = i$.

```
A = [14 8 3; 8 5 2; 3 2 1];

n = size(A)(1);

L = zeros(n);

for p = 1 : n
    for i = 1 : p
        s = L(p, 1 : i) * L(i, 1 : i)';

        if i == p
            L(p, p) = sqrt(A(p, p) - s);
        else
            L(p, i) = (A(p, i) - s) / L(i, i);
        end
    end
end

disp(L);
disp(L * L');
```

2.2.2 Rezolvarea sistemelor triunghiulare

Sisteme triunghiulare pot fi de 2 tipuri, *superioare* sau *inferioare*, în funcție de tipul matricei. Tratăm prima data cazul sistemelor superior triunghiulare.

$$\begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{nn}x_n = b_n \end{array}$$

Pentru a rezolva un sistem superior triunghiular, putem folosi metoda *substituției înapoi*:

$$x_i = \frac{b_i - \sum_{j=i+1}^n a_{ij}x_j}{a_{ii}}, \quad \forall i \in \{n, n-1, \dots, 1\} \quad (4)$$

```
A = [14 8 3; 0 5 2; 0 0 1];
b = [1; 2; 3];

n = size(A)(1);

x = zeros(n, 1);

for i = n : -1 : 1
    x(i) = (b(i) - A(i, (i + 1) : n) * x((i + 1) : n)) / A(i, i);
end

disp(x);
disp(A * x);
```

Sistemele inferior triunghiulare arată similar:

$$\begin{aligned} a_{11}x_1 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \\ &\vdots \\ a_{n1}x_1 + \dots + a_{nn}x_n &= b_n \end{aligned}$$

Pentru a rezolva un sistem inferior triunghiular, putem folosi metoda *substituției înainte*:

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j}{a_{ii}}, \quad \forall i \in \{1, 2, \dots, n\} \quad (5)$$

```
A = [14 0 0; 8 5 0; 3 2 1];
b = [1; 2; 3];

n = size(A)(1);

x = zeros(n, 1);

for i = 1 : n
    x(i) = (b(i) - A(i, 1 : (i - 1)) * x(1 : (i - 1))) / A(i, i);
end

disp(x);
disp(A * x);
```

3 Probleme

1. Pentru matricea dată mai jos, determinați matricele L și U folosind metoda Crout.

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 11 \\ 3 & 22 & 42 \end{bmatrix}$$

2. Pentru matricea de la exercițiul anterior aplicați factorizarea Doolittle.
3. Scrieți două funcții în MATLAB care să rezolve un sistem de ecuații superior triunghiular, respectiv inferior triunghiular. Folosiți următoarele prototipuri:

```
function x = superior(U, b)  
function x = inferior(U, b)
```