

Proiect Metode Numerice

Tema: Rezolvarea Sistemelor de Ecuații Lineare $Ax = b$

**Autor: Juncu Alexandru
(312CA)**

Continut:

1. Metoda Gauss-Jordan.
2. Calculul matricei inverse
3. Factorizare Crout
4. Metoda Suprarelaxării successive
5. Legile lui Kirchhoff
6. Modelul Leontief
7. Metoda Simplex de Programare Lineară

Introducere

Rezolvarea sistemelor cu un mare numar de ecuatii liniare reprezinta unul din domeniile in care calculatoarele numerice si-au dovedit din plin eficienta. Problema rezolvarii sistemelor de ecuatii liniare este foarte des intalnita in simularea numerica. Enumeram cateva situatii: interpolare cu functii spline cubice, rezolvarea sistemelor de ecuatii neliniare cu ajutorul metodelor iterative care au la baza liniarizarea ecuatiilor, discretizarea ecuatiilor diferentiale ordinare cu conditii la limita, discretizarea ecuatiilor cu derivate parțiale. In mod corespunzator, a trebuit sa fie puse la punct procedee numerice adecvate, atat pentru reducerea numarului mare de operatii, cat si pentru reducerea erorilor de calcul care cresc cu dimensiunile sistemului de ecuatii.

$$\sum_{j=1}^n a_{i,j} x_j = b_i \quad i=1,m \quad (1)$$

unde $a_{i,j} \in \mathbf{R}$ sunt coeficienti, x_j , $j \in 1,n$ sunt necunoscutele sistemului, iar b_i sunt termenii liberi. Vom distinge trei situatii :

- 1) Pentru $m < n$ sistemul este *subdeterminat*, avem mai putine ecuatii decat necunoscute. In general, vor trebui alesi $n-m$ parametrii pentru a obtine o solutie.
- 2) Pentru $m = n$ si $\det(A) \neq 0$ sistemul este *compatibil determinat*. Sistemul are o solutie unica. Este cazul cel mai des intalnit. Pentru $m = n$ si $\det(A) = 0$ sistemul poate fi *compatibil nedeterminat*, cu o infinitate de solutii posibile, sau *incompatibil*, cu nici o solutie.

- 3) Pentru $m > n$ sistemul este *supradeterminat*, caz in care se cauta o solutie care sa verifice “cel mai bine” ecuatiile (1) in sensul minimizarii

$$\text{reziduului } R = \sum_{i=1}^m (b_i - \sum_{j=1}^n a_{i,j} x_j)^2$$

Nu trebuie uitat ca, la reprezentarea numerelor in calculator nu putem retine decat un numar finit de cifre, iar erorile de rotunjire se propaga. Se poate

ajunge în situația că din cauza acestor erori, determinantul să devină egal cu zero sau să aibă o valoare foarte mică în modul. Complexitatea implementărilor prezente în bibliotecile matematice se datorează în mare parte tehnicilor de evitare a acestei situații, dar totodată și eforturilor de minimizare a memoriei utilizate și a timpului de calcul. Metodele de rezolvare a sistemelor de ecuații liniare sunt de două tipuri: (a) *metode directe* (sau *metode de eliminare* sau *metode exacte*), în care soluția este obținută în urma unui număr de operații dinainte cunoscut; (b) *metode iterative*, care se bazează pe folosirea unei aproximații inițiale ce se îmbunătățește de la o etapă la alta.

Metoda Gauss-Jordan

Inversarea unei matrici ar presupune rezolvarea a n sisteme de ecuații liniare. Pornim de la

$$A X = I_n \quad \Rightarrow \quad X = A^{-1}$$

Dezvoltând pe coloane obținem

$$A [x_1 \ x_2 \ \dots \ x_n] = [e_1 \ e_2 \ \dots \ e_n] \quad (2)$$

$$A x_i = e_i \quad i=1:n$$

Matricea inversă ar avea drept coloane soluțiile sistemelor având ca termeni liberi coloanele matricii unitate.

Metoda este inacceptabilă datorită complexității ei $O(n^4)$ dar complexitatea poate fi redusă observând că sistemele (2) au toate aceeași matrice A , deci ar putea fi rezolvate simultan, adică matricea A ar fi triunghiularizată o singură dată. În acest mod complexitatea devine $O(n^3)$.

Vom considera:

$$B = [A \mid I_n] \text{ cu } B \in \mathbb{R}^{n \times 2n}$$

Transformarea se efectuează progresiv, într-un pas se modifică o singură coloană dintre primele n ale matricii B și constă în :

- normalizare:

$$A_{p,j} = \frac{A_{p,j}}{A_{p,p}} \quad p=1:n \quad j=p:2n$$

- reducere:

$$A_{i,j} = A_{i,j} - A_{i,p} * A_{p,j} \quad p=1:n \quad i=1:n \quad j=p:2n$$

Algoritm Gauss-Jordan:

```

X=Gauss_Jordan(n,A)
{
    pentru i=1:n
    {
        X(i,1:n)=0
        X(i,i)=1
    }
    pentru p=1:n-1
        pentru i=p+1:n
        {
            t=A(i,p)/ A(p,p)
            A(i,p)=0
            A(i,p+1:n)= A(i,p+1:n)-t* A(p,p+1:n)
            X(i,1:n)= X(i,1:n)- t* X(p,1:n)
        }
    pentru i=n:-1:1
        pentru k=1:n
        {
            suma=A(i,i+1:n)*X(i+1:n,k)
            X(i,k)= ( X(i,k) -suma )/ A(i,i)
        }
    }
}

```

Calculul Matricei Inverse

Inversarea unei matrice patrata si nesingurlaer este o operatie aproximativ de 3 ori mai costisitoare decat rezolvarea unui system liniar de acelasi ordin. Complexiatatea operatiei de inversare este in general $O(n^3)$.

Metodele directe de inversare se bazează fie pe factorizarea matricii, reducând problema la cea a inversării a două matrici triunghiulare fie pe diagonalizarea matricii.

1. Inversarea matricilor triunghiulare

Inversa unei matrici triunghiulare este tot o matrice triunghiulară de aceeași formă a cărei elemente se calculează explicit din relația

$$A B = I_n \\ \Rightarrow B = A^{-1}$$

Pentru o matrice superior triunghiulară relația precedentă scrisă pe componente are forma

$$b_{j,j} = \frac{1}{a_{j,j}} \quad j=1:n$$

$$b_{i,j} = \frac{-1}{a_{i,j}} \sum_{k=i+1}^j a_{i,k} * b_{k,j} \quad i=j-1:-1:1$$

O matrice inferior triunghiulară, relațiile sunt:

$$b_{i,i} = \frac{1}{a_{i,i}} \quad i=1:n$$

$$b_{i,j} = \frac{-1}{a_{i,i}} \sum_{k=j}^{i-1} a_{i,k} * b_{k,j} \quad j=1:i-1$$

Complexitatea acestor algoritmi este $O(n^3)$.

Coduri Matlab:

```
%Inversarea unei matrice superior triunghiulare
function B = InvTriSup(A)
[n, n] = size(A);
for j = n : -1 : 1
    B(j, j) = 1 / A(j, j);
    for i = j - 1 : -1 : 1
        B(i, j) = -A(i, i+1:j) * B(i+1:j, j) / A(i, i);
    end
end
```

```

%Inversarea unei matrice inferior triunghiulare
function B = InvTriInf(A)
[n, n] = size(A);
for i = 1 : n
    B(i, i) = 1 / A(i, i);
    for j = 1:i-1
        B(i,j)=-A(i,j:i-1)*B(j:i-1,j)/A(i,i);
    end
end
end

```

2. Metode de inversare bazate pe factorizari

Utilizarea factorizarii LU reduce inversarea unei matrici la inversarea a doua matrici triunghiulare

$$A^{-1} = (LU)^{-1} = U^{-1} L^{-1}$$

Inversarea, pe baza de eliminare gaussiana cu pivotare partiala, porneste de la relatia

$$T_{n-1}P_{n-1} \dots T_1P_1A=U$$

care inmultita la stanga cu U^{-1} si la dreapta cu A^{-1} se scrie

$$A^{-1} = U^{-1} T_{n-1}P_{n-1} \dots T_1P_1$$

care conduce la algoritmul:

1) se face eliminare gaussiana cu pivotare partiala

2) $X=U^{-1}$

3) pentru $p=n-1:-1:1$

$$X=X*T_p$$

$$X=X*P_p$$

Factorizarea QR reduce de asemeni inversarea unei matrici la inversarea unei matrici triunghiulare

$$A^{-1}=(QR)^{-1} = R^{-1} Q^{-1} = R^{-1} Q^T$$

Factorizare Crout

Metoda eliminării gaussiene fără permutări de linii sau coloane este echivalentă cu descompunerea matricei A într-un produs dintre o matrice superior triunghiulară L și una inferior triunghiulară U

$$A = L U$$

$$A = \begin{matrix} & \begin{matrix} l_{1,1} & 0 & \dots & 0 \end{matrix} & \begin{matrix} u_{1,1} & u_{1,2} & \dots & u_{1,n} \end{matrix} \\ \begin{matrix} l_{2,1} & l_{2,2} & \dots & 0 \end{matrix} & * & \begin{matrix} 0 & u_{2,2} & \dots & u_{2,n} \end{matrix} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \begin{matrix} l_{n,1} & l_{n,2} & \dots & l_{n,n} \end{matrix} & & \begin{matrix} 0 & 0 & \dots & u_{n,n} \end{matrix} \end{matrix}$$

Ținând cont de asociativitatea matricilor,

$$Ax = (LU)x = L(Ux) = b$$

se rezolvă pe rând sistemele

$$Lz = b \text{ și } Ux = z$$

Sistemele se rezolvă ușor datorită faptului că L și U sunt matrice triunghiulare. Avem un sistem de n^2 ecuații cu $n^2 + n$ necunoscute, deci trebuie precizate n dintre necunoscute. Cele mai utilizate alegeri sunt date de factorizarea Doolittle când se alege $l_{i,i} = 1$, factorizarea Crout când se alege $u_{i,i} = 1$ și factorizarea Cholesky când $U = L^T$, deci $l_{i,i} = u_{i,i}$. Factorizarea Cholesky este posibilă numai dacă matricea A este simetrică și pozitiv definită.

În factorizarea Crout dezvoltăm

$$A_{i,j} = \sum_{m=1}^{\min(i,j)} L_{i,m} U_{m,j}$$

Procesul de calcule al elementelor matricelor L și U poate fi inițializat calculând prima coloană din L și prima linie din U :

$$A_{i,1} = L_{i,1} * U_{1,1} \Rightarrow A_{i,1} = L_{i,1} \quad i=1:n$$

$$A_{1,j}=L_{1,1}*U_{1,j} \Rightarrow U_{1,j} = A_{1,j} / L_{1,1} \quad j=2:n$$

Folosind inductia, procesul de calcul poate fi continuat. Presupunand ca au fost calculate primele p-1 coloane din L si primele p-1 linii din U, putem calcula

-coloana p din L

$$A_{i,p} = \sum_{m=1}^{p-1} L_{i,m} * U_{m,p} + L_{i,p} * U_{p,p} \Rightarrow L_{i,p} = A_{i,p} - \sum_{m=1}^{p-1} L_{i,m} * U_{m,p} \quad i=p:n$$

-linia p din U

$$A_{p,j} = \sum_{m=1}^{p-1} L_{p,m} * U_{m,j} + L_{p,p} * U_{p,j} \Rightarrow U_{p,j} = \frac{A_{p,j} - \sum_{m=1}^{p-1} L_{p,m} * U_{m,j}}{L_{p,p}} \quad j=p+1:n$$

Numarul de operatii in factorizarea Crout este $O(n^3)$

Algoritmul Crout realizeaza factorizarea LU cu relatiile de mai sus folosind acelasi spatiu de memorie (matricile L si U vor fi suprascrie matricea A iar elementele diagonale ale lui U vor fi memorate)

Algorithm Crout

A=Crout(n,a)

```
{
    pentru p=1:n
    {
        pentru i=p:n
        {
            s=A(i,1:p-1)*A(1:p-1,p)
            A(i,p)= A(i,p)-s
        }
        j=p+1:n
        {
            s=A(p,1:p-1)*A(1:p-1,j)
            A(p,j)= (A(p,j)-s)/A(p,p)
        }
    }
}
```

Metoda suprarelaxarii succesive

Metoda suprarelaxarii succesive este o varianta modificata a metodei iterative Seidel - Gauss, care urmareste accelerarea suplimentara a convergentei procesului iterativ. In acest scop, se utilizeaza o desfacere $A = N - P$ care depinde de un parametru real μ , denumit parametru de accelerare (relaxare).

$$A = N(\mu) - P(\mu)$$

Metricele de desfacere $N(\mu)$ si $P(\mu)$ se pot alege arbitrar ca functii ale parametrului μ . Sa consideram insa cazul partiluar

$$N(\mu) = L + (1/\mu)D$$

$$P(\mu) = -R - (1 - \mu)D$$

Alegerea judicioasa a parametrului μ poate determina cresterea ratei de convergenta. De fapt, pentru orice matrice A si pentru orice desfacere a acesteia de forma $A = N(\mu) - P(\mu)$ se poate stabili o valoare optima a parametrului de accelerare μ , ca solutie a unei probleme de minimax (minimizarea unei valori maxime). Un alt avantaj al metodei suprarelaxarii succesive consta in posibilitatea transformarii unor procese divergente in procese convergente.

Introducere matricelor $N(\mu)$ si $P(\mu)$ astfel definite in realatia de recurenta si particularizarea acesteia din urma pentru una dintre necunscutele x_i conduc la formula re iterare a metodei suprarelaxarii succesive:

$$x_i^{k+1} = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \cdot x_j^{k+1} - \sum_{j=i+1}^n \frac{a_{ij}}{a_{ii}} \cdot x_j^k + \left(1 - \frac{1}{\mu}\right) \cdot (x_i^{k+1} - x_i^k) \quad i = 1, \dots, n$$

Comparand formula de iterare a metodei Seidel-Gauss

$$x_i^{k+1} = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \cdot x_j^{k+1} - \sum_{j=i+1}^n \frac{a_{ij}}{a_{ii}} \cdot x_j^k \quad i = 1, \dots, n$$

cu relatia de mai sus se constata ca metoda suprarelaxarii succesive determina noua aproximatie prin aplicarea unei corectii (ultimul termen din relatie) aproximatiei calculate dupa metoda Seidel - Gauss. Aceasta corectie depinde de abaterile aproximatiilor in doua iteratii succesive $x_i^{k+1} - x_i^k$ si de parametrul de accelerare μ .

Pe de alta parte, se constata ca metoda suprarelaxarii succesive nu poate folosi direct formula de iterare, deoarece aceasta determina aproximatia x_i^{k+1} in functie de ea insasi. Din acest motiv, algoritmul de calcul determina mai intai o solutie provizorie x_i^{k+1} cu metoda Seidel-Gauss obisnuita, careia ii aplica apoi o corectie in functie de abaterea $x_i^{k+1} - x_i^k$:

$$x_i^{k+1} \leftarrow x_i^{k+1} + \left(1 - \frac{1}{\mu}\right) \cdot (x_i^{k+1} - x_i^k) \quad i = 1, \dots, n$$

Pentru aceasta formula se pot stabili doua expresii echivalente:

$$\begin{aligned} x_i^{k+1} &\leftarrow x_i^{k+1} + \omega_1 \cdot (x_i^{k+1} - x_i^k) \\ x_i^{k+1} &\leftarrow x_i^k + \omega_2 \cdot (x_i^{k+1} - x_i^k) \end{aligned}$$

care introduc doi noi parametri de accelerare:

$$\omega_1 = 1 - (1/\mu) \quad \omega_2 = 2 - (1/\mu)$$

Convergenta metodei Seidel - Gauss modificate, depinde in foarte mare masura de alegerea parametrilor de accelerare μ , ω_1 si ω_2 . Iata doua cazuri particulare:

- $\mu = 1$, $\omega_1 = 0$, $\omega_2 = 1$. Aproximatiei nou calculate nu i se aplica nici o corectie. Se obtine metoda Seidel - Gauss obisnuita;
- $\mu = 1/2$, $\omega_1 = -1$, $\omega_2 = 0$. Aproximatia nou calculata coincide cu aproximatia din iteratia anterioara. Procesul este stationar, toate aproximatiile ramanand identice cu aproximatia initiala.

In practica, factorul de relaxare ω_2 se alege de regula in intervalul $(0, 2)$. Daca $\omega_2 > 2$ procesul iterativ diverge in cele mai multe cazuri. In general, nu exista un criteriu unic de determinare a valorii optime a parametrului de accelerare ω_2 , aceasta valoare fiind influentata de structura particulara a fiecarui sistem de ecuatii liniare analizat. Din acest motiv, se

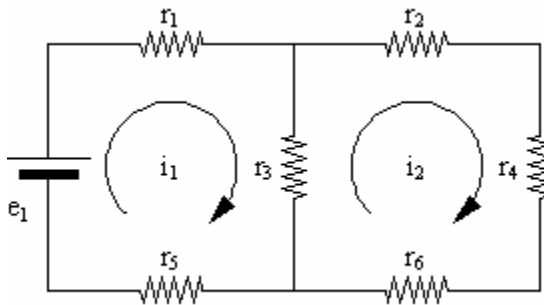
recomanda utilizarea cu prudenta a metodei suprarelaxarii succesive, iar atunci cand aceasta se aplica se recomanda folosirea unor parametri de accelerare cu valori in intervalul 1.4 - 1.5.

In general, valori ale parametrului de accelerare $1 < \omega_2 < 2$ asigura accelerarea convergentei unor procese iterative deja convergente, in timp ce valori ale aceluiasi parametru $0 < \omega_2 < 1$ permit deseori transformarea unor procese divergente in procese convergente.

Legile lui Kirchhoff

O aplicatie a sistemelor de ecuatii liniare este circuitul electric. Legea lui Kirchhoff in tensiune afirma ca suma caderilor de tensiune intr-o bucla trebuie sa fie zero. O bucla este un drum intr-un circuit unde punctul de sosire este cel de plecare.

Exemplu 1



Avem date de intrare $r1=10$ $r2=10$ $r3=10$ $r4=20$ $r5=10$ $r6=30$ si $e1=20$

Din Kirchhoff rezulta:

$$\begin{array}{rcl} (r1+r3+r5)i1 & - & r3i2 & = e1 \\ -r3 & + & (r2+r3+r4+r6)i2 & = 0 \end{array}$$

Folosind Matlab cu functiile imprementate anterior, avem:

```
>> R=[30 -10;-10 70]
```

```
R =
```

```
    30   -10  
   -10    70
```

```
>> e= [20;0]
```

```
e =
```

```
    20  
     0
```

```
>> Ri=Gauss_Jordan(R)
```

```
Ri =
```

```
    0.0350    0.0050  
    0.0050    0.0150
```

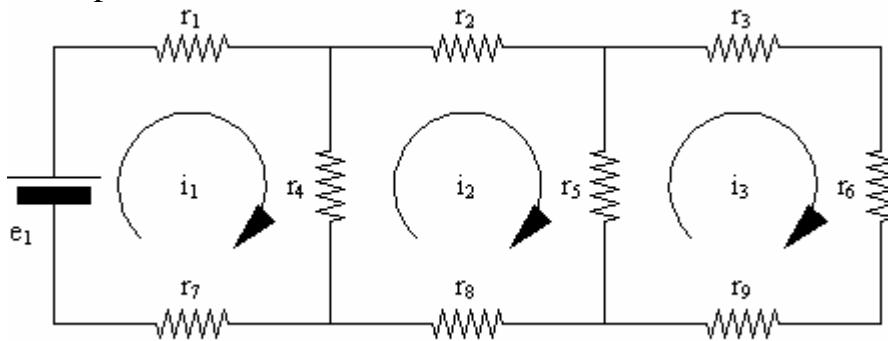
```
>> I=Ri*e
```

```
I =
```

```
    0.7000  
    0.1000
```

Rezulta ca $i_1 = 0.7$ si $i_2 = 0.1$.

Exemplu 2



$r_1=10, r_2=10, r_3=10, r_4=10, r_5=20, r_6=20, r_7=10, r_8=10, r_9=10$ $e_1=36$

Kirchoff:

$$\begin{array}{rcl} (r_1+r_4+r_7)i_1 & -r_4 i_2 & =e_1 \\ -r_4 i_1 & +(r_2+r_4+r_5+r_8)i_2 & =0 \\ & -r_5 i_3 & \\ & +(r_3+r_5+r_6+r_9)i_3 & =0 \end{array}$$

```
>> R=[30 -10 0;-10 50 -20;0 -20 60]
```

R =

```
30  -10   0
-10   50 -20
  0  -20  60
```

```
>> Ri=Gauss_Jordan(R)
```

Ri =

```
0.0333  0.0083  0.0028
  0  0.0250  0.0083
  0  0.0083  0.0194
```

```
>> E=[36;0;0]
```

E =

```
36
  0
```

0

$$\gg I = R_i * E$$

$$I =$$

1.2000

0

0

Modelul economic Leontief

Modelul matematic pentru economia unei țări sau regiuni este bazată pe sectoarele sale economice. Această problemă este adresată de către modelul Leontief. Presupunând că fiecare industrie din economie are două tipuri de cereri: una externă și una internă, modelul Leontief este un sistem liniar de ecuații.

Considerăm că economia constă din n industrii independente (sectoare) $s_1, s_2, \dots, s_n$. Fiecare sector consumă produse fabricate de celelalte industrii, inclusiv de ea însăși. O economie se numește închisă dacă își satisface propriile nevoi (bunurile nu intra sau ies din sistem)

Fie

p_i producția industriei s_i

$a_{i,j}$ este numărul de unități produse de industria s_i necesare

industriei s_j

$a_{i,j}p_j$ unități produse de industria s_i și consumate de s_j

$a_{i,1}p_1 + a_{i,2}p_2 + a_{i,3}p_3 + \dots + a_{i,n}p_n$ nr total de unități produse de

industria s_i

Dacă economia este închisă atunci

$$a_{i,1}p_1 + a_{i,2}p_2 + a_{i,3}p_3 + \dots + a_{i,n}p_n = p_i \quad \text{pentru } i \text{ de la } 1 \text{ la } n$$

Rezulta un sistem liniar

$$\begin{aligned}a_{1,1}p_1 + a_{1,2}p_1 + a_{1,3}p_1 + \dots + a_{1,n}p_1 &= p_1 \\a_{2,1}p_2 + a_{2,2}p_2 + a_{2,3}p_2 + \dots + a_{2,n}p_2 &= p_2 \\&\dots\dots\dots \\a_{n,1}p_n + a_{n,2}p_n + a_{n,3}p_n + \dots + a_{n,n}p_n &= p_n\end{aligned}$$

Acestui sistem ii se asociaza matricile A si P astfel:

$$AP=P$$

In cazul unei economii deschise, mai avem si cererile b_i din exterior. Astfel avem :

$$\begin{aligned}a_{1,1}p_1 + a_{1,2}p_1 + a_{1,3}p_1 + \dots + a_{1,n}p_1 + b_1 &= p_1 \\a_{2,1}p_2 + a_{2,2}p_2 + a_{2,3}p_2 + \dots + a_{2,n}p_2 + b_2 &= p_2 \\&\dots\dots\dots \\a_{n,1}p_n + a_{n,2}p_n + a_{n,3}p_n + \dots + a_{n,n}p_n + b_n &= p_n\end{aligned}$$

In forma matriciala:

$$P=AP + B$$

Pentru a rezolva sistemul, scriem sub forma $P= (I-A)^{-1}B$

Metoda Simplex de programarea liniara

Programarea liniara este o parte a algebrei liniare in care scopul este de a minimiza sau maximiza o functie $f(X)$ cu n variabile $X=(x_1,x_2\dots x_n)$ pe o regiune R ale carei granite este definite de ecuatii si inecuatii liniare.

$$f(X)= f(x_1,x_2\dots x_n) = \sum_{j=1}^n c_j x_j$$

Bibliografie

- METODE NUMERICE (Editura Tehnica Autori C.Berbente, S Mitran, S. Zancu)
- METODE NUMERICE (Editura Albastra Autori Valeriu Iorga, Boris Jora)
- <http://curs.cs.pub.ro/> Curs de Metode Numerice
- <http://math.fullerton.edu/>
- <http://www.ee.tuiasi.ro/> Metode Numerice- Aplicatii