Lucrarea 3

REZOLVAREA NUMERICĂ A SISTEMELOR DE ECUAȚII

1. SCOPUL LUCRĂRII

*Prezentarea unor metode de rezolvare a sistemelor de ecuații liniare şi neliniare şi a algoritmilor acestora într-o variantă de pseudocod asemănătoare limbajului **C**, implementarea într-un limbaj de nivel înalt (în particular, C) şi folosirea lor în rezolvarea unor probleme de electronică.*

2. PREZENTARE TEORETICĂ

Fie funcția $f: X -> Y$, $X \subset \mathbf{R}^n$, $Y \subset \mathbf{R}^n$ şi $f(x) = 0$ un sistem de ecuații. După gradul necunoscutelor din ecuații, sistemele sunt liniare, dacă acestea sunt de gradul întâi şi neliniare dacă există ecuații ce conțin necunoscute la puteri mai mari ca unu.

2.1. REZOLVAREA NUMERICĂ A SISTEMELOR LINIARE

Un sistem liniar de *n* ecuații cu *n* necunoscute se prezintă astfel:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2 \\ - - - - - - - - - - - - - - - - \\ a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n \end{cases} \tag{3.1}$$

sau matriceal

$$AX=B \tag{3.2}$$

unde:

$$A=\begin{pmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ - & - & - & - \\ a_{n1} & a_{n2} & ... & a_{nn} \end{pmatrix} \tag{3.3}$$

$$X=\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \quad (3.4) \qquad \text{şi} \quad B=\begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix} \quad (3.5)$$

Metodele de rezolvare a sistemelor liniare le putem împărți în două categorii: ***metode directe*** care presupun un număr finit de operații şi ***metode indirecte*** care implică un număr infinit de operații.

2.1.1. METODE DIRECTE

Aceste metode sunt caracterizate prin aceea că pot oferi soluția exactă a sistemului de rezolvat, şi dacă sunt îndeplinite condițiile de existență. Soluția oferită nu cuprinde decât erorile proprii operațiilor aritmetice, demonstrate cu ajutorul grafurilor de procedură în prima lucrare de laborator, împreună cu erorile de rotunjire şi trunchiere interne, d.p.d.v. al reprezentării interne în virgulă mobilă.

Dintre metodele directe prezentăm metoda pentru sisteme *superior şi inferior triunghiulare*, metoda eliminării a lui *Gauss*, metoda lui *Crout*, metoda *QR* şi metoda de rezolvare a sistemelor *tridiagonale*.

2.1.1.1 SISTEME SUPERIOR TRIUNGHIULARE

Sistemele de tipul superior triunghiular sunt de forma:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1 \\ a_{22}x_2 + a_{21}x_2 + \ldots + a_{2n}x_n = b_2 \\ a_{33}x_3 + \ldots + a_{3n}x_n = b_3 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{nn}x_n = b_n \end{cases} \quad (3.6)$$

Calculul soluțiilor sistemului se face printr-o *retro-substituție* astfel:

$$x_n = \frac{b_n}{a_{nn}} \qquad \text{şi} \qquad x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}} \quad i=n\text{-}1,\ n\text{-}2,\ \ldots,1 \quad (3.7)$$

2.1.1.1.1. Algoritmul 3.1. Sistem superior triunghiular

```
real Superior_Triunghiular
(
   întreg n,              // ordinul sistemului
   real A[N][N],    // matricea necunoscutelor sistemului
   real B[N],             // vectorul termenilor liberi
   real X[N]              // vectorul soluţiilor
)
{
// declararea şi definirea variabilelor locale
   real d;      // reţine valoarea determinantului
   întreg i, j; // indici pentru ciclurile 'for()' şi pentru
                // indexarea vectorilor.

// corpul de instrucţiuni al funcţiei
d=1;
pentru i=1,n   d = d*A[i][i];
dacă(d == 0) return 0;
pentru i= n,1
   {
      x[i] = b[i];
      pentru j= n,i+1   x[i] = x[i] - A[i][j]*x[j];
      x[i] = x[i]/A[i][i];
   }

returnează d;
// se poate returna valoarea determinantului, deoarece
// forma particulară a sistemului permite calcularea cu
// uşurinţă a acestuia.
}
```

2.1.1.2. METODA LUI GAUSS DE ELIMINARE

Această metodă constă în realizarea unui sistem triunghiular prin eliminarea termenilor de sub diagonala principală, sistemul (3.1). Pentru aceasta se analizează toţi termenii a_{k1} , k=1,...,n , şi ecuaţia care are valoarea maximă a acestui coeficient este adusă pe primul loc. Dacă toţi a_{k1}=0 pentru k=1,...,n sistemul este incompatibil. Prima ecuaţie a sistemului după reordonare se înmulţeşte pe rând cu factorul :

$$m_{k1} = \frac{a_{k1}}{a_{11}}, \quad k=2, \ldots, n \tag{3.8}$$

şi se scade din ecuaţia de pe poziţia k, obţinând pe coloana 1 toţi termenii zero în afară de a_{11}. Procedeul continuă cu linia şi coloana a doua, prima linie şi prima coloană rămânând neschimbate. Ultimul index de linie pentru care se aplică acest procedeu este *n-1*.

2.1.1.2.1. Algoritm 3.2. Metoda lui Gauss de eliminare

```
real Gauss
(
    întreg n,              // ordinul sistemului
    real A[ ][N],          // matricea sistemului
    real B[ ],             // vectorul termenilor liberi
    real *X                // vectorul soluţiilor.
)
{ // declaraţiile şi definiţiile variabilelor locale
          întreg i;        // indicele liniei
          întreg k;        // indice suplimentar al liniei
          întreg j;        // indice coloană
          real m;          // multiplicator
          real d;          // determinantul sistemului
          real max;        // elementul maxim de pe coloana coeficienţilor
          real lp;         // indicele liniei cu element maxim
// corpul de instrucţiuni al funcţiei
    d=1;    // iniţializarea  cu 1 a variabilei în care ţin valoarea
            // determinantului.
    pentru i = 1,n-1 {  // for 1 .
            max = fabs(A[i][i]);
            lp = i;         // linia ce conţine coeficientul maxim.
            pentru k = i+1,n { // for 2 .
                dacă (fabs(A[k][i])>max ) {       max = fabs(A[k][i]);
                                                  lp = k;
                                                  }
            } // for 2
            dacă (max == 0)  returnează 0;
            dacă (i != lp){ // în caz ca linia ce conţine coeficientul maxim
                            // este diferită de prima linie, se face
                            // interschimbarea.
                pentru j = i,n  Schimb(A[i][j], A[lp][j]);
                            // interschimbarea coeficienţilor necunoscutelor
                            // de pe linia i cu cei de pe linia lp.
                Schimb(B[i], B[lp]);      // schimbarea între ei a
                                          // termenilor liberi.
                d = -d;
                // odată cu schimbarea a două linii într-un determinant,
                // valoarea sa îşi schimbă semnul.
            }
        pentru k =  i+1,n  {
            m = A[k][i]/A[i][i];
            pentru j = i , n  A[k][j] = A[k][j] - m*A[i][j];
            B[k] = B[k] - m*B[i];
            }
    } // for 1.
// Rezolvă sistem superior triunghiular.
    dacă (A[n][n] = = 0)  returnează 0;
    pentru i =1,n   d = d*A[i][i];        // Calculează determinantul
    pentru i = n,1  {
            x[i] = B[i];
            pentru j = n,i+1  x[i] = x[i] - A[i][j]*x[j];
            x[i] = x[i] / A[i][i];
            }
    returnează d;  // odată calculată, valoarea determinantului poate
                   // fi returnată de către funcţie.
}
```

Obs.: Operația de interschimbare a coeficienților necunoscutelor de pe două linii distincte se poate face fie prin intermediul unei *funcții macro* (definită cu ajutorul directivei *#define* - vedeți și *Anexa B* - "*Noțiuni de C necesare desfășurării lucrărilor de laborator*") fie printr-o funcție propriu-zisă, având argumente referință.

2.1.1.3. METODA LUI CROUT

Rezolvarea sistemului $AX=B$ constă în descompunerea matricei A sub forma:

$$A=LU$$

L fiind o matrice *inferior triunghiulară*, iar U o matrice *superior triunghiulară* cu elementele diagonalei principale egale cu unitatea. Sistemul se scrie:

$$LUX=B$$

sau, dacă notăm $UX=Y$, atunci:

$$LY=B$$

Aşadar, rezolvarea sistemului dat revine la rezolvarea sistemului inferior triunghiular $LY=B$ (determinarea elementelor vectorului coloană Y) apoi la rezolvarea sistemului superior triunghiular $UX=Y$, cu determinarea elementelor vectorului necunoscutelor X (de asemenea vector coloană).

2.1.1.3.1. Algoritm 3.3. Metoda lui Crout

```
întreg Crout (
    întreg n,      // ordinul sistemului
    real A[ ][N],  // matricea coeficienţilor necunoscutelor sistemului
    real B[ ],     // vectorul termenilor liberi
    real X[ ],     // vectorul soluţiilor sistemului
    real U[ ],     // matricea superior triunghiulară cu diagonala
                   // principală unitară
    real L[ ],     // matricea inferior triunghiulară
    )
    { // declaraţiile şi definiţiile variabilelor locale
         întreg i, j;            // indici de linie, respectiv de coloană
         întreg k, s;            // indici suplimentari
         real D[N];              // vector suplimentar
         real sum, prod;         // variabile suplimentare

    // corpul de instrucţiuni al funcţiei
         pentru  i = 1,n  U[i][i] = 1;

         pentru k =  1,n { // for k .

           pentru i =  k,n {  // for i
                 sum = 0;
                 pentru s= 1,k-1  sum = sum + L[i][s] * U[s][k];
                 L[i][k] = A[i][k] - sum;
                 } // for i .

           pentru j =  k+1,n { // for j .
```

```
                sum=0;
                pentru s = 1,k-1  sum = sum + L[k][s] * U[s][j];
                dacă (L[k][k]= =0) returnează False;
                U[k][j] = (A[k][j] - sum)/L[k][k];
                } // for j .
          }  // for k .

    // Rezolvă LY=B
        prod = 1;
        pentru  i= 1,n   prod = prod * L[i][i];
         dacă (prod ==0) returnează False;

        pentru i= 1,n {
            D[i]=B[i];
            pentru  j= 1,i-1   D[i] = D[i] - L[i][j] * D[j];
            D[I] = D[i]/L[i][i];
            }

    // Rezolvă UX=Y
            pentru i= n,1 {
                X[i] = D[i];
                pentru j= n,i+1  X[i] = X[i] - U[i][j] * X[j];
                X[i] = X[i]/U[i][i];
                }
    returnează True;
}
```

2.1.1.4. METODA FACTORIZĂRII *QR*

Această metodă face posibilă rezolvarea sistemelor $AX=B$ pentru cazul când $A \in \mathbf{R}^{mxn}$ şi $B \in \mathbf{R}^{m}$, m>n caz în care nu au soluție în general. Soluționarea se face cu ajutorul metodei lui *Householder*.

2.1.1.4.1. ALGORITMUL 3.4. Metoda factorizării QR

```
void QR
    (
      întreg m,  // număr de ecuaţii;
      întreg n,  // număr de necunoscute (m>n) ;
      real A[ ][N],     // matricea sistemului;
      real B[ ], // vectorul termenilor liberi;
      real X[ ], // vectorul soluţiilor;
      real Q[ ], // matricea ortogonală de ordinul m*m;
      real R[ ], //  matricea  superior  triunghiulară  de  ordinul
          m*n;
    )
    {// declaraţiile şi definiţiile variabilelor locale
      întreg k, i, j;   // indici;
      real s, p, t;     // variabile suplimentare;
      real v [N];// vector reflector;

      // corpul de instrucţiuni al funcţiei
```

```
pentru i= 1,m {
 pentru j= 1,m  Q[i][j] = 0;
Q[i][i] = 1;
}
pentru k= 1,m {
 s = 0.0;
      pentru i=k,m     s = s + A[i][k] * A[i][k];
            s = sqrt(s);
            dacă (A[k][k]<0) s = -s;
   pentru  i= 1,k-1    v[i] = 0;
         v[k] = A[k]k] + s;
         A[k][k] = -s;
   pentru i= k+1,m {
      v[i] = A[i][k];
      A[i][k]= 0.0;
      }

   p = s*v[k];
   pentru j= k+1,n  {
      t=0;
                        pentru  i=  k,m     t  =  t  +  v[i]  *
A[i][j];
      t = t/p;
      pentru i= k,m  A[i][j] = A[i][j] - t*v[i];
                        }
      t=0;
             pentru i= k,m  t = t + v[i]*tl[i];
             t = t/p;
             pentru i=k,m     t1[i] = tl[i] - t * v[i];
             pentru j= 1,m {
      t=0;
                         pentru    i=k,m        t    =    t    +
v[i]*Q[i][j];
                         t = t/p;
                         pentru i= k,m     Q[i][j] = Q[i][j]
- t*v[i];
                        }
                  }
                  pentru i= n,1 {
      s=0;
                        pentru  j=i+1,n            s  =  s  +
A[i][j]*x[j];
                        X[i] = (B[i]-s)/A[i][i];
                        }
                  pentru i= 1,m
```

```
                    pentru j= 1,n    R[i][j] = A[i][j];
                pentru i= 1,m
                    pentru j= 1,m { t = Q[i][j]; Q[i][j] =
Q[j][i]; Q[j][i] = t; }
                }
```

2.1.1.5. METODA DE REZOLVARE A SISTEMELOR TRIDIAGONALE

Un tip special de sistem liniare îl reprezintă sistemele tridiagonale:

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & \dots & 0 & 0 & 0 \\ - & - & - & - & - & - & - & - \\ 0 & 0 & 0 & 0 & \dots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ - \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ - \\ d_{n-1} \\ d_n \end{pmatrix} \tag{3.9}$$

Metoda simplă de rezolvare a acestor probleme constă în descompunerea matricei tridiagonale în două matrice *L, U* .

$$LU = A \tag{3.10}$$

Matricele L şi U sunt *matrice triunghiulare*, adică au fie zona de sub diagonala principală, fie pe cea de deasupra, diferite de zero.

Forma concretă a matricelor este (relația 3.11):

$$L=\begin{pmatrix} l_1 & 0 & 0 & ... & 0 & 0 & 0 \\ p_2 & l_2 & 0 & ... & 0 & 0 & 0 \\ 0 & p_3 & l_3 & ... & 0 & 0 & 0 \\ - & - & - & - & - & - & - \\ 0 & 0 & 0 & ... & p_{n-1} & l_{n-1} & 0 \\ 0 & 0 & 0 & ... & 0 & p_n & l_n \end{pmatrix}, U=\begin{pmatrix} 1 & u_1 & 0 & 0 & ... & 0 & 0 \\ 0 & 1 & u_2 & 0 & ... & 0 & 0 \\ 0 & 0 & 1 & u_3 & ... & 0 & 0 \\ - & - & - & - & - & - & - \\ 0 & 0 & 0 & 0 & ... & 1 & u_{n-1} \\ 0 & 0 & 0 & 0 & ... & 0 & 1 \end{pmatrix} \tag{3.11}$$

Rezolvarea sistemului se face plecând de la înmulțirea matricei *L* cu matricea *U* şi apoi prin identificarea cu matricea *A* a matricei rezultate, element cu element. Se obțin astfel relațiile de recurență pentru calculul elementelor matricelor *L* şi *U*:

$$\begin{aligned} & p_i = a_i && i=2,3,...,n \\ & u_i = \frac{c_i}{l_i} && i=1,2,3,...,n \\ & l_1 = b_1 \\ & l_i = b_i - a_i u_{i-1} && i=2,3,...,n \end{aligned} \tag{3.12}$$

Sistemul poate fi scris sub forma

$$LUX=D \tag{3.13}$$

sau sub forma a două sisteme

$$\begin{cases} L \cdot Y = D \\ U \cdot X = Y \end{cases} \tag{3.14}$$

Prin identificare se deduc următoarele relaţii recursive de calcul:

$$\begin{cases} y_1 = \dfrac{d_1}{l_1} \\ y_i = \dfrac{\left(d_i - a_i y_{i-1}\right)}{l_i}, i = 2,3,...n \end{cases} \quad ; \; l_i \neq 0 \tag{3.15}$$

şi

$$\begin{cases} x_n = y_n \\ x_i = y_i - u_i x_{i+1}, \qquad i=n\text{-}1,...,1 \end{cases} \tag{3.16}$$

care reprezintă soluţiile sistemului tridiagonal.

2.1.1.5.1. Algoritm 3.5. Sisteme Tridiagonale

```
întreg Tridiagonal
(
   întreg n, // ordinul sistemului;
   real a[ ],   // vectorul elementelor sub-diagonle din
                // matricea sistemului;
   real b[ ],// vectorul elementelor diagonale din matricea
             // sistemului;
   real c[ ],      // vectorul elementelor supra-diagonale din
                   // matricea sistemului;
   real d[ ],      // vectorul termenilor liberi;
   real x[ ]       // vectorul soluţiilor
)
{
// declaraţiile şi definiţiile variabilelor locale
real t[N+1];       // vectorul elementelor diagonale din L;
real p[N+1]; // vectorul elementelor sub-diagonale din L;
real s[N];   // vectorul elementelor supra-diagonale din
             // matricea U
real y[N+1]; // vectorul UX;
întreg i, j; // indici pentru ciclurile 'for()' şi indexarea
             // vectorilor

// corpul de instrucţiuni al funcţiei
pentru  i= 2,n   p[i] = a[i];
t[1] = b[1];
dacă (t[1]= =0) returnează False;
s[1] = c[1]/t[1];
y[1] = d[1]/t[1];

pentru i= 2,n-1 {
   t[i] = b[i] - a[i]*s[i-1];
   dacă (t[i] = = 0) returnează False;
   s[i] = c[i] / t[i];
```

```
    y[i] = (d[i] - a[i]*y[i-1]) / t[i];
    }

t[n] = b[n] - a[n]*s[n-1];
dacă (t[n]= =0)  returnează False;

x[n] = (d[n] - a[n]*y[n-1])/t[n];
pentru i=n-1,1 X[i] = Y[i] - s[i]*X[i+1];
returnează True;
}
```

2.1.2. METODE INDIRECTE

Dintre metodele indirecte sau metodele iterative care rezolvă un sistem de ecuații liniare prezentăm *metoda lui Jacobi*.

2.1.2.1. METODA LUI JACOBI

Fie sistemul liniar (3.1). Se explicitează fiecare necunoscută din sistem de pe diagonala principală funcție de celelalte necunoscute ale sistemului .

$$\begin{cases} x_1 = \frac{b_1}{a_{11}} - 0 - \frac{a_{12}}{a_{11}}x_2 - \frac{a_{13}}{a_{11}}x_3 - \ldots - \frac{a_{1n}}{a_{11}}x_n \\ x_2 = \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}}x_1 - 0 - \frac{a_{23}}{a_{22}}x_3 \ldots - \frac{a_{2n}}{a_{22}}x_n \\ \text{----------} \\ x_n = \frac{b_n}{a_{nn}} - \frac{a_{n2}}{a_{nn}}x_1 - \ldots - \frac{a_{nn\text{-}1}}{a_{nn}}x_{n\text{-}1} + 0 \end{cases} \tag{3.17}$$

Se alege o soluție a sistemului oarecare $x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)},$ ce va reprezenta soluția de start, după care se continuă iterațiile, până când două soluții succesive diferă cu mai puțin de o eroare impusă (precizia impusă).

2.1.2.1.1. Algoritm 3.6. Metoda Jacobi

```
#define N 16  // presupunem că gradul sistemului nu depăşeşte 15
întreg Jacobi
(
   întreg n,     // ordinul sistemului;
   real A[ ][N],       // matricea sistemului;
   real B[ ],    // vectorul termenilor liberi;
   real X[ ]     //  vectorul  de  start  la  intrare,  soluția  la
ieşire
)
{
// declarațiile şi definițiile variabilelor locale
```

```
întreg i, j;           // indici;
real sum;              // variabilă auxiliară;
real Xn[N], Xn1[N];    // soluţiile la pasul curent şi precedent
întreg sem;            // variabilă logică auxiliară;

// Verifică dacă matricea e dominant-diagonală
   pentru  i= 1,n {
       sum = 0;
       pentru j= 1,n    sum = sum + fabs(A[i][j]);
       dacă (sum > 2*fabs(A[i][i])) returnează False;
       }

// Iteraţiile efective ale metodei
   pentru i= 1,n   Xn[i] = X[i];
   // se porneşte în calcule cu valorile conţinute de vectorul de
   // start (rolul vectorului de start este ţinut de către
   // vectorul X)
   execută
   {
       sem = True;
       pentru i= 1,n    Xn_1[i] = Xn[i];
       pentru i= 1,n
       {
          Xn[i] = B[i];
          pentru j= 1,n
                 dacă (j != i)  Xn[i] = Xn[i] - A[i][j] * Xn_1[j];
          Xn[i] = Xn[i]/A[i][i];
       }
       pentru i= 1,n
          dacă (fabs(Xn[i]-Xn_1[i]) > eps )  sem = False;
   } cât timp (sem = = False);
   pentru i= 1,n    X[i] = Xn[i];
   returnează True;
}
```

2.1.2.2. METODA GAUSS-SEIDEL

Are acelaşi principiu cu cel al metodei Jacobi, însă prezintă o diferenţiere majoră: pentru calculul necunoscutelor la pasul curent, se folosesc, acolo unde este posibil, şi valorile necunoscutelor calculate chiar la acest pas, şi nu în exclusivitate toate valorile de la pasul anterior.

Nu aceeaşi este situaţia în metoda Jacobi. După cum s-a prezentat, soluţia la pasul *n* se bazează în întregime pe valorile de la pasul *n-1*, şi nu foloseşte nici un calcul de la pasul *n*.

În consecinţă, metoda Gauss-Seidel prezintă un avantaj incontestabil: *viteza mărită de convergenţă*, şi aceasta pentru că, de exemplu, componenta a treia de la pasul *n* este aceeaşi cu cea de la pasul *n+1* din Jacobi. Deci valorile calculate în Gauss-Seidel sunt 'înaintea' celor din Jacobi, şi aceasta din cauza modului în care sunt calculate.

2.1.2.2.1. Algoritmul metodei Gauss-Seidel

```
defineşte tipul de dată LOGIC;      // tipul Boolean nu este
                                    // definit în ANSI C
defineşte constantele False şi True; // eventual cu o enumerare

intreg GaussSeidel
(
   intreg n,    // ordinul sistemului
   real a[][DIM],      // matricea sistemului
   real b[DIM], // vectorul termenilor liberi
   real x[DIM], // vectorul soluţie; la intrare este vector de
                // start
   real epsilon  // precizia impusa solutiilor
)
{
// corpul funcţiei
intreg i, j, pas=1;
LOGIC stop, marker[DIM];      // marker este vectorul in care
                              // marchez care componente au fost
                              // calculate la pasul curent.
real sum;              // pentru verif dominant-diagonalitatii
real xn[DIM], xn1[DIM];       // componentele curent si anterior
real eroare[DIM];      // vectorul erorilor componentelor
real valFolosita;      // pt depanare.


// Testul de dominant-diagonalitate al matricei de intrare;
// in caz de eroare se iese cu False
for i = 0,n
{
   suma = 0;    // inainte de fiecare suma, variabila suma devine
                // zero
         for j = 0,n
         {
            suma = suma + absolut(a[i][j]);
            daca (2*absolut(a[i][i]) < suma) returneaza False;
            // Structura matricei nu permite rezolvarea sist...
         }
}

// iteratiile asociate calculului solutiilor sistemului
execută
{
   for i = 0,n
   {
      xn1[i] = xn[i];        // comp curente devin anterioare;
      marker[i] = False;     // reset marker pt pasul curent
   }
  // se calc comp la pasul curent
   stop = True;
   for i = 0, n
       {
          xn[i] = b[i];// se începe calculul componentelor
                       // curente de la term liber.
          // Pentru calculul comp curente se ţine cont de
          // valorile logice memorate într-un vector de
          // markeri
```

```
            for j = 0,n
            {
               daca (marker[j] == 0) valFolosita = xn1[j];
               else valFolosita = xn[j];
               dacă (j != i) xn[i] = xn[i] - a[i][j]*valFolosita;
            }
            xn[i]  =  xn[i]/a[i][i];  //  s-a  încheiat  calc  comp
            curente
            marker[i] = True;  // si se marcheaza acest lucru
         }
      for i = 0,n
      {
         eroare[i] = fabs(xn[i] - xn1[i]);
         dacă (eroare[i] > epsilon) stop = False;
       }
} cât timp (stop este False);
// stabilirea vectorului soluție și marcarea succesului la ieșire
for i = 0,n
   x[i] = xn[i];       //  se  considera  solutie  ultimele  valori
calculate
returnează True;
}
```

2.2. REZOLVAREA NUMERICĂ A SISTEMELOR NELINIARE

Se numeşte *sistem neliniar* acel sistem pentru conține cel puțin o ecuație având gradul peste 1, din punctul de vedere al necunoscutelor.

Sistemul se prezintă astfel:

$$\begin{cases} f_1\left(x_1,x_2,...,x_n\right)=0 \\ f_2\left(x_1,x_2,...,x_n\right)=0 \\ \text{----------} \\ f_n\left(x_1,x_2,...,x_n\right)=0 \end{cases} \tag{3.18}$$

Dacă cel puțin una din funcțiile $f_1, f_2, f_3 \dots f_n$ sunt neliniare în variabilele $x_1,x_2,...,x_n$ sistemul de ecuații (3.18) se numeşte *sistem de ecuații neliniare*.

2.2.1. METODA LUI NEWTON DE REZOLVARE A ECUAȚIILOR NELINIARE

2.2.1.1. Algoritm 3.7. Metoda lui Newton

Prezentăm în cele ce urmează aplicarea *metodei Newton-Raphson* pentru rezolvarea unui sistem neliniar cu două ecuații şi două necunoscute. Generalizarea este imediată.

```
întreg Sis_Newton_Raphson
(
Adr(f1),      // adresa primei funcţii de două variabile care
              // formează sistemul (pointer la o funcţie);
Adr(f2),      // adresa celei de-a doua funcţii de două
              // variabile care formează sistemul (pointer la o
              // funcţie);
Adr(dxf1),
Adr(dxf2),    // Adresele derivatelor celor două funcţii în
              // raport cu prima variabilă (pointeri la
              // funcţie);
Adr(dyf1),
Adr(dyf2),    // Adresele derivatelor celor două funcţii în
              // raport cu a doua variabilă (pointeri la
              // funcţie);
real *solx,   // la intrare: soluţia de start pentru prima
              // variabilă la ieşire: soluţia finală a primei
              // variabile;
real *soly,   // la intrare: soluţia de start pentru a doua
              //  variabilă la ieşire: soluţia finală a celei
de-
              // a doua variabile;
real eps,           // eroarea de calcul;
întreg niter        // numărul maxim de iteraţii .
)
{
// declaraţiile şi definiţiile variabilelor locale
real xn, xn_1; // pasul curent şi precedent al variabilei x;
real yn, yn_1; // pasul curent şi precedent al variabilei y;
real dx, dy; // variaţiile celor două variabile;
// corpul de instrucţiuni al funcţiei
 xn = *solx;
 yn = *soly;
   execută
   {
      xn_1 = xn;  yn_1 = yn;
      dacă  ((*Adr(dxf1))(xn_1,yn_1) * (*Adr(dyf2))(xn_1,yn_1)
            (*Adr(dxf2))(xn_1,yn_1) * (*Adr(dyf1))(xn_1,yn_1)
             = = 0 )
      returnează False;

      dx = (-(*Adr(f1))(xn_1,yn_1) * (*Adr(dyf2))(xn_1,yn_1)
      + (*Adr(f2))(xn_1,yn_1) * (*Adr(dyf1))(xn_1,yn_1) )
      / ((*Adr(dxf1))(xn_1,yn_1) * (*Adr(dyf2))(xn_1,yn_1)
      - (*Adr(dxf2))(xn_1,yn_1) * (*Adr(dyf1))(xn_1,yn_1) );

      dy = (- (*(Adr(dxf1))(xn_1,yn_1)*(*Adr(f2))(xn_1,yn_1)
      + (*Adr(dxf2))(xn_1,yn_1) * (*Adr(f1))(xn_1,yn_1))
      / ((*Adr(dxf1))(xn_1,yn_1) * (*Adr(dyf2))(xn_1,yn_1)
      - (*Adr(dxf2))(xn_1,yn_1) * (*Adr(dyf1))(xn_1,yn_1));

      niter = niter-1;
      xn=xn_1+dx;
      yn=yn_1+dy;
} cât timp(  ((fabs(dx) >= eps) SAU (fabs(dy) >= eps))
                  ŞI (niter >0) );
dacă (niter = = 0) returnează False;
*solx = xn;
*soly = yn;
returnează True;
}
```

Obs.: Notațiile ‘Adr(f_i)’ şi ‘(*Adr(f))’ sunt simbolice. Ele reprezintă, respectiv, un pointer la o funcție şi apelul unei funcții prin pointer. Vedeți *Anexa B* - “*Noțiuni de C pentru laborator*” pentru lămuriri.

2.2.1.2. Implementarea algoritmului 3.7 (paragraful 2.2.1.1.) A metodei lui newton de rezolvare a sistemelor de ecuații neliniare

// Varianta unui sistem de trei ecuații cu trei necunoscute

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

#define h 0.0001       // pentru calculul derivatei numerice (relația
                       // conform definiției matematice).

// prototipurile funcțiilor folosite în program.
double ec1(double x,double y,double z);  // prima ecuație a sistemului.
double ec2(double x,double y,double z);  // a doua ecuație a sistemului.
double ec3(double x,double y,double z);  // a doua ecuație a sistemului.

double dec1x(double x,double y,double z);
// derivata prin două puncte, în raport cu x, a primei ecuații.

double dec1y(double x,double y,double z);
// derivata prin două puncte, în raport cu y, a primei ecuații.

double dec1z(double x, double y, double z);
// derivata prin două puncte, în raport cu z, a primei ecuații.

double dec2x(double x,double y,double z);
// derivata prin două puncte, în raport cu x, a celei de-a doua ecuații.

double dec2y(double x,double y,double z);
// derivata prin două puncte, în raport cu y, a celei de-a doua ecuații.

double dec2z(double x,double y,double z);
// derivata prin două puncte, în raport cu z, a celei de-a doua ecuații.

double dec3x(double x,double y,double z);
// derivata prin două puncte, în raport cu x, a celei de-a treia ecuații.

double dec3y(double x,double y,double z);
// derivata prin două puncte, în raport cu y, a celei de-a treia ecuații.

double dec3z(double x,double y,double z);
// derivata prin două puncte, în raport cu z, a celei de-a treia ecuații.

// calculul determinantului unui sistem liniar de trei ecuații cu trei
// necunoscute, conform regulii lui Sarrus.
    double dt3(    double a11, double a12, double a13,
                   double a21, double a22, double a23,
                   double a31 ,double a32, double a33);

    // funcția propriu-zisă de determinare a soluțiilor sistemului.
    int RezSistN(double *x0,double *y0,double *z0,int MAX,double err);


    // Programul principal.
    void main()
    {
    double x0,y0,z0,err;
    int imax,flag;
    clrscr();
```

```
// citirea componentelor vectorului soluţiilor de start: (x0, y0, z0).
printf("x0=");scanf("%lf",&x0);
printf("y0=");scanf("%lf",&y0);
printf("z0=");scanf("%lf",&z0);

// citirea erorii de calcul a soluţiilor.
printf("err=");scanf("%lf",&err);

// numărul maxim de iteraţii.
printf("Nr maxim de iteratii=");scanf("%d",&imax);

// apelul funcţiei de rezolvare a sistemului neliniar.
 flag = RezSistN(&x0,&y0,&z0,imax,err);
 if(flag)  printf("\nRezultat:\n x=%lf  y=%lf  z=%lf",x0,y0,z0);
 else   printf("\nEroare");

getch();
}


// Implementarea funcţiilor

double ec1(double x,double y,double z)  // prima ecuaţie a sistemului.
{
// return x+y+z-3;
//  return x*x + x*y + 2*z - 3.09;
  return x*y + y*y + 2*z - 2.48;
// return x*y + 2*y*y + 2*z - 2.84;
// return x*x + 2*x*y + 2*z - 3.28;
//  return x*x + 2*y*x + z - 1.61;
// return x*x + y*y + z*z - 1;
}

double ec2(double x,double y,double z)  // a doua ecuaţie a sistemului.
{
// return x*x-y+4*z*z-4;
//  return x*z + z*y + 4*z - 7.35;
  return x + z*y + 3*z - 3.8;
// return x + y*z + z - 1.8;
// return 2*x + y*z + 2*z - 3.;
//  return x*z + 3*y*z + z - 4.5;
//  return 2*x*y + y*z + z - 2.35;
// return 2*x*x + y*y - 4*z;
}

double ec3(double x,double y,double z)  // a treia ecuaţie a sistemului.
{// return x*x+y*y*y+2*z-4;
// return x + y*y + z*y - 1.94;
 return x + y + z*z - 1.8;
// return x + y + 2*z*z - 2.8;
// return 2*x + y + z*z - 2.;
// return x + y*y + 4*y*z - 4.16;
// return x + y*y + 2*y*z - 1.85;
// return x + y*y + y*z - 1.94;
// return 3*x*x - 4*y + z*z;
}

double dec1x(double x,double y,double z)
{
// derivata prin două puncte, în
// raport cu x, a primei ecuaţii.
{
return
(ec1(x-2*h,y,z)-8*ec1(x-h,y,z)+8*ec1(x+h,y,z)-ec1(x+2*h,y,z))/(12*h);
}
```

```
double dec1y(double x,double y,double z)
{
// derivata prin două puncte, în
// raport cu y, a primei ecuaţii.
return
(ec1(x,y-2*h,z)-8*ec1(x,y-h,z)+8*ec1(x,y+h,z)-ec1(x,y+2*h,z))/(12*h);
}

double dec1z(double x, double y, double z)
{
// derivata prin două puncte, în
// raport cu z, a primei ecuaţii.
return
(ec1(x,y,z-2*h)-8*ec1(x,y,z-h)+8*ec1(x,y,z+h)-ec1(x,y,z+2*h))/(12*h);
}

double dec2x(double x,double y,double z)
{
// derivata prin două puncte, în
// raport cu x, a celei de-a doua ecuaţii.
return
(ec2(x-2*h,y,z)-8*ec2(x-h,y,z)+8*ec2(x+h,y,z)-ec2(x+2*h,y,z))/(12*h);
}

double dec2y(double x,double y,double z)
{
// derivata prin două puncte, în
// raport cu y, a celei de-a doua ecuaţii.
return
(ec2(x,y-2*h,z)-8*ec2(x,y-h,z)+8*ec2(x,y+h,z)-ec2(x,y+2*h,z))/(12*h);
}

double dec2z(double x,double y,double z)
{
// derivata prin două puncte, în
// raport cu z, a celei de-a doua ecuaţii.
return
(ec2(x,y,z-2*h)-8*ec2(x,y,z-h)+8*ec2(x,y,z+h)-ec2(x,y,z+2*h))/(12*h);
}

double dec3x(double x,double y,double z)
{
// derivata prin două puncte, în
// raport cu x, a celei de-a treia ecuaţii.
return
(ec3(x-2*h,y,z)-8*ec3(x-h,y,z)+8*ec3(x+h,y,z)-ec3(x+2*h,y,z))/(12*h);
}

double dec3y(double x,double y,double z)
{
// derivata prin două puncte, în raport cu y, a celei de-a treia
// ecuaţii.
return
(ec3(x,y-2*h,z)-8*ec3(x,y-h,z)+8*ec3(x,y+h,z)-ec3(x,y+2*h,z))/(12*h);
}


double dec3z(double x,double y,double z)
{
// derivata prin două puncte, în
// raport cu z, a celei de-a treia ecuaţii.
{
return
(ec3(x,y,z-2*h)-8*ec3(x,y,z-h)+8*ec3(x,y,z+h)-ec3(x,y,z+2*h))/(12*h);
}
```

```
double dt3(    double a11, double a12, double a13,
               double a21, double a22, double a23,
               double a31, double a32, double a33){
 return        a11*a22*a33+a21*a32*a13+a12*a23*a31-
               a31*a22*a13-a21*a12*a33-a11*a32*a23;
}

int RezSistN(double *x0,double *y0,double *z0,int MAX,double err)
{
int i;
double dx,dy,dz,det,detx,dety,detz,xn1,yn1,zn1,xn,yn,zn;

i=0;
xn=*x0;yn=*y0;zn=*z0;
do{
xn1=xn;yn1=yn;zn1=zn;
det=    dt3(dec1x(xn1,yn1,zn1),dec1y(xn1,yn1,zn1),dec1z(xn1,yn1,zn1),
        dec2x(xn1,yn1,zn1),dec2y(xn1,yn1,zn1),dec2z(xn1,yn1,zn1),
        dec3x(xn1,yn1,zn1),dec3y(xn1,yn1,zn1),dec3z(xn1,yn1,zn1));
detx=dt3(-ec1(xn1,yn1,zn1),dec1y(xn1,yn1,zn1),dec1z(xn1,yn1,zn1),
        -ec2(xn1,yn1,zn1),dec2y(xn1,yn1,zn1),dec2z(xn1,yn1,zn1),
        -ec3(xn1,yn1,zn1),dec3y(xn1,yn1,zn1),dec3z(xn1,yn1,zn1));
dety=dt3(dec1x(xn1,yn1,zn1),-ec1(xn1,yn1,zn1),dec1z(xn1,yn1,zn1),
        dec2x(xn1,yn1,zn1),-ec2(xn1,yn1,zn1),dec2z(xn1,yn1,zn1),
        dec3x(xn1,yn1,zn1),-ec3(xn1,yn1,zn1),dec3z(xn1,yn1,zn1));
detz=dt3(dec1x(xn1,yn1,zn1),dec1y(xn1,yn1,zn1),-ec1(xn1,yn1,zn1),
        dec2x(xn1,yn1,zn1),dec2y(xn1,yn1,zn1),-ec2(xn1,yn1,zn1),
        dec3x(xn1,yn1,zn1),dec3y(xn1,yn1,zn1),-ec3(xn1,yn1,zn1));
if(det==0) return 0;
dx=detx/det;dy=dety/det;dz=detz/det;
// aplicarea relaţiilor lui Cramer pentru determinarea soluţiilor
// sistemului liniar în necunoscutele dx, dy şi dz.

xn=xn1+dx;yn=yn1+dy;zn=zn1+dz;printf("\n");
printf("%lf\t",xn);printf("%lf\t",yn);printf("%lf",zn);
i++;    // contorizarea numărului de iteraţii.
}
while(((fabs(xn-xn1)>err)||(fabs(yn-yn1)>err)
        ||(fabs(zn-zn1)>err))&&(i<=MAX));

if(i>=MAX) return 0;
*x0=xn;*y0=yn;*z0=zn;
return 1;
}
```

3. DESFĂŞURAREA LUCRĂRII

3.1. PROBLEME LABORATOR

1. Să se implementeze algoritmii 3.1-3.7 în limbajul C.

2. Se dă circuitul electric din figura 3.1 ce are următoarele componente:

$$R_1 = 2.4k\Omega, R_2 = 3.6k\Omega, R_3 = 1.5k\Omega, R_4 = 1k\Omega,$$
$$R_5 = 1.8k\Omega, R_6 = 2.7k\Omega, R_7 = 1.2k\Omega, R_8 = 3k\Omega,$$
$$E_1 = 10V, E_2 = 12V, E_3 = 15V$$

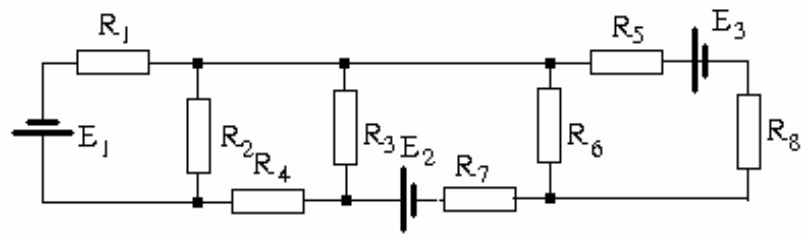


Fig. 3.1 Circuit electric cu surse de c.c. și rezistoare

Se cer curenții prin fiecare latură a circuitului.

3. Se consideră sistemul:

$$4.4x_1 - 0.4x_2 + x_3 + 0.7x_4 = 3.2$$
$$0.23x_1 + 5.4x_2 - 1.5x_3 + 0.1x_4 = 2.7$$
$$1.3x_1 + 0.34x_2 + 6.2x_3 - 0.3x_4 = 1.4$$
$$0.3x_1 + 1.4x_2 - 0.34x_3 + 2.8x_4 = 2.1$$

Să se determine soluțiile sistemului cu ajutorul metodelor care au fost programate într-un limbaj de nivel înalt.

4. Modificând programul scris pentru tei ecuații cu trei necunoscute, să se rezolve sistemul: $\begin{aligned} x^2 + xy &= 3.05 \\ xy + y^2 &= 3.24 \end{aligned}$ luând vectorul de start (1,1).

5. Să se rezolve următorul sistem neliniar folosind algoritmul general de rezolvare a ecuațiilor neliniare (paragraful 2.2.1.2.1), afişându-se vectorul soluțiilor şi eroarea de calcul corespunzătoare fiecărei componente, după *trei paşi* de iterație:

$$x \cdot y + y^2 + 2 \cdot z = 2.48$$
$$x + y \cdot z + 3 \cdot z = 3.8$$
$$x + y + z^2 = 1.8$$

Se va considera vectorul de start: (0.4, -0.3, 0.6). Comentați valabilitatea alegerii acestui vector de start.

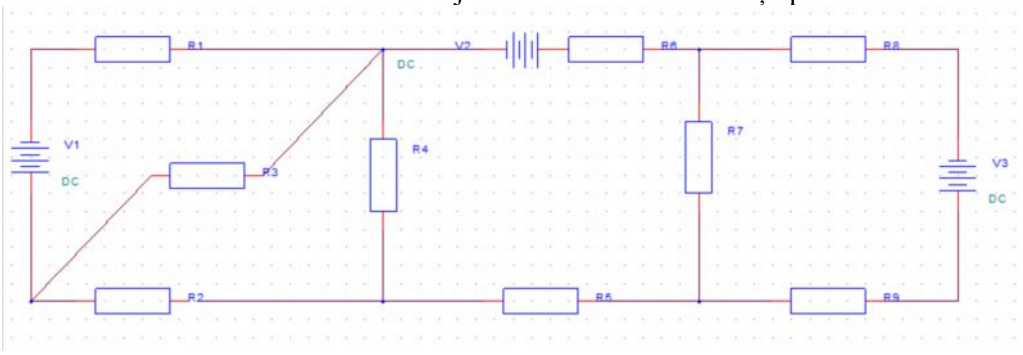
6. Să se rezolve prin metoda lui *Jacobi* şi prin metoda *Gauss-Seidel* următorul sistem:

```
5x +     y +   z  =  4
x +   2y + 8z  =  8
3x +10 y +   z  =  9
```

Comparați viteza de calcul (prin numărul iterațiilor) şi rezultatele. Comparați rezultatele după iterația a 4-a . Se consideră vectorul de start X_0 = (1,1,1).

3.2. TEME DE CASĂ

1. Se consideră circuitul de mai jos. Să se determine curenţii prin fiecare latură.



Se dau:

R_1 = 1 kΩ, R_2 = 5.6 kΩ,, R_3 = 3.6 kΩ, R_4 = 10 kΩ, R_5 = 3 kΩ,
R_6 = 7.5 kΩ, R_7 = 8.2 kΩ, R_8 = 12 kΩ, R_9 = 4.7 kΩ.
V_1 = 5V; V_2 = 9V, V_3 = 12V.

2. Se consideră sistemul liniar pătratic:

$$\begin{aligned} 5x + y + z &= 1 \\ x + y + 4z &= 1.5 \\ x + 6y + z &= 1.6 \end{aligned}$$

Să se rezolve prin metoda lui *Jacobi* şi *Gauss-Seidel*. Vectorul de start este: (1,1,1). Ce alt vector de start se poate alege?

3. Să se rezolve sistemele neliniare următoare, prin *metoda iterativă a lui Newton*:

a. $$\begin{cases} \cos(0.4y + x^2) + x^2 + y^2 - 1.6 = 0 \\ 1.5x^2 - \dfrac{y^2}{0.36} - 1 = 0 \end{cases},$$

cu valorile de start: x_0=1.04, y_0=0.47.

b. $$\begin{cases} e^{x^y} = x^2 - y + 1.1 \\ (x + 0.5)^2 + y^2 = 0.7 \end{cases}$$

Se consideră valori de start: x_0=0.1, y_0=0.1.

BIBLIOGRAFIE

1. RUSU – "*Metode numerice în electronică. Aplicaţii în limbaj C*", Editura Tehnică, Bucureşti, 1997
2. N.V. Kopchenova, I.A. Maron - "*Computational Mathematics - worked examples and problems with elements of theory*", Ed. Mir Publisher Moscow, 1975.