METODE ȘI PROGRAME DE CALCUL NUMERIC

-NOTE DE CURS-

GRECU LUMINIȚA

I. CONCEPTE DE BAZĂ ŞI TIPURI DE ERORI

I.1. INTRODUCERE

Metodele numerice sunt acele tehnici care permit transformarea modelelor matematice în modele numerice (ce operează pe spații finite), şi presupun algoritmi ce pot fi uşor transformați în coduri sursă, folosind diferite limbaje de programare, iar prin intermediul acestora rezolvarea problemelor cărora li se adresează cu ajutorul calculatorului. Pe scurt putem spune că ele permit rezolvarea problemelor matematice cu ajutorul calculatorului.

Trecerea de la modelul matematic la cel numeric se face, în general, pe baza unor aproximări şi de aceea soluția oferită de algoritmii rezultați ca urmare a aplicării metodelor numerice este, de cele mai multe ori, una aproximativă. Ea poartă numele de soluție numerică şi, în cele mai multe situații, este diferită de cea exactă.

I.2. TIPURI DE ERORI

Pentru a înțelege mai bine principalele surse de erori care pot să apară în rezolvarea numerică a problemelor este bine să înțelegem mai întâi modul în care numerele sunt înregistrate şi depozitate în memoria unui calculator. Deoarece numai numere cu un număr finit de zecimale pot fi reprezentate într-un calculator, se foloseşte reprezentarea în virgulă mobilă a numerelor reale.

Reprezentarea în virgulă mobilă (sau flotantă) presupune că numerele sunt reprezentate în calculator sub forma:

$$y = \pm\left(0.x_1x_2\ldots x_t\right)\cdot b^e$$

$$y = \pm b^e \cdot \sum_{k=1}^{t} x_k b^{-k}$$

unde b reprezintă baza de numeraţie iar x_k sunt cifre din axa respectivă, adică valori din mulţimea: $\{0,1,\ldots b-1\}$.

Deci, reprezentarea în virgulă mobilă poate fi sintetizată astfel:

$$y = s \cdot M \cdot b^e,$$

unde s reprezintă bitul de semn (0 pentru + şi 1 pentru -), M este mantisa, b este baza de numeraţie (de obicei 2) iar e este exponentul bazei.

Astfel pentru reprezentarea unui număr este necesar un cuvânt binar cu trei câmpuri: semnul, mantisa şi exponentul.

Lungimea mantisei reprezintă precizia de reprezentare a numărului. Astfel numerele pot avea mai multe formate: formate cu precizie simplă (24 biţi pentru mantisă) şi precizie simplă extinsă (≥ 32 biţi pentru mantisă), şi formate cu precizie dublă (52 biţi) şi precizie dublă extinsă (≥ 64 biţi).

Calculatorul operează cu numere normalizate, adică numere reprezentate astfel încât mantisa să fie un număr subunitar cu prima cifră după virgulă diferită de zero, deci astfel:

$$y = \pm 0.x_1x_2\ldots x_t,\ y = \pm\sum_{k=1}^{t} x_k b^{-k}.$$

De exemplu numărul 1,5 admite mai multe reprezentări (folosind baza 2) printre care: $+1111$, $+111{,}1\cdot 2^1$, $+11{,}11\cdot 2^2$, $+1{,}111\cdot 2^3$, $+0{,}1111\cdot 2^4$ şi $+0{,}001111\cdot 2^6$.

Dintre aceste reprezentări ale numărului 1,5 penultima este cea normalizată.

Când se operează cu numere normalizate acestea trebuie să fie scrise cu ajutorul aceluiaşi exponent. Dacă numerele au exponenţi diferiţi se aduce numărul cu exponenet mai mic la o forma corespunzătoare, ce utilizează exponentul mai mare. Se face operaţia propriu-zisă, şi apoi se normalizează rezultatul.

Exemplul I.1. Să se adune următoarele două numere: $x = 4{,}156832;\ y = 246{,}548$, considerând că ele sunt reprezentate sub formă normalizată într-un sistem de calcul cu virgulă mobilă având baza de numeraţie $b = 10$, $t = 6$.

Soluţie:

Forma normalizată a numerelor este:

$x = 0{,}4156832 \cdot 10 = 0{,}415683 \cdot 10;\ y = 0{,}246548 \cdot 10^3$.

Cifra 2 din scrierea lui x se pierde ţinând cont că *t*, lungimea mantisei, este 6. Observăm că formele normalizate nu prezintă acelaşi exponent. Aducem numărul scris cu exponent mai mic la o formă în care apare exponentul mai mare, şi astfel avem $x = 0{,}004156832 \cdot 10^3$. Păstrând doar 6 cifre semnificative obţinem reprezentarea $x = 0{,}004156 \cdot 10^3$. Efectuăm suma numerelor adunând mantisele: $x + y = 0{,}250704 \cdot 10^3$.

Efectuând calculele în mod obişnuit obţinem:

$x + y = 250{,}704832$.

Exemplul I.2. Se consideră un sistem de calcul cu virgulă mobilă având baza de numeraţie $b = 10$, $t = 4$ şi o reprezentare normalizată a numerelor. Se consideră următoarele numere:

$$x_1 = 0.2146,\ x_2 = 3.175,\ x_3 = 15.421,\ x_4 = 176.86$$

Să se efectueze adunarea acestor numere :

a) în ordine crescătore

b) în ordine descrescătoare

Soluţie:

Formele normalizate ale numerelor sunt:

$$x_1 = 0.2146,\ x_2 = 0.3175 \cdot 10,\ x_3 = 0.1542 \cdot 10^2,\ x_5 = 0.1768 \cdot 10^3$$

a) Adunarea în ordine crescătoare

Se observă că numerele sunt date chiar în ordine crescătoare. Astfel algoritmul este: se adună x_1 cu x_2, iar rezultatul cu x_3. Noul rezultat obţinut se adună cu x_4. Notăm cu a_i (i = 1, 2, 3) rezultatele acestui proces.

$$a_1 = x_1 + x_2 = (0.0214 + 0.3175) \cdot 10 = 0.3389 \cdot 10,$$

$$a_2 = a_1 + x_3 = (0.0338 + 0.1542) \cdot 10^2 = 0.1880 \cdot 10^2,$$

$$a_3 = a_2 + x_4 = (0.0188 + 0.1768) \cdot 10^3 = 0.1956 \cdot 10^3,$$

b) Adunarea în ordine descrescătoare

Algoritmul de calcul este următorul: se adună x_4 cu x_3, iar rezultatul cu x_2. Noul rezultat obţinut se adună cu x_1. Rezultatele intermediare se notează cu b_i (i =1, 2, 3**)**

$$b_1 = x_4 + x_3 = (0.0154 + 0.1768) \cdot 10^3 = 0.1922 \cdot 10^3$$

$$b_2 = b_1 + x_2 = (0.1922 + 0.0031) \cdot 10^3 = 0.1953 \cdot 10^3$$

$$b_3 = b_2 + x_1 = (0.1953 + 0.0002) \cdot 10^3 = 0.1955 \cdot 10^3$$

Concluzie: Adunarea numerelor în calculator nu mai are aceleaşi proprietăţi ca operaţia cunoscută de adunare a numerelor reale. Soluția finală a calculelor generate de modelarea numerică, ce presupune efectuarea unor operaţii aritmetice, care se execută repetat şi în număr finit, poate fi afectată de acestea.

Principalele clase de erori sunt următoarele: erori inerente, erori de metodă, erori de rotunjire şi de trunchiere (sau reziduale).

Erorile inerente nu au legătură cu elaborarea modelului numeric. Ele se pot produce fie când se obţine modelul fizic sau cel matematic, fie când se fac măsuratori, sau chiar când se introduc datele de intrare într-un program de calcul numeric.

Erorile de metodă sunt specifice folosirii metodelor numerice şi algoritmilor de calcul.

Exemplul I.3. Să se evalueze numeric $\int_0^1 \frac{x^n}{2x+9}dx$, pentru n=7.

Soluţie:

Notând cu I_n integrala precedentă se demonstrează cu uşurinţă că are loc relaţia de recurenţă: $I_n = \frac{1}{2n} - \frac{9}{2} I_{n-1}$.

$$\int_0^1 \frac{x^n}{2x+9}dx = \int_0^1 \frac{\frac{1}{2}x^{n-1}(2x+9) - \frac{9}{2}x^{n-1}}{2x+9}dx = \frac{1}{2}\int_0^1 x^{n-1}dx - \frac{9}{2}\int_0^1 \frac{x^{n-1}}{2x+9}dx =$$

$$= \frac{x^n}{2n}\Big|_0^1 - \frac{9}{2}I^{n-1} = \frac{1}{2n} - \frac{9}{2}I_{n-1}$$

Calculăm I_7.

Avem:

$$I_0 = \frac{1}{2} ln(2x+9)\Big|_0^1 = \frac{1}{2}\ln\frac{11}{9} \cong 0{,}10034$$

$$I_1 = \frac{1}{2} - \frac{9}{2}\cdot 0{,}10034 = 0{,}04847$$

$$I_2 = 0{,}25 - \frac{9}{2}\cdot 0{,}04847 = 0{,}031885$$

$$I_3 = \frac{1}{6} - \frac{9}{2}\cdot 0{,}031885 = 0{,}023184$$

$$I_4 = \frac{1}{8} - \frac{9}{2}\cdot 0{,}023184 = 0.020672$$

$$I_5 = \frac{1}{10} - \frac{9}{2}\cdot 0{,}020672 = 0.006976$$

$$I_6 = \frac{1}{12} - \frac{9}{2}\cdot 0.006976 = 0.051941$$

$$I_7 = \frac{1}{14} - \frac{9}{2}\cdot 0.051941 = -0.162306\ .$$

Rezultatul este incorect deoarece integrandul este un număr pozitiv astfel ca integrala este pozitivă.

Eroarea care apare se datorează faptului că eroarea obţinută în calculul lui I_0, prin aproximarea numărului $\ln\frac{11}{9}$ s-a amplificat la fiecare pas al algoritmului datorită produsului $\frac{9}{2} I_{n-1}$. La pasul *n* de calcul eroarea cumulată devine proporţională cu 4.5^n.

Erori de rotunjire şi de trunchiere sunt erorile generate în principal de faptul că dispunem de un spaţiu limitat de reprezentare a numerelor într-un calculator, dar nu numai.

Întâlnim erori de trunchiere atunci când aproximăm o mărime reprezentată în calcule printr-un număr infinit de termeni, printr-un număr finit de astfel de termeni. Eroarea apare datorită termenilor neglijaţi. În mare parte aceste erori sunt acceptate în calculul numeric deoarece nu pot fi evitate.

Un exemplu de astfel de eroare este cea care se face când aproximăm numărul *e* cu o parte finită din suma dată de dezvoltarea lui în serie Taylor: $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$, deci prin înlăturarea unui număr infinit de termeni din această sumă.

Erorile de rotunjire apar deoarece numerele sunt reprezentate în calculator cu un număr redus de cifre semnificative, depinzând de lungimea cuvantului (numărul de biţi) utilizat de calculator.

I.3. EVALUAREA ERORILOR

În calculul numeric noţiunea de eroare are semnificaţia de precizie a aproximării. În calculele numerice sunt acceptate erorile mai mici decât eroarea admisibilă precizată.

Dacă ne raportăm la momentul în care se determină erorile în cadrul aplicării unei metode numerice întâlnim două tipuri de erori:

- apriori- se estimează înainte de începerea calculului propriu-zis;
- aposteriori- se estimează după începerea aplicării algoritmului, după una sau mai multe etape de rezolvare.

Indiferent de natura lor, erorile se evaluează în funcţie de valoarea exactă (reală) x a mărimii de interes şi de valoarea aproximată (calculată) $\bar{x}$ a acestei mărimi. Evaluarea se poate face în mod absolut sau relativ.

Definiţia I.1. Eroarea absolută se notează cu e_x şi este dată de relaţia:

$$\text{a) } e_x = x - \bar{x}\text{, sau b) } e_x = \left| x - \bar{x} \right| \tag{1}$$

Eroarea absolută nu ţine cont de ordinul mărimii studiate şi din acest motiv nu este totdeauna suficientă doar cunoaşterea acesteia.

Definiţia I.2. Eroarea relativă se notează cu ε_x şi este dată de relaţia: $\text{a) } \varepsilon_x = \frac{e_x}{x}$, sau b) $\varepsilon_x = \left| \frac{e_x}{x} \right|$

Uneori eroarea relativă se exprimă, în procente, prin relaţia:

$$\varepsilon_x = \left| \frac{e_x}{x} \right| \cdot 100[\%]. \tag{2}$$

Deoarece valoarea exactă nu se cunoaşte de cele mai multe ori, se introduce în relaţiile precedente valoarea calculată în locul valorii reale şi se obţin relaţiile:

$$\text{a) } \varepsilon_x = \frac{e_x}{\bar{x}}\text{, sau b) } \varepsilon_x = \left| \frac{e_x}{\bar{x}} \right| \text{ sau } \varepsilon_x = \left| \frac{e_x}{\bar{x}} \right| \cdot 100[\%]\text{, în procente.} \tag{3}$$

Exemplul I.5. Să considerăm o mărime, $x = 13$, a cărei valoare calculată este $\bar{x} = 14$. Eroarea absolută este $e_x = 1$. Aceeaşi eroare absolută, $e_y = 1$, se obţine şi dacă de exemplu analizăm o altă mărime *y*, pentru care $y = 1386$, iar $\bar{y} = 1387$.

Folosirea valorii calculate drept aproximare a mărimii reale în prima situație convine mai mult decât în cazul doi, dacă ne raportăm la ordinul de mărime al celor două mărimi analizate. Erorile absolute, fiind egale, nu dovedesc acest lucru, însă erorile relative sugerează asta: $\varepsilon_x = \frac{1}{14} \cong 0.0714$, $\varepsilon_y = \frac{1}{1387} = 7.2098 \cdot 10^{-4}$, şi deci $\varepsilon_y < \varepsilon_x$.

Astfel se realizează o aproximare mai bună a mărimii y prin $\bar{y}$ decât aproximarea mărimii x prin $\bar{x}$.

1.4. CONDIŢIONARE ŞI STABILITATE

Condiţionarea unei probleme se referă la sensibilitatea datelor de ieșire, adică a soluției, față de variațiile (perturbațiile) datelor de intrare. Problemele pot fi bine condiționate, sau slab condiționate. Când variații mici ale datelor de intrare determină variații mici în soluție, problema este bine condiționată, iar când determină variații mari ale soluției ea este slab condiționată.

Astfel pentru diferite clase de probleme de calcul numeric se definesc numere de condiționare care exprimă factorul de amplificare a erorii.

O problemă caracterizată de un număr de condiționare mare va fi o problemă slab condiționată, iar una caracterizată de un număr de condiționare mic (în general subunitar) va fi bine condiționată.

Conceptele de stabilitate sau instabilitate numerică se referă la algoritmii ce însoțesc metodele numerice. Acei algoritmi care se dovedesc a nu amplifica erorile în timpul calculelor pe care le presupun se spune că sunt stabili din punct de vedere numeric, respectiv instabili, dacă produc un astfel de efect.

De exemplu în cazul algoritmilor care oferă soluții aproximative care converg către o limită, se poate întâmpla ca, deşi condițiile de convergență să fie satisfăcute, datorită cumulării erorilor de rotunjire, soluția generată să se depărteze, uneori chiar foarte mult, de această limită. Spunem în acest caz că algoritmul este instabil.

Cele mai mari erori care ar putea să apară în rezolvarea pe cale numerică a problemelor s-ar obține deci în cazul în care s-ar aplica un algoritm instabil la rezolvarea unei probleme slab condiționate.

O astfel de situație trebuie evitată prin găsirea altor metode de soluționare. Pentru a obține o bună soluție numerică la o problemă aceasta trebuie să fie bine condiționată iar algoritmul de rezolvare stabil.

II. METODE DIRECTE PENTRU REZOLVAREA SISTEMELOR DE ECUAŢII LINIARE

II.1. INTRODUCERE

În ceea ce priveşte un sistem de n ecuaţii cu n necunoscute, rezolvarea însăşi a unui astfel de sistem, şi deci posibilitatea de a vedea dacă avem sau nu soluţie, este la fel de costisitoare (din punct de vedere al timpului şi al memoriei) ca și calcularea determinantului sistemului sau a rangului matricei acestuia. De aceea de multe ori se trece direct la soluționarea sistemelor fără ca în prealabil să se studieze existența soluției.

Pentru rezolvarea sistemelor liniare există două clase de metode :

a) Metodele directe sau exacte, cum sunt metodele de tip Gauss, ce furnizează soluţia exactă într-un număr finit de paşi, abstracţie făcând de erorile de rotunjire sau trunchiere care pot să apară.

b) Metodele iterative, precum metodele Jacobi şi Gauss-Seidel, ce aproximează soluţia generând un şir ce converge către aceasta.

Metodele exacte necesită multe operaţii aritmetice, ce duc la acumulări mari ale erorilor de rotunjire, şi presupun timp mare de utilizare a calculatorului şi spaţii mari de memorie.

Metodele iterative sunt mai rapide decât cele exacte, necesită un număr mai mic de operaţii aritmetice, dar e mai greu de precizat

de la început numărul de iteraţii necesare obţinerii unei soluţii suficient de bune.

II.2. REZOLVAREA SISTEMELOR TRIUNGHIULARE

Un sistem de n ecuaţii cu n necunoscute se numeşte triunghiular dacă matricea ataşată acestuia este superior sau inferior triunghiulară.

Definiţia II.1. Matricea $T = (t_{ij}) \in M(C)$ se numeşte superior (inferior) triunghiulară dacă:

$t_{ij} = 0$ pentru $i > j$ $\left(t_{ij} = 0 \text{ pentru } i < j\right)$.

Un sistem de n ecuaţii cu n necunoscute a cărei matrice este superior, respectiv inferior triunghiulară se rezolvă uşor prin substituţie inversă (sau înapoi), respectiv substituţie directă (substituţie înainte).

Vom considera cazul sistemelor ce admit o soluţie unică, deci cazul în care elementele $t_{ii} \neq 0, \forall i = \overline{1,n}$.

Să considerăm un sistem caracterizat de o matrice inferior triunghiulară. Sistemul are în acest caz forma:

$$\begin{cases} t_{11}x_1 & = b_1 \\ t_{21}x_1 + t_{22}x_2 & = b_2 \\ .... & \\ t_{n1}x_1 + t_{n2}x_2 + ... + t_{nn}x_n & = b_n \end{cases}$$

Metoda substituţiei directe presupune obţinerea necunoscutelor succesiv, începând cu prima din ecuaţiile sistemului. Astfel obţinem:

$$x_1 = \frac{b_1}{t_{11}},$$

$$x_2 = \frac{b_2 - t_{21}x_1}{t_{22}},$$

....

$$x_n = \frac{b_n - \sum_{k=1}^{n-1} t_{nk}x_k}{t_{nn}}$$

Algoritmul de rezolvare a sistemelor inferior triunghiulare

Date de intrare: *n*, numărul de ecuaţii şi necunoscute; matricea triunghiulară a sistemului, *T,* care îndeplineşte condiţia $t_{ii} \neq 0, \forall i = \overline{1,n}$; termenii liberi $b_i, i = \overline{1,n}$.

Date de ieşire: soluţia sistemului

1. Citeşte datele de intrare;
2. Iniţializează variabila *i* $i \leftarrow 1$;
3. Atribuie lui $x_1 \leftarrow \frac{b_1}{t_{11}}$;
4. Pentru *i* de la 2 la *n* , cu pasul 1, execută

   4.1.Iniţializează *S* cu 0 $S \leftarrow 0$;

   4.2. Pentru $k = \overline{1, i-1}$, cu pasul 1, execută

   $$s \leftarrow s + t_{ik}x_k;$$

   4.3. Atribuie valoare necunoscutei x_i, $x_i \leftarrow \frac{b_i - S}{t_{ii}}$;

5. Afişează valorile necunoscutelor;
6. Stop.

Analog se obţine şi algoritmul pentru rezolvarea sistemelor superior triunghiulare, denumit substituţie inversă, sau înapoi. Prima necunoscută care se află este x_n, apoi se determină $x_{n-1},...,x_1$. Sistemul se rezolvă practic începând cu ultima ecuaţie:

$$x_n = \frac{b_n}{t_{nn}},$$

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} t_{ij} x_j}{t_{ii}} \quad , \quad i = n-1, n-2, \cdots, 1.$$

Algoritmul de rezolvare a sistemelor superior triunghiulare

Date de intrare: *n*, numărul de ecuaţii şi necunoscute; matricea triunghiulară a sistemului, *T,* care îndeplineşte condiţia $t_{ii} \neq 0, \forall i = \overline{1,n}$; termenii liberi $b_i, i = \overline{1,n}$.

Date de ieşire: soluţia sistemului

1. Citeşte datele de intrare;
2. Iniţializează variabila i, $i \leftarrow n$;
3. Atribuie lui $x_n \leftarrow \frac{b_n}{t_{nn}}$;
4. Pentru *i* de la n-1 la 1 , cu pasul -1, execută
   4.1.Iniţializează *S* cu 0, $S \leftarrow 0$;
   4.2. Pentru $k = \overline{n, i-1}$, cu pasul -1, execută
   $S \leftarrow S + t_{ik} x_k$;

4.3. Atribuie valoare necunoscutei x_i, $x_i \leftarrow \frac{b_i - S}{t_{ii}}$;

5. Afişează valorile necunoscutelor;
6. Stop.

Exemplul II.1.

1) Să se rezolve sistemul de ecuaţii de mai jos prin metoda substituţiei inverse:

$$\begin{aligned} 4x_1 + x_2 - x_3 + x_4 &= 5 \\ 3x_2 - x_3 + x_4 &= 3 \\ 5x_3 + x_4 &= 6 \\ 2x_4 &= 2 \end{aligned}$$

Soluţie:

Matricea coeficienţilor sistemului dat este superior triunghiulară şi deci aplicăm substituţia înapoi. Se începe cu calculul lui x_4, apoi se calculează şi celelalte necunoscute: x_3, x_2, x_1.

Obţinem astfel: $x_4 = 1, x_3 = 1, x_2 = 1, x_1 = 1$.

Folosind limbajul C de programare s-au realizat următoarele programe pentru rezolvarea sistemelor superior, respectiv inferior triunghiulare.

```
/*Program Rezolvare Sistem Superior Triunghiular*/
#include<stdio.h>
#include<conio.h>

        float s,x[30],T[30][30],B[30];
        int i,k,j,n;
        void main()
        {
```

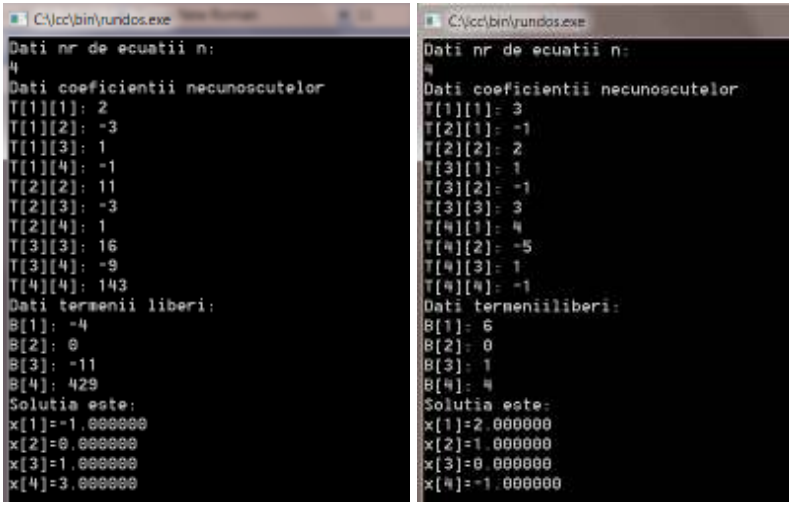
```
printf("Dati nr de ecuatii n:\n");
scanf("%d",&n);
printf("Dati coeficientii necunoscutelor\n");
for (i=1;i<=n;i++)
        for (j=i;j<=n;j++)
        {
                printf("T[%d][%d]: ",i,j);
                scanf("%f",&T[i][j]);
        }
printf("Dati termenii liberi:\n");
for (i=1;i<=n;i++)
        {
                printf("B[%d]: ",i);
                scanf("%f",&B[i]);
        }
x[n]=B[n]/T[n][n];
for (i=n-1;i>=1;i--)
        {   s=0;
                for (k=n;k>=i-1;k--)
                s=s+T[i][k]*x[k];
                x[i]=(B[i]-s)/T[i][i];
        }
printf("Solutia este: \n");
for (i=1;i<=n;i++)
printf("x[%d]=%f \n",i,x[i]);
getch();
}
```

/*Program Rezolvare Sistem Inferior Triunghiular*/

```
#include<stdio.h>
#include<conio.h>
        float s,x[30],T[30][30],B[30];
        int i,k,j,n;
        void main()
        {
        printf("Dati nr de ecuatii n:\n");
        scanf("%d",&n);
```

```
printf("Dati coeficientii necunoscutelor\n");
for (i=1;i<=n;i++)
        for (j=1;j<=i;j++)
        {
                printf("T[%d][%d]: ",i,j);
                scanf("%f",&T[i][j]);
        }
printf("Dati termeniiliberi:\n");
for (i=1;i<=n;i++)
        {
                printf("B[%d]: ",i);
                scanf("%f",&B[i]);
        }
x[1]=B[1]/T[1][1];
for (i=2;i<=n;i++)
        {   s=0;
                for (k=1;k<=i-1;k++)
                s=s+T[i][k]*x[k];
                x[i]=(B[i]-s)/T[i][i];
        }
printf("Solutia este: \n");
for (i=1;i<=n;i++)
printf("x[%d]=%f \n",i,x[i]);
getch();
}
```

În continuare sunt prezentate câteva rezulte obținute pe baza rulării acestor programe:



II.3. ALGORITMUL GAUSS PENTRU REZOLVAREA SISTEMELOR LINIARE. VARIANTA GAUSS-JORDAN.

Metoda lui Gauss este una din cele mai cunoscute metode de rezolvare a sistemelor de ecuaţii liniare, de calcul al determinanţilor şi de determinare a inversei unei matrici. Se poate utiliza inclusiv la aflarea rangului unei matrici. Considerăm un sistem liniar de n ecuaţii cu n necunoscute:

$$Ax = B \text{, unde } A \in M_n(R), B \in M_{n\times 1}(R).$$

Algoritmul Gauss, denumit şi metoda eliminării parţiale sau tehnica de pivotare, constă în a aduce sistemul, prin n-1 transformări elementare, la o formă echivalentă, caracterizată de o matrice superior triunghiulară, notată A', prin eliminarea succesivă a necunoscutelor din ecuaţiile sistemului iniţial.

Pentru aflarea soluţiei sistemului considerat se rezolvă apoi sistemul triunghiular obţinut, $A'x = B'$, prin substituţie inversă.

Transformările succesive la fiecare etapă se aplică matricei extinse a sistemului.

Pentru a prezenta aceste transformări succesive notăm printr-un indice superior pasul curent, astfel că $a_{ij}^{(k)}$ reprezintă elementul de pe linia *i* coloana *j* al matricei de la pasul *k*. Notăm elementele matricei *A*, cu indicele 0 sus, adică $a_{ij} = a_{ij}^0 \ \forall i, j \in \{1,...,n\}$.

Transformările efectuate la etapa *k* pot fi rezumate în formule:

$$a_{ij}^{(k)} = \begin{cases} a_{ij}^{(k-1)} & pentru \quad i \le k, \quad j \in \overline{1,n} \\ 0 & pentru \quad i \ge k+1, \quad j \le k \\ a_{ij}^{(k-1)} - \dfrac{a_{ik}^{(k-1)}}{a_{k,k}^{(k-1)}} a_{k,j}^{(k-1)} & pentru \quad i \ge k+1, \quad j \ge k+1. \end{cases}$$

$$b_i^{(k)} = \begin{cases} b_i^{(k-1)} & pentru \quad i \le k, \\ b_i^{(k-1)} - \dfrac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}} b_k^{(k-1)} & pentru \quad i \ge k+1 \end{cases}$$

Elementul $a_{k,k}^{(k-1)}$ poartă numele de pivot al transformării de la pasul *k*. Evident el trebuie să fie nenul pentru ca algoritmul să funcţioneze.

Se obţine în final un sistem echivalent caracterizat de o matrice superior triunghiulară a cărei soluţie se obţine prin substituţie inversă.

Dacă se întâmplă ca pe parcursul aplicării algoritmului să întâlnim un element nul pe poziţia pivotului se permută linia curentă cu o linie situată sub ea, astefel încât noul element, ce ajunge pe poziţia pivotului, să fie nenul.

Algoritmul de rezolvare a sistemului $Ax = B$ prin metoda Gauss este descris în continuare.

Date de intrare: *n*, matricea *A* şi vectorul coloană format cu termenii liberi, *B*

Date de ieşire: soluţia $(x_1, x_2, ..., x_n)$ sau un mesaj de eroare

1. Pentru i de la 1 la n-1 execută:
    a. Caută p cel mai mic întreg $p = \overline{i,n}$ astfel ca $a_{pi} \neq 0$
    b. Dacă nu există , atunci scrie „sistemul nu are soluţie unică” şi du-te la pasul 4
    c. Dacă $p \neq i$, atunci permută liniile p şi i
    d. Pentru j de la i+1 la n execută
        i. $m_{ji} := \frac{a_{ji}}{a_{ii}}$,
        ii. $linia_j := linia_j - m_{ji} linia_i$
2. Dacă $a_{nn} \neq 0$,atunci

    2.1 $x_n := b_n / a_{nn}$

    2.2 pentru i de la n-1 la 1 execută

$$x_i := \left[b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right] / a_{ii}$$

altfel

2.3 Dacă $b_n = 0$, afisează sistemul nu are soluție unică și du-te la pasul 4

altfel

2.4 afisează sistemul nu are soluție și du-te la pasul 4

3. Afişează $x_1, x_2, ..., x_n$
4. Stop.

Metoda Gauss poate fi folosită şi la calculul determinantului, unei matrici, precum şi la aflarea inversei unei matrici.

Valoarea determinantului unei matrici superior triunghiulare, de forma matricii A' se obţine simplu cu ajutorul relaţiei:

$$\det(A) = \Delta = a_{11} a_{22}^{(1)} a_{33}^{(2)} ... a_{nn}^{(n-1)}.$$

Reamintim că pentru a putea folosi această metodă este necesar ca elementele alese drept pivoţi să fie nenule. În plus, pot apărea erori numerice dacă ele au valori prea mici. Pentru a evita această situaţie se pot inversa ecuaţiile sistemului între ele, fapt care duce însa la schimbarea semnului determinantului.

II.4. TEHNICI DE PIVOTARE

Pentru a micşora erorile ce pot să apară în cazul aplicării metodei Gauss s-au conturat două tehnici de pivotare: pivotarea parţială şi pivotarea totală.

În cazul pivotării parţiale se alege drept pivot elementul de sub diagonală, de pe aceeaşi coloană, maxim în modul. Acesta este adus pe poziţia pivotului printr-o permutare convenabilă de linii, după care algoritmul se continuă în modul obişnuit.

În cazul pivotării totale, la pasul *i* se alege pivot elmentul de pe linia *i* coloana *i* doar dacă acesta este elementul maxim în modul între elementele submatricei obținute la pasul precedent $A_i^{(i-1)} = \left(a_{kl}^{(i-1)}\right)_{i\le k,l\le n}$ (formate cu liniile şi coloanele cu indici mai mari sau egali cu *i*). Dacă nu el nu respectă această condiție se caută un astfel de element în submatricea precizată şi se aduce acesta pe poziţia (*i*, *i*) prin permutări de linii dar şi de coloane.

Observaţia II.1. Permutările de linii nu afectează structura soluţiei sistemului, pe când cele de coloane, da. De aceea permutările de coloane trebuie reţinute, în ordinea efectuării lor, pentru ca, parcurgând în sens invers şirul lor ordonat, şi permutând corespunzător componentele soluţiei sistemului rezultat, să obţinem soluţia sistemului iniţial.

Exemplul II.2.

Aplicând metoda Gauss să se rezolve sistemul:

$$\begin{cases} -x + 2y + z = 2 \\ 2x + y - 4z = 8 \\ 3x - y - z = 2 \end{cases}$$

Soluţie:

$$\begin{array}{cccc}
[-1] & 2 & 1 & 2 \\
2 & 1 & -4 & 8 \\
3 & -1 & -1 & 2 \\
\hline
-1 & 2 & 1 & 2 \\
0 & [5] & -2 & 12 \\
0 & 5 & 2 & 8 \\
\hline
-1 & 2 & 1 & 2 \\
0 & 5 & -2 & 12 \\
0 & 0 & 4 & -4
\end{array}$$

Astfel se obţine forma echivalentă, caracterizată de o matrice superior triunghiulară:

$$\begin{aligned}
-x+2y+z &= 2 \\
5y-2z &= 12 \\
4z = -4 &\Rightarrow z = -1 \\
y &= \frac{12+2\cdot(-1)}{5} = \frac{10}{5} = 2 \\
x &= -2+2\cdot 2+(-1) = 1
\end{aligned}$$

Soluţia sistemului este deci tripletul: $(1\ ,\ 2\ ,\ -1)$

Exemplul II.3

Să se rezolve acelaşi sistem aplicând tehnica pivotării parţiale.

Soluţie:

În cadrul tehnicii de pivotare parţială, se permută între ele liniile sistemului astfel încât la fiecare pas pivotul ales să fie maxim în modul între elementele aflate pe coloana sa, sub linia pe care este situat.

La primul pas se observă că $\max(|-1|,2,3)=3$.

Elementul maxim în modul de pe prima coloană, adică 3, trebuie adus, printr-o permutare de linii, pe prima poziţie. Astfel permutăm liniile 1 şi 3.

Obţinem:

$$\begin{cases} 3x-y-z=2 \\ 2x+y-4z=8 \\ -x+2y+z=2 \end{cases} \Rightarrow$$

$$\begin{array}{cccc|} [3] & -1 & -1 & 2 \\ 2 & 1 & -4 & 8 \\ -1 & 2 & 1 & 2 \\ \hline 3 & -1 & -1 & 2 \\ 0 & [5/3] & -10/3 & 20/3 \\ 0 & 5/3 & 2/3 & 8/3 \\ \hline 3 & -1 & -1 & 2 \\ 0 & 5/3 & -10/3 & 20/3 \\ 0 & 0 & 4 & -4 \end{array}$$

Forma echivalentă a sistemului, dată de ultimul tabel este :

$$\begin{aligned} & 3x-y-z=2 \\ & \tfrac{5}{3}y-\tfrac{10}{3}z=\tfrac{20}{3} \\ & 4z=-4 \Rightarrow z=-1 \\ & y=\frac{20+10(-1)}{5}=2 \\ & x=\frac{2+2+(-1)}{3}=1 \end{aligned}$$

Obţinem astfel soluţia unică: $(1,2,-1)$.

Exemplul II.4

Să se rezolve acelaşi sistem aplicând tehnica pivotării totale.

Soluţie:

La primul pas se caută elementul maxim în modul din toată matricea sistemului. Se observă că elementul maxim în modul este 4. Acesta trebuie adus pe linia 1 coloana 1. Astfel se permută liniile 1 şi 2, rezultând:

$$\begin{cases} 2x + y - 4z = 8 \\ 3x - y - z = 2 \\ -x + 2y + z = 2 \end{cases}$$

Apoi se permută coloanele 1 şi 3. Reţinem această permutare de coloane($C_1 \leftrightarrow C_3$).

Sistemul devine:

$$\begin{cases} -4x_1 + x_2 + 2x_3 = 8 \\ -x_1 - x_2 + 3x_3 = 2 \\ x_1 + 2x_2 - x_3 = 2 \end{cases}$$

$$\begin{array}{ccc|c} [-4] & 1 & 2 & 8 \\ -1 & -1 & 3 & 2 \\ 1 & 2 & -1 & 2 \\ \hline -4 & 1 & 2 & 8 \\ 0 & [-5/4] & 10/4 & 0 \\ 0 & 9/4 & -2/4 & 4 \end{array}$$

La o aplicare a tehnicii de pivotare obişnuite, la pasul doi ar trebui ales drept pivot elementul $-5/4$.

Se observă însă că elementul maxim în modul al submatricei

$$\begin{pmatrix} -5/4 & 10/4 \\ 9/4 & -2/4 \end{pmatrix} \text{ este } 10/4$$

Acesta va fi pivotul următoarei transformări. El trebuie adus pe linia 2, coloana 2. Permutăm pentru aceasta coloana 2 cu coloana 3, şi fiind vorba de o permutare de coloane o reţinem şi pe aceasta ($C_2 \leftrightarrow C_3$). Obţinem:

$$\begin{array}{ccc|c} -4 & 2 & 1 & 8 \\ 0 & \boxed{10/4} & -5/4 & 0 \\ 0 & -2/4 & 9/4 & 4 \\ \hline -4 & 2 & 1 & 8 \\ 0 & 10/4 & -5/4 & 0 \\ 0 & 0 & 2 & 4 \end{array}$$

Forma echivalentă (scrisă pentru alte necunoscute decât cele iniţiale, deoarece au avut loc permutări de coloane) este :

$$(*)\begin{cases} -4y_1 + 2y_2 + y_3 = 8 \\ \quad 10/4\, y_2 - 5/4\, y_3 = 0 \Rightarrow \\ \quad\quad 2y_3 = 4 \end{cases}$$

$$\begin{cases} y_3 = 2 \\ y_2 = \dfrac{5\cdot(2)}{+10} = 1 \\ y_1 = \dfrac{2+2-8}{4} = \dfrac{-4}{4} = -1 \end{cases}$$

Soluţia sistemului $(*)$ este tripletul: $(-1,1,2)$.

Revenim la permutările de coloane pe care le-am făcut.

Parcurgându-le în ordine inversă, şi efectuându-le asupra componentelor soluţiei, obţinem în final soluţia sistemului iniţial.

Astfel avem :

$$(-1,1,2) \overset{C_2 \to C_3}{\to} (-1,2,1) \overset{C_1 \to C_3}{\to} (1,2,-1).$$

Astfel soluţia sistemului iniţial este tripletul $(1,2,-1)$.

Exemplul II.5.

Folosind tehnica pivotării să se rezolve sistemul :

$$\begin{cases} 2x-3y+z-t=-4 \\ x+4y-z=-2 \\ -2x-y+3z-t=2 \\ -x+2y+t=4 \end{cases}$$

Soluţie:

$[2]$	-3	1	-1	-4
1	4	-1	0	-2
-2	-1	3	-1	2
-1	2	0	1	4
2	-3	1	-1	-4
0	$[11/2]$	$-3/2$	$1/2$	0
0	-4	4	-2	-2
0	$1/2$	$1/2$	$1/2$	2
2	-3	1	-1	-4
0	11	-3	1	0
0	0	$[32/11]$	$-18/11$	$-22/11$
0	0	$7/11$	$5/11$	$22/11$
2	-3	1	-1	-4
0	11	-3	1	0
0	0	$32/11$	$-18/11$	$-22/11$
0	0	0	$143/176$	$429/176$

Se obţine astfel următoarea formă echivalentă pentru sistem:

$$\begin{cases} 2x-3y+z-t=-4 \\ 11y-3z+t=0 \\ 32/11\,z-18/11\,t=-2 \\ 143/176\,t=429/176 \end{cases}$$

Prin substituţie inversă obţinem soluţia: $(-1,0,1,3)$.

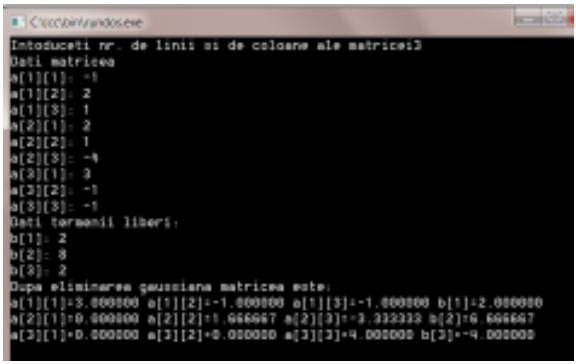
Un cod sursă (în C) pentru algoritmul eliminării gaussiene-varianta pivotării parţiale (până la obținerea matricei superior triunghiulare) este prezentat în continuare.

/*Program Eliminare Gaussiana*/

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
double t,m;
int i,j,k,l,n;
float s, a[10][10], b[10];
void main()
{
printf ("Intoduceti nr. de linii si de coloane ale matricei");
scanf("%d",&n);
printf("Dati matricea\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
printf("a[%d][%d]: ",i,j);
scanf("%f",&a[i][j]);
}
printf("Dati termenii liberi:\n");
for (i=1;i<=n;i++)
        {
        printf("b[%d]: ",i);
```

```
        scanf("%f",&b[i]);
        }
for(k=1;k<=n;k++)
{
l=k;
for (i=k+1;i<=n;i++)
if (abs(a[i][k])>abs(a[k][k]))
l=i;
if(l!=k)
{
for(j=1;j<=n;j++)
{
t=a[k][j];a[k][j]=a[l][j];a[l][j]=t;
}
t=b[k];b[k]=b[l];b[l]=t;
}
for(i=k+1;i<=n;i++)
{
m=a[i][k]/a[k][k];
a[i][k]=0;
for(j=k+1;j<=n;j++)
a[i][j]=a[i][j]-m*a[k][j];
b[i]=b[i]-m*b[k];
}
}
printf("Dupa eliminarea gaussiana matricea este: \n");
for (i=1;i<=n;i++)
                {
for (j=1;j<=n;j++)
        printf("a[%d][%d]=%f ",i,j,a[i][j]);
printf("b[%d]=%f ",i,b[i]);
printf("\n");
                }
                getch();
        }
```

Soluționarea problemei de la exemplul II.3 cu ajutorul programului de calul numeric anterior este prezentată în continuare.



II.5. VARIANTA GAUSS-JORDAN.

Transformările efectuate sunt asemănătoare cu cele din cadrul metodei Gauss, însă afectează toată matricea de la pasul curent, în final obținîndu-se un o matrice diagonală.

Formulele de transformare iterativă a elementelor matricei *A* la etapa *k* sunt :

$$\begin{cases} a_{kj}^{(k)} = a_{kj}^{(k-1)}, \ pentru \ j \geq 1 \\ a_{ij}^{(k)} = a_{ij}^{(k-1)} - \dfrac{a_{i,k}^{(k-1)}}{a_{k,k}^{k-1}} a_{k,j}^{(k-1)}, \ pentru \ 1 \leq i \leq n, i \neq k, j \geq k+1 \\ a_{ij}^{(k)} = a_{ij}^{(k-1)}, \ pentru \ j < k \\ a_{ik}^{(k)} = 0, \ pentru \ 1 \leq i \leq n, i \neq k \end{cases}$$

iar membrul drept se modifică după relaţiile :

$$\begin{cases} b_k^{(k)} = b_k^{(k-1)} \\ b_i^{(k)} = b_i^{(k-1)} - \dfrac{a_{i,k}^{(k-1)}}{a_{k,k}^{(k-1)}} b_k^{(k-1)} \quad pentru \quad i \neq k \end{cases}$$

În acest caz soluţia sistemului va fi : $x_i = \dfrac{b_i^{(n-1)}}{a_{ii}^{(n-1)}}$, $i = \overline{1,n}.$

Metoda Gauss-Jordan mai poartă denumirea de metoda eliminării complete. Se poate aplica şi uşor modificată în sensul că se poate obţine în final un sistem caracterizat de matricea unitate. Pentru aceasta linia pivotului se împarte la pivot. Se fac într-adevăr mai multe operaţii algebrice asupra matricei sistemului, faţă de varianta prezentată, dar se găseşte direct soluţia acestuia ($x_i = b_i'^{(n-1)}, i = \overline{1,n}$).

Exemplul II.6.

Să se rezolve următoarele sisteme folosind metoda Gauss-Jordan.

a) $\begin{cases} x_1 - x_2 + x_3 = 4 \\ 2x_1 + x_2 - x_3 = 1 \\ -x_1 + 2x_2 + x_3 = 6 \end{cases}$, b) $\begin{cases} x_1 - x_2 + x_3 + 2x_4 = 3 \\ x_2 - 2x_3 + x_4 = 2 \\ 2x_1 + 2x_2 - x_3 + x_4 = 1 \end{cases}$.

Soluţie:

a) Obţinem succesiv următoarele tablouri:

$$
\begin{array}{ccc|c}
1 & -1 & 1 & 4 \\
2 & 1 & -1 & 1 \\
-1 & 2 & 1 & 6 \\
\hline
1 & -1 & 1 & 4 \\
0 & 3 & -3 & -7 \\
0 & 1 & 2 & 10 \\
\hline
1 & 0 & 0 & 5/3 \\
0 & 1 & -1 & -7/3 \\
0 & 0 & 3 & 37/3 \\
\hline
1 & 0 & 0 & 15/9 \\
0 & 1 & 0 & 16/9 \\
0 & 0 & 1 & 37/9
\end{array}
$$

În acest caz ultima formă echivalentă pentru sistem este una diagonală, cu elemente unitate, deci soluţia se obţine direct. Astfel,

$$x_1 = \frac{15}{9}, \quad x_2 = \frac{16}{9}, \quad x_3 = \frac{37}{9}$$

reprezintă soluţia unică a sistemului.

b) În acest caz avem un sistem cu 3 ecuaţii şi 4 necunoscute. Procedând analog în cazul sistemului de la punctul b) găsim:

$$\begin{array}{cccc|c}
1 & -1 & 1 & 2 & 3 \\
0 & 1 & -2 & 1 & 2 \\
2 & 2 & -1 & 1 & 1 \\
\hline
1 & -1 & 1 & 2 & 3 \\
0 & 1 & -2 & 1 & 2 \\
0 & 4 & -3 & -3 & -5 \\
\hline
1 & 0 & -1 & 3 & 5 \\
0 & 1 & -2 & 1 & 2 \\
0 & 0 & 5 & -7 & -13 \\
\hline
1 & 0 & 0 & 8/5 & 12/5 \\
0 & 1 & 0 & 1/5 & -16/5 \\
0 & 0 & 1 & -7/5 & -13/5
\end{array}$$

O formă echivalentă pentru sistem este:

$$\begin{cases} x_1 + \frac{8}{5}x_4 = \frac{12}{5} \\ x_2 - \frac{7}{5}x_4 = -\frac{16}{5} \\ x_3 - \frac{7}{5}x_4 = -\frac{13}{5} \end{cases} \Rightarrow \text{rang}A=3,\ \text{rang}\,\overline{A}=3,$$

Sistemul este deci compatibil simplu nedeterminat. Necunoscutele principale sunt x_1, x_2, x_3 iar necunoscuta secundară x_4. Alegem $x_4 = \alpha$.

$$x_4 = \alpha \Rightarrow S: \begin{cases} x_1 = \frac{12}{5} - \frac{8}{5}\alpha \\ x_2 = \frac{-16}{5} + \frac{9}{5}\alpha \\ x_3 = -\frac{13}{5} + \frac{7}{5}\alpha \\ x_4 = \alpha \end{cases}$$

III. METODE NUMERICE ÎN CALCULUL MATRICEAL

III.1. FACTORIZAREA MATRICELOR

Prin factorizarea unei matrici înțelegem scrierea acesteia ca un produs de două matrice: una inferior triunghiulară cealaltă superior triunghiulară (denumită factorizare *LR*).

Definiţia III.1. Fie $A \in M_n(R)$, reprezentarea lui A sub forma:

$$A = L \cdot R,$$

cu *L*- matrice inferior triunghiulară şi *R*- matrice superior triunghiulară, reprezintă factorizarea LR a lui A .

Factorizarea unei matrici se foloseşte în soluţionarea sistemelor liniare cu *n* ecuaţii şi *n* necunoscute.

Considerând sistemul $Ax=B$, prin înlocuirea lui A în relaţia precedentă cu expresia obţinută prin factorizare, obţinem:

$$LRx=B$$

Astfel rezolvarea sistemului $Ax = B$ presupune descompunerea acestuia în rezolvarea a doua sisteme:

1) $Ly = B$ (sistem caracterizat de o matrice inferior triunghiulară, care se rezolvă printr-o "substituţie înainte");

2) $Rx = Y$ (sistem caracterizat de o matrice superior triunghiulară, care se rezolvă printr-o "substituţie înapoi").

Teorema III.1. Dacă matricea A are toţi minorii diagonali principali nenuli, atunci există o factorizare LR a lui A în care L este nesingulară.

Pentru determinarea matricelor L şi R există două tipuri importante de algoritmi asemănători, care diferă doar datorită aspectului matricelor rezultate în urma descompunerii:

1) Metoda Crout – caz în care diagonala lui R este unitară.
2) Metoda Doolittle –caz în care diagonala lui L este unitară

1) **Factorizarea Crout (metoda Crout)**

În acest caz avem următoarea descompunere $A = L \cdot R$ cu

$$L = \begin{pmatrix} l_{11} & 0 & 0 & . & . & 0 \\ l_{21} & l_{22} & 0 & . & . & 0 \\ l_{31} & l_{32} & l_{33} & . & . & 0 \\ . & . & . & . & . & . \\ l_{n1} & l_{n2} & l_{n3} & . & . & l_{nn} \end{pmatrix} \quad R = \begin{pmatrix} 1 & r_{12} & r_{13} & . & . & r_{1n} \\ 0 & 1 & r_{23} & . & . & r_{2n} \\ 0 & 0 & 1 & . & . & r_{3n} \\ . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & 1 \end{pmatrix}$$

Determinarea elementelor se face impunând condiţia ca rezultatul produsului celor două matrice L şi R, în această ordine, să fie egal cu matricea A, adică:

$$l_{i1}r_{1j} + l_{i2}r_{2j} + \ldots + l_{in}r_{nj} = a_{ij},\ 1 \le i, j \le n$$

Deoarece se ştie că $l_{ij} = 0$ pentru $j \ge i+1$, $r_{ij} = 0$ pentru $j \le i-1$ şi $r_{ii} = 1, 1 \le i \le n$, obţinem: $a_{ij} = \sum_{k=1}^{\min(i,j)} l_{ik}r_{kj}$ şi de aici relaţiile ce determină elementele necunoscute ale celor două matrice:

- Pentru matricea inferior triunghiulară:

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} r_{kj}\ ,\ j \le i\ ,$$

unde pentru j=1 se consideră $\sum_{k=1}^{0} = 0$ şi $l_{i1} = a_{i1}$

- Pentru matricea superior triunghiulară:

$$r_{ij} = \left[a_{ij} - \sum_{k=1}^{i-1} l_{ik} r_{kj} \right] \cdot l_{ii}^{-1}\ ,\ j \ge i+1\ ,$$

unde pentru i=1 se consiedră $\sum_{k=1}^{0} = 0$ şi $r_{1j} = \frac{a_{1j}}{a_{11}}$

Pentru a face mai accesibilă aplicarea algoritmului de factorizare prezentat îl vom detalia, insistând asupra unor reguli practice, uşor de reţinut.

Observaţia III.1. Determinarea practică a matricelor se face în paralel, astfel că la fiecare pas se determină o coloană a lui L şi o linie a lui R, în modul următor:

Algoritmul factorizării Crout

Pasul 1:

- Se determină prima coloana a lui L, care coincide cu cea a lui A.
- Se determină prima linie a lui R, folosind formulele date. Înmulţim practic prima linie a lui L cu toate coloanele lui R ce au indicele mai

mare strict decât 1 şi egalăm elementele corespunzătoare cu cele ale lui *A*.

Pasul 2:

- Se determină a doua coloană a lui *L*, înmulţind toate liniile lui *L*, de la 2 la *n*, cu a doua coloană a lui *R* şi egalând elementele corespunzătoare cu cele ale lui A.

- Se determină a doua linie a lui *R*, înmulţind a doua linie a lui L cu toate coloanele lui *R* de la 3 la *n*, şi egalând elementele corespunzătoare cu cele ale lui *A*.

...,etc.

Pasul *n*:

- Se determină ultima coloană a lui *L* (practic elementul l_{nn}).

Observaţia III.2. Numărul de elemente calculate, şi implicit numărul de operaţii ce se efectuează la fiecare pas, scade pe măsura creşterii numărului de ordine al pasului, astfel că la ultimul pas se calculează de fiecare dată doar un singur element.

Exemplul III.1. Folosind metoda Crout să se factorizeze matricea

$$A=\begin{pmatrix} 2 & -1 & 2 \\ 1 & 3 & 1 \\ -1 & 2 & 1 \end{pmatrix}$$

Soluţie:

Calculând minorii diagonali principali oservăm că ei respectă condiţia din teoremă**.**

Fie cele două matrice *L* şi *R*.

$$L=\begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \qquad R=\begin{pmatrix} 1 & r_{12} & r_{13} \\ 0 & 1 & r_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

Pasul 1:

-Se determină prima coloană a lui *L*. Avem relaţiile:

$$l_{11}=a_{11}=2\Rightarrow l_{11}=2$$

$$l_{21}=a_{21}=1\Rightarrow l_{21}=1$$

$$l_{31}=a_{31}=-1\Rightarrow l_{31}=-1$$

-Se determină prima linie a lui *R* :

$$l_{11}\cdot r_{12}=a_{12}=-1\Rightarrow r_{12}=\frac{-1}{l_{11}}=\frac{-1}{2},\ \Rightarrow r_{12}=-\frac{1}{2}$$

$$l_{11}\cdot r_{13}=a_{13}=2\Rightarrow r_{13}=\frac{2}{2}=1$$

Pasul 2:

-Se determină a doua coloană a lui *L*.

$$l_{21}\cdot r_{12}+l_{22}=a_{22}=3,\ \Rightarrow l_{22}=3+\frac{1}{2}=\frac{7}{2}$$

$$l_{31}\cdot r_{12}+l_{32}=a_{32}=2\ \Rightarrow l_{32}=2-\frac{1}{2}=\frac{3}{2}$$

-Se determină a doua linie a lui *R* .

$$l_{21}\cdot r_{13}+l_{22}\cdot r_{23}=a_{23}=1\Rightarrow r_{23}=\frac{1-1\cdot 1}{\frac{7}{2}}=0.$$

Pasul 3:

-Se determină a treia coloană a lui *L*, adică elementul l_{33}.

$l_{31} \cdot r_{13} + l_{32} \cdot r_{23} + l_{33} = a_{33} = 1 \Rightarrow l_{33} = 1 + 1 = 2$.

Am obținut următoarea factorizare pentru matricea A:

$$A = L \cdot R, \text{ unde } L = \begin{pmatrix} 2 & 0 & 0 \\ 1 & \frac{7}{2} & 0 \\ -1 & \frac{3}{2} & 2 \end{pmatrix} \text{ şi } R = \begin{pmatrix} 1 & -\frac{1}{2} & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

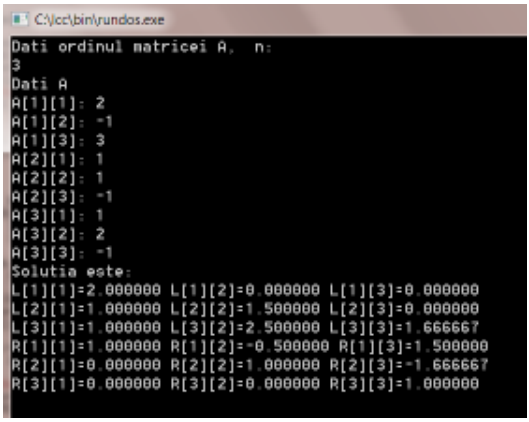
/*Program Factorizare Crout*/

```
#include<stdio.h>
#include<conio.h>
float s, A[30][30], L[30][30], R[30][30];
int i,k,j,n;
        void main()
                {
                printf("Dati ordinul matricei A,  n:\n");
                scanf("%d",&n);
                printf("Dati A\n");
                for (i=1;i<=n;i++)
                        for (j=1;j<=n;j++)
                        {
                                printf("A[%d][%d]: ",i,j);
                                scanf("%f",&A[i][j]);
                        }
                for (i=1;i<=n;i++)
                   L[i][1]=A[i][1];
                           for (i=1;i<=n;i++)
                    {
                         R[1][i]=A[1][i]/L[1][1];
                        R[i][i]=1;
                    }
                 for (j=2;j<=n;j++)
```

```
        {
          for (i=j;i<=n;i++)
           {
                  s=0;
                  for (k=1;k<=j-1;k++)
                  s=s+L[i][k]*R[k][j];
                  L[i][j]=A[i][j]-s;
           }
          for (i=j+1;i<=n;i++)
          {
                  s=0;
                  for (k=1;k<=i-1;k++)
                  s=s+L[j][k]*R[k][i];
                  R[j][i]=(A[j][i]-s)/L[j][j];
            }
          }
printf("Solutia este: \n");
for (i=1;i<=n;i++)
          {
          for (j=1;j<=n;j++)
          printf("L[%d][%d]=%f ",i,j,L[i][j]);
          printf("\n");
          }
for (i=1;i<=n;i++)
          {
          for (j=1;j<=n;j++)
          printf("R[%d][%d]=%f ",i,j,R[i][j]);
          printf("\n");
          }
getch();
}
```

Folosirea programului anterior pentru factorizarea matricei

$A = \begin{pmatrix} 2 & -1 & 2 \\ 1 & 3 & 1 \\ -1 & 2 & 1 \end{pmatrix}$, conduce la următoarele rezultate:



2) **Factorizarea Doolittle (metoda Doolittle)** :

În acest caz matricele L şi R implicate sunt următoarele:

$$L=\begin{pmatrix} 1 & 0 & 0 & - & - & 0 \\ l_{21} & 1 & 0 & - & - & 0 \\ l_{31} & l_{32} & 1 & - & - & 0 \\ - & - & - & - & - & - \\ l_{n1} & l_{n2} & - & - & - & 1 \end{pmatrix} \qquad R=\begin{pmatrix} r_{11} & r_{12} & r_{13} & - & - & r_{1n} \\ 0 & r_{22} & r_{23} & - & - & r_{2n} \\ 0 & 0 & r_{33} & - & - & r_{3n} \\ - & - & - & - & - & - \\ 0 & 0 & - & - & - & r_{nn} \end{pmatrix}$$

Determinarea elementelor lor se face analog, diferă doar ordinea de determinare a acestora.

Algoritmul factorizării Doolittle

Pasul 1:

- Se determină prima linie a lui R.
- Se determină prima coloană a lui L .

Pasul 2:

- Se determină a doua linie a lui R .
- Se determină a doua coloană a lui L .

.....

Pasul n:

- Se determină linia n a lui R, practic elementul r_{nn} al lui R.

Exemplul III.2. Fie A cea din exemplul III.1:

$$A=\begin{pmatrix} 2 & -1 & 2 \\ 1 & 3 & 1 \\ -1 & 2 & 1 \end{pmatrix}$$

Fie $L=\begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix}$ $R=\begin{pmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{pmatrix}$

Pasul 1:

Se determina prima linie a lui R

$$r_{11}=2,\quad r_{12}=-1,\quad r_{13}=2$$

Se determină prima coloană a lui L

$$l_{21}r_{11}=a_{21}\Rightarrow l_{21}=\frac{1}{2},\ l_{31}r_{11}=a_{31}\Rightarrow l_{31}=\frac{-1}{2}$$

Pasul 2:

Se determina a doua linie a lui R .

$$l_{21}r_{12}+r_{22}=a_{22}\Rightarrow r_{22}=3+\frac{1}{2}=\frac{7}{2}$$

$$l_{21}r_{13}+r_{23}=a_{23}\Rightarrow r_{23}=1-1=0$$

Se determina a doua coloană a lui L

$$l_{31}r_{12} + l_{32}r_{22} = a_{32} \Rightarrow l_{32} = \frac{\left(2 - \frac{1}{2}\right)}{\frac{7}{2}} = \frac{3}{7}$$

Pasul *3*

Se determină r_{33}

$l_{31}r_{13} + l_{32}r_{23} + r_{33} = a_{33} \Rightarrow r_{33} = 1 + 1 - 0 = 2$

Obținem astfel matricele:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ -1/2 & 3/7 & 1 \end{pmatrix} \quad R = \begin{pmatrix} 2 & -1 & 2 \\ 0 & 7/2 & 0 \\ 0 & 0 & 2 \end{pmatrix}.$$

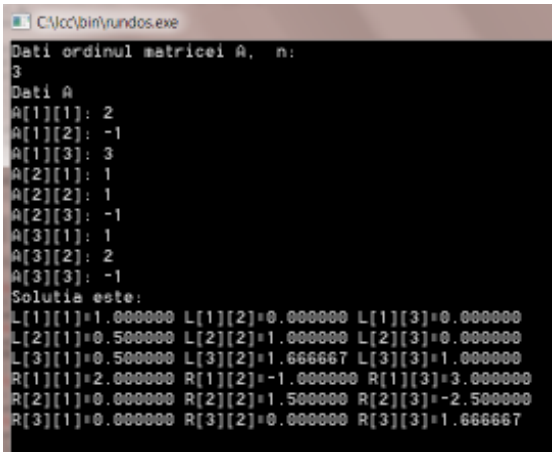
/*Program Factorizare Doolittle*/

```
#include<stdio.h>
#include<conio.h>
float s, A[30][30], L[30][30], R[30][30];
int i,k,j,n;
void main()
{
        printf("Dati ordinul matricei A,  n:\n");
        scanf("%d",&n);
        printf("Dati A\n");
        for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
                {
                  printf("A[%d][%d]: ",i,j);
                  scanf("%f",&A[i][j]);
                  }
        for (j=1;j<=n;j++)
               R[1][j]=A[1][j];
```

```
for (j=1;j<=n;j++)
            {
            L[j][1]=A[j][1]/R[1][1];
            L[j][j]=1;
            }
for (i=2;i<=n;i++)
{
        for (j=i;j<=n;j++)
                {
                  s=0;
                  for (k=1;k<=i-1;k++)
                  s=s+L[i][k]*R[k][j];
                  R[i][j]=A[i][j]-s;
                }
        for (j=i+1;j<=n;j++)
                {
                  s=0;
                  for (k=1;k<=j-1;k++)
                  s=s+L[j][k]*R[k][i];
                  L[j][i]=(A[j][i]-s)/R[i][i];
                  }
}
printf("Solutia este: \n");
for (i=1;i<=n;i++)
        {
          for (j=1;j<=n;j++)
          printf("L[%d][%d]=%f ",i,j,L[i][j]);
          printf("\n");
          }
for (i=1;i<=n;i++)
          {
            for (j=1;j<=n;j++)
            printf("R[%d][%d]=%f ",i,j,R[i][j]);
            printf("\n");
          }
getch();
}
```

Rularea programului anterior pentru factorizarea Doolittle a matricei

$A = \begin{pmatrix} 2 & -1 & 3 \\ 1 & 1 & -1 \\ 1 & 2 & -1 \end{pmatrix}$, conduce la rezultatele de mai jos.



3) Factorizarea Cholesky (sau metoda rădăcinii pătrate)

<u>Teorema III.2.</u> Dacă $A \in M_n(R)$ e simetrică şi pozitiv definită, adică dacă:

$A = A^t$, $x^t Ax \geq 0\ (\forall) x \in R^n$ şi $x^t Ax = 0 \Leftrightarrow x = 0$,

atunci există o factorizare *LR* a lui A cu $R = L^t$, deci $A = L \cdot L^t$

Astfel, în aceste condiţii avem:

$$A=\begin{pmatrix} l_{11} & 0 & - & - & 0 \\ l_{21} & l_{22} & - & - & 0 \\ - & - & - & - & - \\ l_{n1} & l_{n2} & - & - & l_{nn} \end{pmatrix} \cdot \begin{pmatrix} l_{11} & l_{21} & - & - & l_{n1} \\ 0 & l_{22} & - & - & l_{n2} \\ - & - & - & - & - \\ 0 & 0 & - & - & l_{nn} \end{pmatrix}.$$

$$l_{11} == \sqrt{a_{11}}\ , ...,\ l_{ii} = \sqrt{a_{ii} - \sum_{j=1}^{i-1} l_{jj}^2}\ ,\ i = \overline{2,n}\ .$$

$$l_{11}l_{21} = a_{12} \Rightarrow l_{21} = \frac{a_{12}}{l_{11}} \Rightarrow l_{i1} = \frac{a_{1i}}{l_{11}}, (\forall) i = \overline{2,n} \Rightarrow$$

Prima coloană şi prima linie se calculează simultan. La pasul următor se determină mai întâi l_{22}, apoi şi restul elementelor de pe coloana 2 a lui *L.* Se continuă analog pentru toţi paşii până la pasul *n*.

În continuare este prezentat un program în C realizat pe baza algoritmului factorizării Cholesky.

```
/*Program Factorizare Cholesky*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
        float s, sum,A[30][30], L[30][30];
        int i,k,j,n;
        void main()
        {
        printf("Dati ordinul matricei A,  n:\n");
        scanf("%d",&n);
        printf("Introduceti Asimetrica si pozitiv definita\n");
        for (i=1;i<=n;i++)
```

```
        for (j=1;j<=n;j++)
        {
                printf("A[%d][%d]: ",i,j);
                scanf("%f",&A[i][j]);
        }
        j=1;
        L[j][j]=sqrt(A[1][1]);
        for (i=2;i<=n;i++)
             L[i][j]=A[1][i]/L[1][1];

              for (j=2;j<=n;j++)
                {
                        s=0;
                        for (k=1;k<=j-1;k++)
                        s=s+L[j][k]*L[j][k];
                        L[j][j]=sqrt(A[j][j]-s);
                for (i=j+1;i<=n; i++)
                     {
                        sum=0;
                        for (k=1;k<=j-1;k++)
                                sum=sum+L[i][k]*L[j][k];
                        L[i][j]=(A[i][j]-sum)/L[j][j];
                      }
                 }
printf(" Matricea L este: \n");
for (i=1;i<=n;i++)
        {
                for (j=1;j<=n;j++)
                printf("L[%d][%d]=%f ",i,j,L[i][j]);
                printf("\n");
        }
getch();
}
```

III. 2. METODE NUMERICE PENTRU AFLAREA INVERSEI UNEI MATRICI

III.2.1. Matrice elementare

Dacă permutăm liniile sau coloanele matricei unitate, sau dacă facem o serie de combinaţii liniare cu acestea, obţinem o serie de matrice ce se numesc matrice elementare. Aceste matrice elementare, notate în general cu E sunt folosite pentru a introduce o serie de schimbări în structura unei matrici date, fără a-i schimba rangul, sau determinantul.

De exemplu, pentru a înmulţi linia a treia dintr-o matrice $A = (a_{ij})_{1 \le i, j \le 3}$ cu un scalar λ, construim matricea elementară care să aiba pe linia a treia în loc de 1 valoarea λ.

$$E = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \lambda \end{pmatrix}$$

Făcând produsul dintre E si A, în această ordine, obţinem schimbarea propusă :

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \lambda \end{pmatrix} \cdot \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ \lambda a_{31} & \lambda a_{32} & \lambda a_{33} \end{pmatrix}$$

Să construim acum o serie de matrice elementare care să inducă în matricea data A următoarele schimbari :

E_1 să înmulţească linia a doua cu 3 ;

E_2 să adune linia a doua la linia a treia ;

E_3 să permute linia a treia cu a doua.

$$E_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix},\ E_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix},\ E_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Aplicăm succesiv aceste transformări elementare matriciei *A*. Se obţine următorul rezultat :

$$E_3 \cdot E_2 \cdot E_1 \cdot A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 3a_{21} + a_{31} & 3a_{22} + a_{32} & 3a_{23} + a_{33} \\ 3a_{21} & 3a_{22} & 3a_{23} \end{pmatrix}$$

Matricele elemetare pot fi folosite pentru aflarea inversei unei matrici date, în anumite condiţii restrictive.

III.2.2. Metoda transformarilor elementare successive pentru aflarea inversei unei matrici

Această metodă constă în construirea a *n* matrice elementare $E_1, E_2, \ldots, E_n$ astfel ca:

1. $E_1 A$ să aibă prima coloană identică cu prima coloană a lui I_n.
2. $E_2 E_1 A$ să aibă primele două coloane identice cu cele ale lui I_n, s.a.m.d..

3. Procedeul se repetă recursiv până când se obţine matricea I_n, adică relaţia: $E_n ... E_2 E_1 A = I_n$.

4. Dacă A este inversabilă, atunci $E_n ... E_2 E_1 = A^{-1}$, deci inversa ei se află făcând produsul matricelor astfel construite, considerate în ordinea inversă obţinerii lor.

Fie A matricea pătratică a cărei inversă o vom afla:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Pasul 1

Construim

$$E_1 = \begin{bmatrix} \frac{1}{a_{11}} & 0 & \dots & 0 \\ -\frac{a_{21}}{a_{11}} & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ -\frac{a_{n1}}{a_{11}} & 0 & \dots & 1 \end{bmatrix}$$

Apoi calculăm $A_1 = E_1 A$, matrice ale cărei elemente le vom nota cu a_{ij}^1

Deci:

$$A_1 = E_1 A = \begin{bmatrix} 1 & a_{12}^1 & \dots & a_{1n}^1 \\ 0 & a_{22}^1 & \dots & a_{2n}^1 \\ \dots & \dots & \dots & \dots \\ 0 & a_{n2}^1 & \dots & a_{nn}^1 \end{bmatrix}$$

Pasul 2

Construim acum matricea elementară E_2.

$$E_2 = \begin{bmatrix} 1 & -\dfrac{a_{12}^1}{a_{22}^1} & \dots & 0 \\ 0 & \dfrac{1}{a_{22}^1} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & -\dfrac{a_{n2}^1}{a_{22}^1} & \dots & 1 \end{bmatrix}$$

Calculăm $A_2 = E_2 A_1 = E_2 E_1 A$, matrice ale cărei elemente le vom nota cu a_{ij}^2

$$A_2 = E_2 E_1 A = \begin{bmatrix} 1 & 0 & \dots & a_{1n}^2 \\ 0 & 1 & \dots & a_{2n}^2 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{nn}^2 \end{bmatrix}$$

Continuând în mod analog se construiesc matricele $E_3, ..., E_n$ şi apoi cu relaţia (2) matricea A^{-1}.

Observaţia III.4. Relaţiile precedente arată că la pasul i trebuie să avem $a_{ii}^{i-1} \neq 0$ pentru ca algoritmul să poată fi aplicat, unde $a_{11}^{0} = a_{11}$.

Exemplul III.3 Să se determine inversa matricei *A*, unde

$$A = \begin{pmatrix} 3 & -4 \\ 1 & 2 \end{pmatrix}.$$

Soluţie:

Determinam $E_1 = \begin{pmatrix} 1/3 & 0 \\ -1/3 & 1 \end{pmatrix}$ şi calculăm $A_1 = E_1 \cdot A$:

$$E_1 \cdot A = \begin{pmatrix} 1/3 & 0 \\ -1/3 & 1 \end{pmatrix}\begin{pmatrix} 3 & -4 \\ 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & -4/3 \\ 0 & 10/3 \end{pmatrix} = A_1$$

Determinăm $E_2 = \begin{pmatrix} 1 & 4/10 \\ 0 & 3/10 \end{pmatrix}$ şi calculăm $E_2 \cdot A_1$

$$E_2 \cdot A_1 = \begin{pmatrix} 1 & 4/10 \\ 0 & 3/10 \end{pmatrix} \cdot \begin{pmatrix} 1 & -4/3 \\ 0 & 10/3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

În concluzie inversa lui *A* este dată de:

$$A^{-1} = E_2 \cdot E_1 = \begin{pmatrix} 1 & 4/10 \\ 0 & 3/10 \end{pmatrix} \begin{pmatrix} 1/3 & 0 \\ -1/3 & 1 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 1/5 & 2/5 \\ -1/10 & 3/10 \end{pmatrix}$$

Procedeul prezentat mai sus nu este singurul procedeu numeric ce poate fi utilizat pentru calculul inversei unei matrici când aceasta există.

Alte două procedee larg utilizate sunt:

-metoda lui Gauss-Jordan

-metoda iterativă.

Metoda Gauss-Jordan se aplică tabloului format din matricea *A*, a cărei inversă dorim să o aflăm, aşezată în partea stângă, urmată de matricea unitate corespunzătoare ordinului lui *A*.

În tabloul final se obţine în stânga matricea unitate iar în dreapta inverse matricei *A*.

În cazul aflării inversei unei matrici pivoţii se aleg pe diagonala principală. Dacă la un anumit pas elementul ce urmează a fi ales pivot este nul, şi totuşi există inversa lui *A*, se permută două linii (coloane) astfel încât să se obţină un pivot nenul şi se continuă algoritmul. Matricea inversă se obţine prin permutarea coloanelor (liniilor) corespunzătoare în matricea din partea dreaptă a ultimului tablou.

Exemplul III.4. Să se afle cu ajutorul metodei Gauss-Jordan inversa matricei:

$$A = \begin{pmatrix} 1 & -3 \\ 1 & 7 \end{pmatrix}$$

Construim primul tablou ce conţine în dreapta matricea unitate de ordinul trei şi în partea stângă matricea A.

[1]	-3	1	0
1	7	0	1
1	-3	1	0
0	[10]	-1	1
1	0	7/10	3/10
0	1	-1/10	1/10

Din ultimul tablou deducem inversa lui A, A^{-1}.

$$A^{-1} = \begin{pmatrix} 7/10 & 3/10 \\ -1/10 & 1/10 \end{pmatrix}.$$

Exemplul III.5. Să se afle cu ajutorul metodei Gauss-Jordan inversa matricei:

$$A = \begin{pmatrix} 1 & 2 & 0 \\ -1 & 3 & 1 \\ 2 & 0 & 1 \end{pmatrix}$$

Soluţie:

1	2	0	1	0	0
-1	3	1	0	1	0
2	0	1	0	0	1
1	2	0	1	0	0
0	5	1	-2	0	1
0	-4	1	-2	0	1
1	0	-2/5	3/5	-2/5	0
0	1	1/5	1/5	1/5	0
0	0	9/5	-6/5	4/5	1
1	0	0	1/3	-2/9	2/9
0	1	0	1/3	1/9	-1/9
0	0	1	-6/9	4/9	5/9

Din ultimul tablou deducem:

$$A^{-1} = \begin{pmatrix} \frac{1}{3} & \frac{-2}{9} & \frac{2}{9} \\ \frac{1}{3} & \frac{1}{9} & \frac{-1}{9} \\ \frac{-2}{3} & \frac{4}{9} & \frac{5}{9} \end{pmatrix}.$$

/*Program Inversa -varianta G-J cu pivotare partiala*/

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
float t,m;
int i,j,k,l,n;
float s, A[10][20], u[10][10];
void main()
{
printf ("Intoduceti nr. de linii si de coloane ale matricei");
scanf("%d",&n);
printf("Dati matricea\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
                printf("A[%d][%d]: ",i,j);
                scanf("%f",&A[i][j]);
        }
for (i=1;i<=n;i++)
        for(j=n+1;j<=n+n;j++)
        {
                if (j==i+n) A[i][j]=1;
                else A[i][j]=0;
        }
for(k=1;k<=n;k++)
        {
                l=k;
                for (i=k;i<=n;i++)
                        {u[i][k]=sqrt(A[i][k]*A[i][k]);
                        if (u[i][k]>u[k][k])
                        l=i;}
                if(l!=k)
                for(j=1;j<=n+n;j++)
                {
                        t=A[k][j];A[k][j]=A[l][j];A[l][j]=t;
                }
                for(i=1;i<=n;i++)
                        {
```

```
                    m=A[i][k]/A[k][k];
                    A[i][k]=0;
                    for(j=k+1;j<=n+n;j++)
                    A[i][j]=A[i][j]-m*A[k][j];
                    }
            A[k][k]=1;
    }
printf("Matricea inversa este: \n");
for (i=1;i<=n;i++)
    {
        for (j=n+1;j<=n+n;j++)
        printf("A[%d][%d]=%f",i,j-n,A[i][j]);
        printf("\n");
    }
    getch();
    }
```

Exemplul III.6. Să se determine, folosind metoda transformărilor elementare, inversa matricei $A=\begin{pmatrix} 4 & -1 & 3 \\ 2 & 0 & -1 \\ 2 & 1 & -2 \end{pmatrix}$.

Soluţie:

Pasul 1: Determinăm $E_1=\begin{pmatrix} 1/4 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix}$

Calculăm

$$E_1 \cdot A=\begin{pmatrix} 1/4 & 0 & 0 \\ -1/2 & 1 & 0 \\ -1/2 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & -1 & 3 \\ 2 & 0 & -1 \\ 2 & 1 & -2 \end{pmatrix} = \begin{pmatrix} 1 & -1/4 & 3/4 \\ 0 & 1/2 & -5/2 \\ 0 & 3/2 & -7/2 \end{pmatrix} \overset{not}{=} A_1$$

Pasul 2: Determinăm $E_2 = \begin{pmatrix} 1 & 1/2 & 0 \\ 0 & 2 & 0 \\ 0 & -3 & 1 \end{pmatrix}$

Calculăm

$$E_2 \cdot A_1 = \begin{pmatrix} 1 & 1/2 & 0 \\ 0 & 2 & 0 \\ 0 & -3 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & -1/4 & 3/4 \\ 0 & 1/2 & -5/2 \\ 0 & 3/2 & -7/2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -1/2 \\ 0 & 1 & -5 \\ 0 & 0 & 4 \end{pmatrix} \overset{not}{=} A_2$$

Pasul 3: Determinăm $E_3 = \begin{pmatrix} 1 & 0 & 1/8 \\ 0 & 1 & 5/4 \\ 0 & 0 & 1/4 \end{pmatrix} \Rightarrow$

$$E_3 \cdot A^2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \Rightarrow A^{-1} = E_3 E_2 E_1 .$$

Efectuând calculele obţinem:

$$A^{-1} = \begin{pmatrix} 1/8 & 1/8 & 1/8 \\ 1/4 & -7/4 & 5/4 \\ 1/4 & -3/4 & 1/4 \end{pmatrix}.$$

IV. METODA APROXIMAŢIILOR SUCCESIVE ÎN REZOLVAREA SISTEMELOR LINIARE

IV.1. METODA APROXIMAŢIILOR SUCCESSIVE

Definiţia IV.1. Spunem că (X,d) este un spaţiu metric complet dacă orice şir Cauchy cu elemente din X este şir convergent.

Definiţia IV.2. Să considerăm un spaţiu metric (X,d) şi o aplicaţie $f : X \to X$. Vom spune că $x^* \in X$ este punct fix pentru f dacă $f(x^*) = x^*$.

Definiţia IV.3. Aplicaţia f se numeşte contracţie dacă există $a \in R, 0 < a < 1$, astfel încât $d(f(x), f(y)) \le ad(x,y)$ pentru oricare ar fi $x, y \in X$. a se numeşte constanta contracţiei.

Teorema IV.1. (Teorema de punct fix a lui Banach) Fie (X,d) un spaţiu metric complet şi $f : X \to X$ o contracţie. Atunci f are un unic punct fix $x^* \in X$. El este limita şirului

$$x_{n+1} = f(x_n), n = 0,1,....$$

unde $x_0 \in X$ este arbitrar.

Din demonstraţia teoremei se poate deduce că:

$d(x_n, x_{n+1}) \le a^n d(x_0, x_1), n \ge 0$

$$d(x_n, x_{n+p}) \le \frac{a^n}{1-a} d(x_0, x_1), n, p \in N$$

Relaţia precedentă spune că şirul (x_n) este şir Cauchy. Întrucât (X,d) este spaţiu metric complet, (x_n) va fi şir convergent. Notăm cu $x^* \in X$ limita sa. Avem:

$$\lim_{n\to\infty} d(x_n, x^*) = 0 \text{ şi } x^* = f(x^*)$$

Observaţia IV.1. Elementele x_0, x_1, x_2,… se numesc aproximaţiile succesive ale lui x^*. Aproximaţia iniţială x_0 poate fi luată arbitrar în X.

Observaţia IV.2.

Se poate deduce că:

1. $d(x_n, x^*) \le \frac{a^n}{1-a} d(x_0, x_1), n \ge 0$

Această inegalitate ne arată cât de mare poate fi eroarea pe care o obţinem când aproximăm pe x^* cu x_n.

2. $d(x_n, x^*) \le \frac{a}{1-a} d(x_{n-1}, x_n), n \ge 1$

IV.2. METODE ITERATIVE PENTRU REZOLVAREA SISTEMELOR DE ECUAŢII LINIARE

Metodele iterative sunt mai simple însă prezintă dezavantajul că nu se poate stabili de la început numărul de paşi necesari rezolvării. Astfel sunt costisitoare, mai ales în ceea ce priveşte timpul de calul. Se folosesc în general pentru

sisteme de dimensiuni mari şi sunt de asemenea potrivite rezolvării sistemelor ce prezintă mulţi coeficienţi nuli.

Metodele iterative constau în construirea unui şir $\left(x^{(k)} \in R^n\right), k = 0,1,.....$, convergent către soluţia exactă x a sistemului: $Ax = b$, unde $A \in R^{u \times n}, b \in R^n$.

Oprirea procesului iterativ este influenţată de eroarea admisibilă dată, notată cu ε sau cu ε_{adm}.

Observaţia IV.3. Pentru rezolvarea sistemului $Ax = b$ definim $f : R^n \to R^n$ astfel: $f(x) = y$, unde componentele lui y sunt date de:

$$y_1 = \frac{1}{a_{11}}\left(b_1 - \sum_{j=2}^{n} a_{1j}x_j\right)$$

$$y_2 = \frac{1}{a_{22}}\left(b_2 - \sum_{j=3}^{n} a_{2j}x_j - a_{21}y_1\right)$$

$$\vdots$$

$$y_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}y_j - \sum_{j=i+1}^{n} a_{ij}x_j\right)$$

$$\vdots$$

$$y_n = \frac{1}{a_{nn}}\left(b_n - \sum_{j=1}^{n-1} a_{nj}y_j\right)$$

Sunt adevărate următoarele enunţuri:

1. Această aplicaţie se poate defini doar atunci când diagonala principală a matricei A nu conţine elemente nule $(a_{ii} \neq 0)$.

2. Aplicaţia $d : R^n \times R^n \to R \quad d(x,y) = \max_i |x_i - y_i|$ este o metrică pe R^n iar (R^n, d) reprezintă un spaţiu metric complet.

Teorema IV.2. Dacă $A \in M_n(R)$ este astfel încât $(*)\ |a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \ i = \overline{1,n}$ (diagonal dominantă pe linii), atunci aplicaţia *f* este o contracţie de coeficient *q* dat derelaţia:

$$q = \max_{i=1,n} \frac{\sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|}{|a_{ii}|}$$

De aici obţinem un algoritm de soluţionare a unui sistem algebric caracterizat de o matrice ce satisface cerinţele teoremei.

Algoritm. Alegem $x^{(0)} \in R^n$ şi apoi construim şirul de aproximaţii succesive (iteraţii successive) folosind relaţia:

$x^{(n+1)} = f(x^{(n)}) \quad n = 0,1,2,...$

Atunci $\Rightarrow x^* = \lim_{n \to \infty} x^{(n)}$ este singurul punct fix al lui *f*.

Mai mult $x^* = f(x^*) \Rightarrow x^*$ este soluţia sistemului $Ax = b$.

Există în principal două metode iterative ce se folosesc la rezolvarea sistemelor liniare: metoda Jacobi şi metoda Gauss-Seidel.

I. Metoda Jacobi (metoda iteraţiilor simultane)

Această metodă iterativă constă în construcţia şirului $x^{(k)}, \quad k = 0,1,...$ astfel:

$x^{(0)}$ se alege arbitrar,

$$x_1^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)} \right), \quad i = \overline{1,n}$$

Observaţia IV.4. Se observă că la o iteraţie în formule de calcul din membrul drept se folosesc toate componentele iteraţiei precedente.

II. Metoda Seidel-Gauss (metoda iteraţiilor succesive) Ea constă în construcţia şirului $x^{(k)}, k = 0,1,...$ după modelul de construcţie a funcţiei f din paragrafele anterioare (astfel $y \to x^{(k+1)}$).

Observaţia IV.5. La o iteraţie, în formule, se înlocuiesc valorile obținute la iterația precedentă cu valorile de la iterația curentă, dacă acestea sunt calculate.

Există şi cazuri în care condiţia $(*)$ nu este îndeplinită şi totuşi algoritmul Seidel-Gauss converge.

Metoda Jacobi este mai lent convergentă decât metoda Seidel–Gauss în aceleaşi condiţii iniţiale date $x^{(0)}$ şi ε.

Observaţia IV.6.

Condiția (*) din teorema IV.2 poate fi înlocuită cu condiția:

A să fie diagonal dominantă pe coloane, adică $|a_{jj}| > \sum_{\substack{i=1 \\ i \neq j}}^{n} |a_{ij}|, \; j = \overline{1,n}$.

Observaţia IV.7.

În general iteraţiile se opresc atunci când, pentru un ε dat, adică pentru o eroare admisibilă dată, este îndeplinită condiţia:

$$\left|x_i^{(k+1)} - x_i^{(k)}\right| < \varepsilon \quad (\forall) i = \overline{1,n}$$

Eroarea admisibilă dată reflectă gradul de precizie dorit.

Observaţia IV.8. Algoritmul se poate opri şi dacă nu s-a ajuns la gradul de precizie dorit dar s-au efectuat prea multe iteraţii, adică dacă numărul de iteraţii atinge o valoare precizată, denumită număr maxim admis de iteraţii. Uneori se foloseşte chiar o combinaţie de condiţii în care apar atât eroarea adisibilă cât şi numărul maxim de iteraţii.

În continuare descriem pe larg cele două metode iterative amintite.

I. Metoda Jacobi

Metoda Jacobi este cea mai simplă dintre metodele iterative.

Considerăm un sistem cu n ecuaţii şi n ce satisface condiția de convergență a metodei.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \end{cases}$$

Rearanjăm sistemul scriindu-l sub o nouă formă, denumită formă iterativă, astfel:

$$\begin{cases} x_1 = \dfrac{b_1}{a_{11}} - \left(\dfrac{a_{12}}{a_{11}} x_2 + \dfrac{a_{13}}{a_{11}} x_3 + \cdots + \dfrac{a_{1n}}{a_{11}} x_n \right) \\ x_2 = \dfrac{b_2}{a_{22}} - \left(\dfrac{a_{21}}{a_{22}} x_1 + \dfrac{a_{23}}{a_{22}} x_3 + \cdots + \dfrac{a_{2n}}{a_{22}} x_n \right) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n = \dfrac{b_n}{a_{nn}} - \left(\dfrac{a_{n1}}{a_{nn}} x_1 + \dfrac{a_{n2}}{a_{nn}} x_2 + \cdots + \dfrac{a_{n,n-1}}{a_{nn}} x_{n-1} \right) \end{cases}$$

Această formă ne permite ca la o nouă iterație să evaluăm noi valori ale soluției, folosind în calcul, în membrii drepți, valorile determinate la iterația anterioară. Notând cu x_i^0 componentele vectorului soluție inițială calculăm la pasul *k*, $k \geq 1$ valorile soluțiilor astfel:

$$\begin{cases} x_1^k = \dfrac{b_1}{a_{11}} - \left(\dfrac{a_{12}}{a_{11}} x_2^{k-1} + \dfrac{a_{13}}{a_{11}} x_3^{k-1} + \cdots + \dfrac{a_{1n}}{a_{11}} x_n^{k-1} \right) \\ x_2^k = \dfrac{b_2}{a_{22}} - \left(\dfrac{a_{21}}{a_{22}} x_1^{k-1} + \dfrac{a_{23}}{a_{22}} x_3^{k-1} + \cdots + \dfrac{a_{2n}}{a_{22}} x_n^{k-1} \right) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_n^k = \dfrac{b_n}{a_{nn}} - \left(\dfrac{a_{n1}}{a_{nn}} x_1^{k-1} + \dfrac{a_{n2}}{a_{nn}} x_2^{k-1} + \cdots + \dfrac{a_{n,n-1}}{a_{nn}} x_{n-1}^{k-1} \right) \end{cases}$$

Şirul de valori găsite converge spre soluţia sistemului. Se cosideră de obicei $x_i^0 = 0$.

Algoritmul de rezolvare a sistemelor de ecuaţii liniare prin metoda Jacobi

Date de intrare: rangul n, matricea coeficienţilor A, vectorul termenilor liberi b, eroarea admisibilă ε_{adm} şi numărul maxim de iteraţii admise $N\max$;

Date de ieşire: soluţia aproximativă

1. Se introduc datele de intrare;
2. Se stabileşte aproximaţia iniţială, se alege ca fiind cea nulă:
$$x_i \leftarrow 0, i = \overline{1,n};$$
3. Se iniţializează procesului iterativ: $k \leftarrow 1$;
4. Se execută calculele:
   a. Se calulează noua aproximaţie în vectorul y:
   $$y_i \leftarrow \frac{b_i}{a_{ii}} - \sum_{\substack{j=1 \\ j \neq i}}^{n} \frac{a_{ij}}{a_{ii}} x_j \ , i = \overline{1,n}$$
   b. Se calculează eroarea absolută maximă în iteraţia curentă:
   $D = \max_i |y_i - x_i|$
   c. Se consemnează execuţia iteraţiei $k \leftarrow k+1$
5. Atâta vreme cât $D >= \varepsilon_{adm}$ şi $k \le N\max$
6. Daca $k \le N\max$ se afişează soluţia aproximativă
   $y_i \ , i = \overline{1,n}$ şi numărul de iteraţii efectuate k;
7. Altfel se afiseaza mesajul "S-a depăşit numarul maxim de iteratii" .
8. Stop.

II.Metoda Gauss-Seidel (G-S)

Fie sistemul liniar de n ecuaţii cu n necunoscute scris sub formă iterativă:

$$\begin{cases} x_1 = \frac{b_1}{a_{11}} - (\frac{a_{12}}{a_{11}} x_2 + \frac{a_{13}}{a_{11}} x_3 + ... + \frac{a_{1n}}{a_{11}} x_n) \\ x_2 = \frac{b_2}{a_{22}} - (\frac{a_{21}}{a_{22}} x_1 + \frac{a_{23}}{a_{22}} x_3 + ... + \frac{a_{2n}}{a_{22}} x_n) \\ \vdots \\ x_n = \frac{b_n}{a_{nn}} - (\frac{a_{n1}}{a_{nn}} x_1 + \frac{a_{n2}}{a_{nn}} x_2 + ... + \frac{a_{2n-1}}{a_{nn}} x_{n-1}) \end{cases}$$

Amintim că starea inițială a soluției este definită prin valorile x_i^0 , i = 1,n.

Determinăm valoarea lui x_1 la pasul 1 (notată cu x_1^1), ca la metoda Jacobi:

$$x_1^1 = \frac{b_1}{a_{11}} - (\frac{a_{12}}{a_{11}} x_2^0 + \frac{a_{13}}{a_{11}} x_3^0 + ... + \frac{a_{1n}}{a_{11}} x_n^0)$$

În evaluarea lui x_2^1 ținem cont că pentru x_1 cunoaștem două valori: x_1^0 şi x_1^1 (calculată anterior). Cum x_1^1 reprezintă o valoare mai recentă o vom prefera pentru accelerarea convergenţei. Deci:

$$x_2^1 = \frac{b_2}{a_{22}} - (\frac{a_{21}}{a_{22}} x_1^1 + \frac{a_{23}}{a_{22}} x_3^0 + ... + \frac{a_{2n}}{a_{22}} x_n^0)$$, etc.

Pentru pasul *k* putem scrie:

$$x_1^k = \frac{b_1}{a_{11}} - (\frac{a_{12}}{a_{11}} x_2^{k-1} + \frac{a_{13}}{a_{11}} x_3^{k-1} + ... + \frac{a_{1n}}{a_{11}} x_n^{k-1})$$

$$\vdots$$

$$x_i^k = \frac{b_i}{a_{ii}} - (\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^k + \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j^{k-1})$$

$$\vdots$$

$$x_n^k = \frac{b_n}{a_{nn}} - (\frac{a_{n1}}{a_{nn}} x_1^k + \frac{a_{n2}}{a_{nn}} x_2^k + ... + \frac{a_{nn-1}}{a_{nn}} x_{n-1}^k)$$

Observaţia IV.9. Algoritmul Gauss-Seidel are deci la bază algoritmul Jacobi, dar se distinge de acesta prin folosirea celor mai recente valori ale necunoscutelor. Astfel se obţine pe ansamblu o accelerare a procesului de iteraţie.

Algoritmul de rezolvare a sistemelor de ecuatii liniare prin metoda Gauss-Seidel

Date de intrare: rangul n, matricea coeficienţilor A, vectorul termenilor liberi b, eroarea admisibilă ε_{adm} şi numărul maxim de iteraţii admise $N\max$;

Date de ieşire: soluţia aproximativă

1. Se introduc datele de intrare;
2. Se alege aproximaţia iniţială, aceasta se alege a fi identic nulă:
$$x_i \leftarrow 0, i = \overline{1,n}$$
3. Se iniţializează procesului iterativ: $k \leftarrow 0$;
4. Se iniţializează eroarea absolută maximă în iteraţia curentă cu o valoare superioară lui ε_{adm}: $D \leftarrow \varepsilon_{adm} + 1$;

5. Se trece la o nouă iteraţie: $k \leftarrow k+1$;
6. Se calculează noua aproximaţie în vectorul *y*:

$$y_i = \frac{b_i}{a_{ii}} - (\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} y_j + \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} x_j)$$

7. Se calculează eroarea absolută maximă în iteraţia curentă:

$$D = \max_i |y_i - x_i|$$

8. Daca $D \le \varepsilon_{adm}$ (metoda converge) se trece la pasul 11, altfel (dacă $D > \varepsilon_{adm}$), se compară k şi $N\max$; dacă $k \ge N\max$ (metoda nu converge), se afiseaza mesajul "Depasire nr. maxim iteratii" şi se trece la pasul 12; altfel se trece la pasul următor;
9. Se actualizează vectorul aproximaţiilor din ultima iteraţie:

$$x_i \leftarrow y_i \ , i = \overline{1,n}$$

10. Se revine la pasul 5;
11. Se afişează soluţia aproximativă $y_i \ , i = \overline{1,n}$ şi numărul de iteraţii efectuate *k*;
12. Stop.

Exemplul IV.1. Să se rezolve următorul sistem de ecuaţii prin metoda Jacobi :

$$\begin{cases} -x_1 + 10x_2 + x_3 = 20 \\ 2x_1 - x_2 + 10x_3 = 22 \\ 10x_1 - x_2 + 2x_3 = 12 \end{cases}$$

Se va accepta o soluţie pentru care $\max_i \left| x_i^k - x_i^{k-1} \right| < \varepsilon_{adm}$, unde $\varepsilon_{adm} = 0{,}002$. Se va lucra cu patru zecimale.

Soluţie:

Pentru ca sistemul de ecuaţii trebuie să fie astfel ca elementele diagonale din matricea coeficienţilor să fie dominante în valoare absolută, schimbăm poziţia ecuaţiilor astfel: ecuaţia 3 devine ecuaţia 1, ecuaţia 1 devine ecuaţia 2 iar ecuaţia 2 devine ecuaţia 3 :

$$\begin{cases} 10x_1 - x_2 + 2x_3 = 12 \\ -x_1 + 10x_2 + x_3 = 20 \\ 2x_1 - x_2 + 10x_3 = 22 \end{cases}$$

Rescriem sistemul în forma sa iterativă :

$$\begin{cases} x_1 = 1{,}2 + 0{,}1x_2 - 0{,}2x_2 \\ x_2 = 2 + 0{,}1x_1 - 0{,}1x_3 \\ x_3 = 2{,}2 - 0{,}2x_1 + 0{,}1x_2 \end{cases}$$

- Pasul 1.

Considerăm ca soluţie iniţială : $x_1^0 = x_2^0 = x_3^0 = 0.$

$$\begin{cases} x_1^1 = 1{,}2 + 0{,}1 \cdot 0 - 0{,}2 \cdot 0 = 1{,}2 \quad ; \quad \left|x_1^1 - x_1^0\right| = 1{,}2 \\ x_2^1 = 2 + 0{,}1 \cdot 0 - 0{,}1 \cdot 0 = 2 \quad ; \quad \left|x_2^1 - x_2^0\right| = 2 \\ x_3^1 = 2{,}2 - 0.2 \cdot 0 + 0{,}2 \cdot 0 = 2{,}2 \quad ; \quad \left|x_3^1 - x_3^0\right| = 2{,}2 \end{cases}$$

$$\max_i \left|x_i^1 - x_i^0\right| = 2{,}2 > \varepsilon_{adm}$$

- Pasul 2

Valorile determinate în pasul precedent $x_1^1; x_2^1; x_s^1$ se introduc în membrul drept al sistemului:

$$\begin{cases} x_1^2 = 1{,}2 + 0{,}1 \cdot 2 - 0{,}2 \cdot 2{,}2 = 0{,}96 \quad ; \quad \left|x_1^2 - x_1^1\right| = 0{,}24 \\ x_2^2 = 2 + 0{,}1 \cdot 1{,}2 - 0{,}1 \cdot 2{,}2 = 1{,}9 \quad ; \quad \left|x_2^2 - x_2^1\right| = 0{,}1 \\ x_3^2 = 2{,}2 - 0.2 \cdot 1{,}2 + 0{,}2 \cdot 2 = 2{,}16 \quad ; \quad \left|x_3^2 - x_3^1\right| = 0{,}16 \end{cases}$$

$$\max_i \left|x_i^2 - x_i^1\right| = 0{,}24 > \varepsilon_{adm}$$

- Pasul 3

Valorile determinate la pasul precedent $x_1^2; x_2^2; x_3^2$ se introduc în membrul drept al sistenului și se obține:

$$\begin{cases} x_1^3 = 0{,}958 \quad ; \quad \left|x_1^3 - x_1^2\right| = 0{,}002 \\ x_2^3 = 1{,}88 \quad ; \quad \left|x_2^3 - x_2^2\right| = 0{,}02 \\ x_3^3 = 2{,}198 \quad ; \quad \left|x_3^3 - x_3^2\right| = 0{,}038 \end{cases}$$

$$\max_i \left|x_1^3 - x_i^2\right| = 0{,}038 > \varepsilon_{adm}$$

- Pasul 4

Valorile determinate la pasul precedent $x_1^3; x_2^3; x_3^3$ se introduc în membrul drept al sistemului de ecuaţii și se obține:

$$\begin{cases} x_1^4 = 0{,}9484 \quad ; \quad \left|x_1^4 - x_1^3\right| = 0{,}0096 \\ x_2^4 = 1{,}8760 \quad ; \quad \left|x_2^4 - x_2^3\right| = 0{,}004 \\ x_3^4 = 2{,}1964 \quad ; \quad \left|x_3^4 - x_3^3\right| = 0{,}0016 \\ \max_i \left|x_i^4 - x_i^3\right| = 0{,}0096 > \varepsilon_{adm} \end{cases}$$

- Pasul 5

Valorile determinate la pasul precedent $x_1^4; x_2^4; x_3^4$ se introduc în membrul drept al sistemului de ecuaţii și se obține:

$$\begin{cases} x_1^5 = 0{,}9483 \quad ; \quad \left|x_1^5 - x_1^4\right| = 0{,}0001 \\ x_2^5 = 1{,}8752 \quad ; \quad \left|x_2^5 - x_2^4\right| = 0{,}0008 \\ x_3^5 = 2{,}1979 \quad ; \quad \left|x_3^5 - x_3^4\right| = 0{,}0015 \end{cases}$$

$$\max_i \left|x_i^5 - x_i^4\right| = 0{,}0015 < 0{,}002 = \varepsilon_{adm}$$

Deoarece s-a obţinut precizia de calcul solicitată, procesul de iteraţie se opreşte.

Soluţia sistemului este :

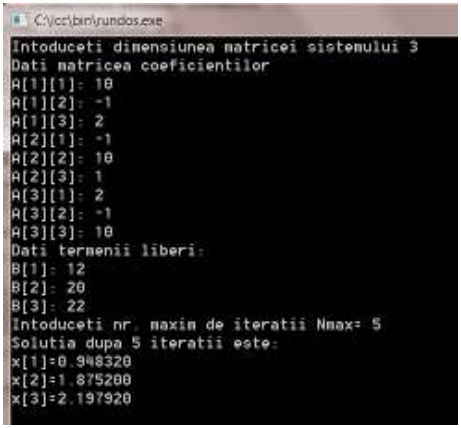
$$\begin{cases} x_1 = x_1^5 = 0{,}9483 \\ x_2 = x_2^5 = 1{,}8752 \\ x_3 = x_3^5 = 2{,}1979 \end{cases}$$

Un program în C, care furnizează soluția aproximativă a unui sistem liniar, obținută după un număr dat de iteraţii, prin metoda Jacobi este prezentat în continuare.

```
/*Program - Metoda JACOBI-nr maxim iteratii*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
float s, A[30][30], x[30],B[30], y[30];
int n,k,j,i,Nmax;
void main()
{
printf ("Intoduceti dimensiunea matricei sistemului ");
scanf("%d",&n);
printf("Dati matricea coeficientilor\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
printf("A[%d][%d]: ",i,j);
```

```
scanf("%f",&A[i][j]);
}
printf("Dati termenii liberi:\n");
for (i=1;i<=n;i++)
        {
        printf("B[%d]: ",i);
        scanf("%f",&B[i]);
        }
        printf ("Intoduceti nr. maxim de iteratii Nmax= ");
scanf("%d",&Nmax);
for(i=1;i<=n;i++)
        x[i]=0;
for(k=1;k<=Nmax;k++)
{
for(i=1;i<=n;i++)
{
s=0;
for(j=1;j<=n;j++)
s=s+A[i][j] * x[j];
s=s- A[i][i] * x[i];
y[i]= (B[i]-s)/A[i][i];
}
for(i=1;i<=n;i++)
x[i]=y[i];
}
printf("Solutia dupa %d iteratii este:\n", Nmax);
for(i=1;i<=n;i++)
printf("x[%d]=%f \n",i,x[i]);
          getch();
}
```

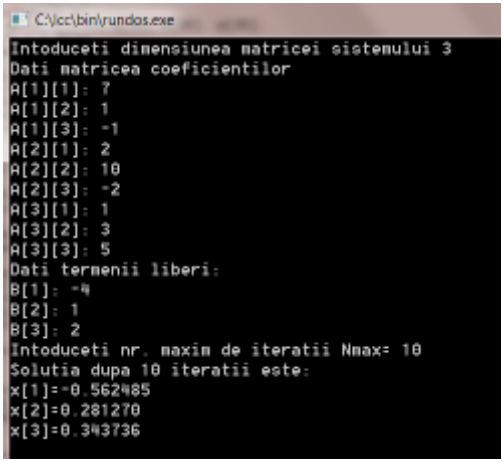
Prin rularea programului precedent, obținem după 5 iterații, pentru problema anterioară, rezultatele următoare (abstracție făcând de erorile de rotunjire, putem spune că sunt identice cu cele anterioare).



Exemplul IV.2. Să se determine, folosind metoda Jacobi, soluția aproximativă, după 10 iterații, pentru sistemul următor:

$$\begin{cases} 7x_1 + x_2 - x_3 = -4 \\ 2x_1 + 10x_2 - 2x_3 = 1 \\ x_1 + 3x_2 + 5x_3 = 2 \end{cases}$$

Prin rularea programului sursă ce folosește numărul maxim de iterații obținem, după 10 iterații pentru problema precedentă rezultatele următoare:

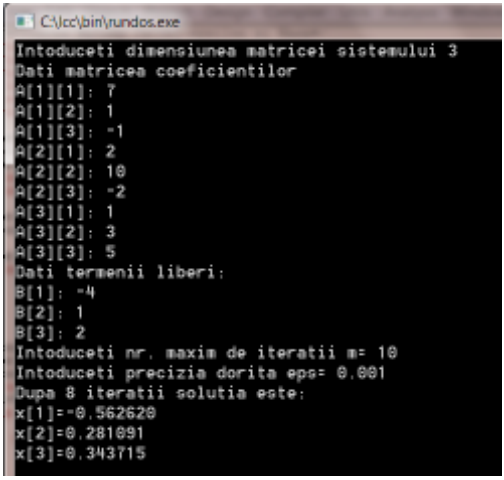


Următorul cod sursă (în C) rezolvă un sistem de ecuaţii prin algoritmul metodei Jacobi dar foloseşte pentru oprire atât un număr maxim de iteraţii admis, cât şi o eroare admisibilă dată.

```
/*Program Metoda JACOBI-epsilon si Nmax*/
#include<stdio.h>
#include<conio.h>
#include<math.h>
float s, max, eps,A[30][30],x[30],B[30],y[30];
int n,Nmax,j,i,k;
void main()
{
printf ("Intoduceti dimensiunea matricei sistemului ");
scanf("%d",&n);
printf("Dati matricea coeficientilor\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
printf("A[%d][%d]: ",i,j);
scanf("%f",&A[i][j]);
}
printf("Dati termenii liberi:\n");
for (i=1;i<=n;i++)
        {
        printf("B[%d]: ",i);
        scanf("%f",&B[i]);
        }
        printf ("Intoduceti nr. maxim de iteratii m= ");
scanf("%d",&Nmax);
printf ("Intoduceti precizia dorita eps= ");
scanf("%f",&eps);
for(i=1;i<=n;i++) x[i]=0;
k=0;
```

```
do
   {
  for(i=1;i<=n;i++)
            {
              s=0;
          for(j=1;j<=n;j++)
            s=s+A[i][j] * x[j];
          s=s- A[i][i] * x[i];
          y[i]= (B[i]-s)/A[i][i];
            }
 max=fabs(y[1]-x[1]);
 for(i=2;i<=n;i++)
        if (max<fabs(y[i]-x[i]))
          max=fabs((y[i]-x[i]));
 for(i=1;i<=n;i++) x[i]=y[i];
 k=k+1;
   }
while (max>=eps && k<=Nmax);
        if (k<=Nmax)
                    {
          printf("Dupa %d iteratii solutia este:\n",k);
                    for(i=1;i<=n;i++)
                    printf("x[%d]=%f \n",i,x[i]);
                     }
        else printf("S-a depasit numarul maxim de iteratii");
          getch();
}
```

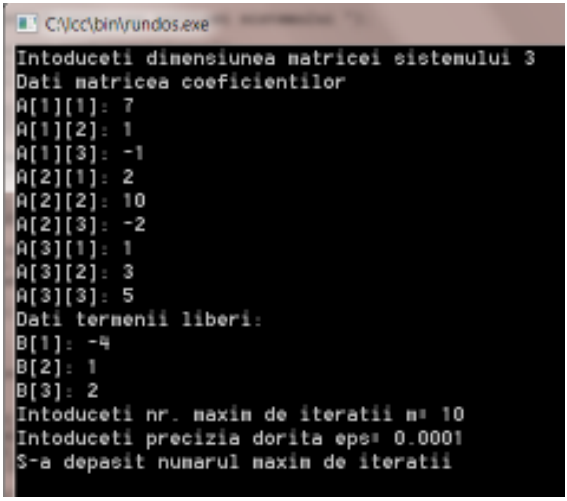
Exemplul IV.3. Să se rezolve sistemul de ecuaţii din exemplul anterior utilizând sursa de mai sus, considerând eroarea admisibilă 10^{-3} .



Se observă că soluția aproximativă a sistemului s-a obținut după un număr de 8 iterații, în limitele preciziei dorite, iar numărul maxim de iterații admis nu a fost depășit.

Vom căuta soluția acestui sistem mărind precizia, adică micșorând eroarea admisibilă dată de 10 ori (adcând-o la valoarea 10^{-4}), dar menținând același timp de execuție, adică același număr maxim de iterații admis.

Se observă din rezultatele următoare că nu putem atinge precizia dorită folosind același număr maxim de iterații admis.



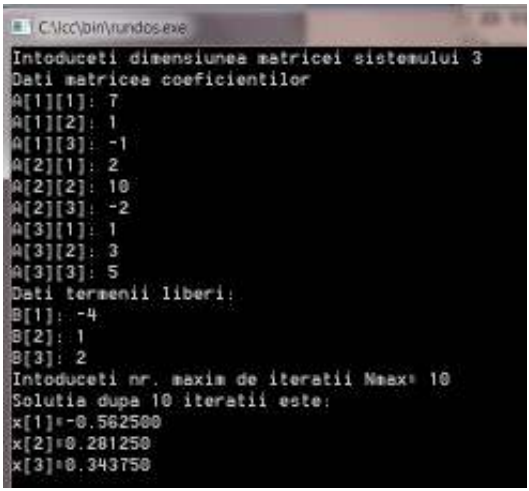
Pentru rezolvarea sistemelor de ecuatii liniare prin metoda Seidel Gauss sunt concepute în continuare două programe simple, în limbajul C. Primul se bazează pe afişarea soluţiei numerice ce se obţine după un număr maxim de iteraţii admis, Nmax, ce se cere a fi introdus de către utilizator.

/*Program - Metoda Seidel-Gauss-nr maxim iteratii*/

```
#include<stdio.h>
#include<conio.h>
float s, A[30][30],x[30],B[30], y[30];
int n,k,j,i,Nmax;
void main()
{
printf ("Intoduceti dimensiunea matricei sistemului ");
scanf("%d",&n);
printf("Dati matricea coeficientilor\n");
for(i=1;i<=n;i++)
```

```
for(j=1;j<=n;j++)
{
printf("A[%d][%d]: ",i,j);
scanf("%f",&A[i][j]);
}
printf("Dati termenii liberi:\n");
for (i=1;i<=n;i++)
        {
        printf("B[%d]: ",i);
        scanf("%f",&B[i]);
        }
        printf ("Intoduceti nr. maxim de iteratii Nmax= ");
scanf("%d",&Nmax);
for(i=1;i<=n;i++)
        x[i]=0;
for(k=1;k<=Nmax;k++)
{
for(i=1;i<=n;i++)
{
s=0;
for(j=1;j<=i-1;j++)
s=s+A[i][j] * y[j];
for(j=i+1;j<=n;j++)
s=s+A[i][j] * x[j];
y[i]= (B[i]-s)/A[i][i];
x[i]=y[i];
}
}
printf("Solutia dupa %d iteratii este:\n", Nmax);
for(i=1;i<=n;i++)
printf("x[%d]=%f \n",i,x[i]);
          getch();
}
```

Soluția obținută cu varianta algoritmului ce folosește doar numărul maxim de iterații pentru oprire, pentru sistemul de la exemplul IV.2., în cazul în care acest număr maxim este 10 este:



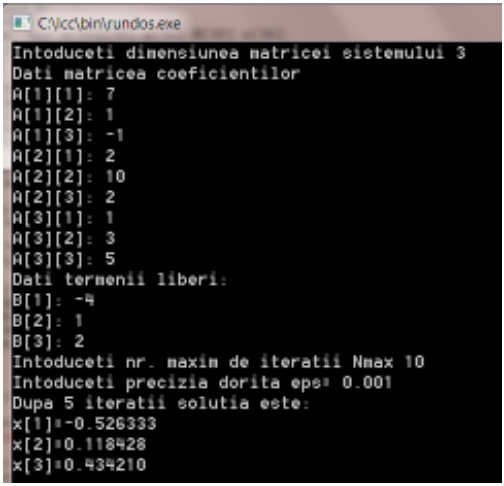
/***Program Metoda Seidel Gauss-epsilon si Nmax***/

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
float s, max, eps,A[30][30],x[30],B[30],y[30];
int n,Nmax,j,i,k;
void main()
{
printf ("Intoduceti dimensiunea matricei sistemului ");
scanf("%d",&n);
printf("Dati matricea coeficientilor\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
```

```
{
printf("A[%d][%d]: ",i,j);
scanf("%f",&A[i][j]);
}
printf("Dati termenii liberi:\n");
for (i=1;i<=n;i++)
        {
        printf("B[%d]: ",i);
        scanf("%f",&B[i]);
        }
        printf ("Intoduceti nr. maxim de iteratii m= ");
scanf("%d",&Nmax);
printf ("Intoduceti precizia dorita eps= ");
scanf("%f",&eps);
for(i=1;i<=n;i++) x[i]=0;
k=0;
do
   {
  for(i=1;i<=n;i++)
            {
              s=0;
         for(j=1;j<=i-1;j++)
           s=s+A[i][j] * y[j];
           for(j=i+1;j<=n;j++)
       s=s+A[i][j] * x[j];
         y[i]= (B[i]-s)/A[i][i];
            }
 max=fabs(y[1]-x[1]);
 for(i=2;i<=n;i++)
        if (max<fabs(y[i]-x[i]))
          max=fabs((y[i]-x[i]));
 for(i=1;i<=n;i++) x[i]=y[i];
 k=k+1;
   }
while (max>=eps && k<=Nmax);
```

```
        if (k<=Nmax)
                {
          printf("Dupa %d iteratii solutia este:\n",k);
                for(i=1;i<=n;i++)
                printf("x[%d]=%f \n",i,x[i]);
                 }
        else printf("S-a depasit numarul maxim de iteratii");
          getch();
}
```

Exemplul IV.4. Să se rezolve sistemul de ecuaţii din exemplul IV.3. prin metoda Gauss-Seidel, cu aceeaşi eroare admisibilă pentru soluţia finală.



Remarcăm faptul că se ajunge la precizia dorită după doar 5 iteraţii prin folosirea metodei Seidel-Gauss. Acelaşi sistem de ecuaţii a fost rezolvat cu o aceeaşi eroare admisibilă în 8 paşi prin metoda Jacobi. Se deduce de aici accelerarea convergenţei în cazul metodei Gauss-Seidel.

V. METODE NUMERICE PENTRU REZOLVAREA ECUAŢIILOR ŞI SISTEMELOR DE ECUAŢII NELINIARE

V.1. METODE NUMERICE PENTRU REZOLVAREA ECUAŢIILOR NELINIARE

Pentru o funcţie : f: [a,b] $\to$R, continuă şi derivabilă (sau doar când $f \in$ C ([a,b])) dorim să găsim rădăcinile ecuaţiei:

$$f(x) = 0. \qquad (*)$$

Dacă f este o funcţie polinomială de gradul 1, 2, sau 3 există formule generale de rezolvare, dar dacă gradul lui f este mai mare decât 4 nu mai dispunem de astfel de formule.

Definiţia V.1 Spunem că o rădăcină reală α este separată într-un interval [a,b] dacă acest interval conţine o singură radacină a ecuaţiei, adică doar rădăcina α.

Definiţia V.2 Spunem că o rădacină α, separată într-un interval [*a*,*b*], este localizată în limitele unei precizii ε, prin x_n, într-un interval $[a_n, b_n]$, dacă este îndeplinită una din condiţiile:

$$|f(x_n)| < \varepsilon, \qquad (1)$$

$$|b_n - a_n| < \varepsilon. \qquad (2)$$

Relaţiile (1), respectiv (2), sunt folosite la oprirea algoritmilor de determinare a rădăcinilor ecuaţiilor neliniare.

Cele mai cunoscute metode numerice pentru aflarea rădăcinilor ecuaţiilor neliniare sunt: metoda bisecţiei, metoda coardei, metoda secantei, metoda aproximaţiilor succesive (sau a contracţiei), metoda lui Newton (sau a tangentei), etc.

V.2. METODA BISECŢIEI

Această metodă constă în reducerea intervalului de separare, prin înjumătăţiri repetate şi selectarea subintervalului în care se află rădacina. Procesul se încheie în momentul satisfacerii uneia dintre condiţiile (1) sau (2).

Considerăm deci ecuaţia: $f(x)=0,\ x\in[a,b]$ cu f continuă, astfel încât $f(a)f(b)<0$, iar în $[a,b]$ a fost separată o soluţie a ecuaţiei.

Introducem notaţia $a_0=a$, $b_0=b$. Determinăm mijlocul intervalului $[a_0,b_0]$, pe care îl notăm cu $x_0=\dfrac{a_0+b_0}{2}$.

Acesta separă intervalul iniţal în două subintervale $[a_0,x_0]$, $[x_0,b_0]$, rădăcina α găsindu-se într-unul din ele, şi anume în acela la capetele căruia funcţia prezintă semne contrare. Vom nota acel interval cu $[a_1,b_1]$. În mod evident:

$$|\alpha-x_0|<\frac{b_0-a_0}{2}$$

Urmând acelaşi procedeu se obţin intervalele $[a_1,b_1]$, $[a_2,b_2]$, $[a_3,b_3]$,..., $[a_n,b_n]$ şi mijloacele acestora:

$$x_n=\frac{a_n+b_n}{2},\ n\geq 0 \qquad (3)$$

La fiecare pas selectarea intervalului următor se face astfel:

Dacă $f(a_n)f(x_n)<0$ atunci:

$a_{n+1}=a_n$, $b_{n+1}=x_n$,

altfel

$a_{n+1}=x_n$, $b_{n+1}=b_n$. (4)

Se observă că:

$$b_{n+1} - a_{n+1} = \frac{b_n - a_n}{2} = \ldots = \frac{b_0 - a_0}{2^{n+1}} \quad (5)$$

şi $$|\alpha - x_n| < b_{n+1} - a_{n+1} = \frac{b_0 - a_0}{2^{n+1}} \quad (6)$$

Relaţia (6) ne arată că şirurile $\{a_n\}$ şi $\{b_n\}$ sunt convergente către α.

Algoritmul metodei bisecţie
Date de intrare:
a,*b*= capetele intervalului de separare,

f = funcţia căreia i se localizează rădăcina.

ε = precizia determinării.
Date de ieşire:
x = valoarea aproximativă a rădăcinii din intervalul considerat.

1. Se iniţializează valoarea aproximativă a rădăcinii

(valoarea corespunzătoare mijlocului intervalului dat)

$x \leftarrow (a+b)/2$

2. Atâta timp cât $|f(x)| \geq \varepsilon$ repetă:
   - daca $f(a).f(x)<0$ atunci
     - $b \leftarrow x$ altfel
     - $a \leftarrow x$
   - $x \leftarrow (a+b)/2$.
3. Dacă $f(x) = 0$ atunci
   - se afişează *x* precizându-se că este soluţia exactă a ecuaţiei neliniare, altfel
   - se afişează x precizându-se că este valoarea aproximativă a rădăcinii
4. Stop

Observaţia V.1. Oprirea algoritmului de mai sus se bazează pe o condiţie de tip (1). Ea poate fi înlocuită de o condiţie de tip (2), caz în care algoritmul devine:

1. Se iniţializează valoarea aproximativă a rădăcinii (valoarea corespunzătoare mijlocului intervalului dat)
$x \leftarrow (a+b)/2$
2. Atâta timp cât $|b-a| \geq \varepsilon$ repetă
   - daca $f(a).f(x)<0$ atunci
     - $b \leftarrow x$ altfel
     - $a \leftarrow x$
   - $x \leftarrow (a+b)/2$.
3. Se afişează valoarea lui x
4. Stop

Se observă că în acest caz nu se mai precizează dacă este vorba de soluţia exactă sau aproximativă a ecuaţiei.

Observaţia V.2. Se poate introduce o condiţie de oprire şi cu ajutorul numărului maxim admis de iteraţii, respectiv cu ajutorul unor combinaţii între condiţiile amintite.

Exemplul V.1. Să se găsească valoarea aproximativă a lui $\sqrt{3}$ folosind metoda bisecţiei şi executând un număr maxim de 5 iteraţii.

Soluţie:

Ataşăm ecuaţia corespunzătoare: $x^2 - 3 = 0 \Rightarrow f(x) = x^3 - 3$

Considerăm $f:[1,2] \to R \Rightarrow [a,b] = [1,2]$. În acest interval este separată o singură rădăcină a acestei ecuaţii. Acest lucru poate fi justificat simplu deoarece se ştie că rădăcinile acestei ecuaţii sunt $\pm\sqrt{3}$, iar $f(1)f(2) < 0$.

Calculăm $x_0 = \frac{a+b}{2} = \frac{1+2}{2} = \frac{3}{2}$

$$f(2)\cdot f\left(\frac{3}{2}\right) = 1\cdot\left(-\frac{3}{4}\right) = \frac{3}{4} < 0 \Rightarrow [a,b] = \left[\frac{3}{2},2\right]$$

Calculăm $x_1 = \frac{a_1+b_1}{2} = \frac{7}{4} = 1{,}75$

$$f\left(\frac{3}{2}\right)\cdot f\left(\frac{7}{4}\right) = -\frac{3}{4}\cdot\frac{1}{16} < 0 \Rightarrow [a_2,b_2] = \left[\frac{3}{2},\frac{7}{4}\right]$$

Calculăm $x_2 = \frac{a_2+b_2}{2} = \frac{\frac{3}{2}+\frac{7}{4}}{2} = \frac{13}{8}$

$$\Rightarrow f\left(\frac{13}{8}\right)\cdot f\left(\frac{7}{4}\right) = \left(\frac{-169}{64}-3\right)\cdot\left(\frac{49}{16}-3\right) < 0 \Rightarrow [a_3,b_3] = \left[\frac{13}{8},\frac{7}{4}\right]$$

Calculăm $x_3 = \frac{a_3+b_3}{2} = \frac{\frac{13}{8}+\frac{7}{4}}{2} = \frac{27}{16}$

$$f\left(\frac{27}{16}\right)\cdot f\left(\frac{7}{4}\right) = (-)\cdot(+) < 0 \Rightarrow [a_4,b_4] = \left[\frac{27}{16},\frac{7}{4}\right]$$

Calculăm $x_4 = \frac{a_4+b_4}{2} = \frac{\frac{27}{16}+\frac{7}{4}}{2} = \frac{55}{32} = 1.71875$

$$f\left(\frac{55}{32}\right)\cdot f\left(\frac{7}{4}\right) = (-)\cdot(+) < 0 \Rightarrow [a_5,b_5] = \left[\frac{55}{32},\frac{7}{4}\right] \Rightarrow \frac{55}{32} < \sqrt{3} < \frac{7}{4}$$

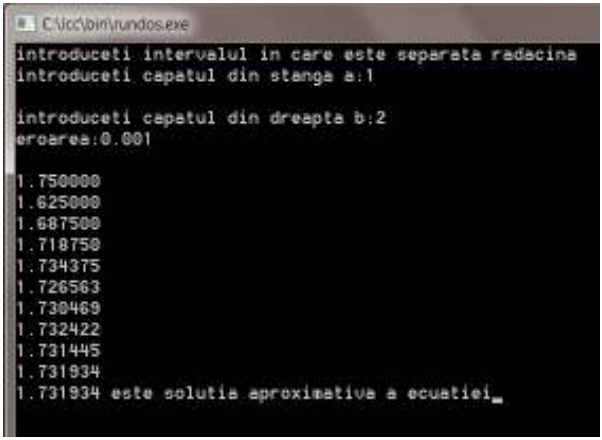
Astfel $x_5 = \frac{a_5+b_5}{2} = 1.734375$, este valoarea aproximativă cerută.

Programul următor, realizat în limbajul de programare C, se referă la calculul valorii aproximative a lui $\sqrt{3}$ cu ajutorul metodei bisecţiei. El poate fi folosit și pentru aflarea valorilor aproximative ale rădăcinior reale ale altor ecuații, modificând expresia fucției cu cea corespunzăoare ecuației în cauză).

```
/* Program Bisectia1-condiție tip (1) */
#include<conio.h>
#include<stdio.h>
#include<math.h>
float a,b,eps,x,C;
float F(float u)
{return u*u-3;}
void main ()
{
printf("introduceti intervalul in care este separata radacina \n");
printf("introduceti capatul din stanga a:");
scanf("%f",&a);
printf("\nintroduceti capatul din dreapta b:");
scanf("%f",&b);
printf("eroarea:");
scanf("%f",&eps);
x=(a+b)/2;
A=F(x);
while (fabs(A)>=eps)
        {
        C=F(a)*F(x);
        if (C<0)
        b=x;
        else a=x;
        x=(a+b)/2;
        printf("\n%f",x);  /*afiseaza valorile intermediare*/
```

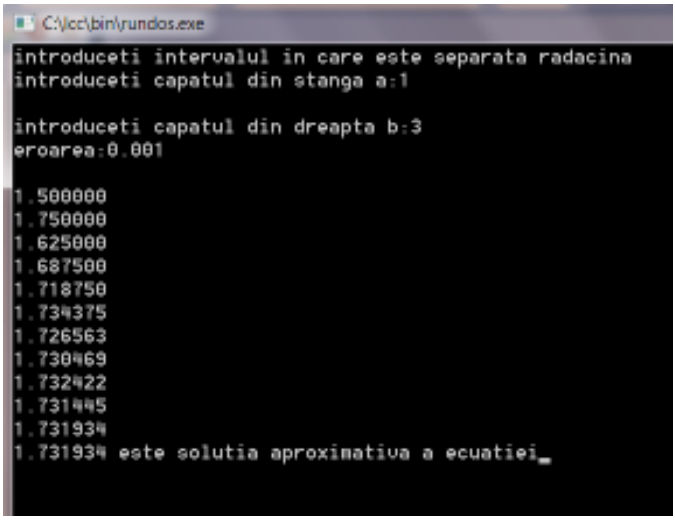
```
        A=F(x);
        }
if (A==0)
        printf ("\n%f este solutia exacta a ecuatiei", x);
else
        printf ("\n%f este solutia aproximativa a ecuatiei", x);
getch();
}
```

Rezultatele obținute prin rularea acestui program pentru datele de intrare de mai jos sunt:



Se observă că $x_5 = 1.734375$, adică cea obținută anterior.

Modificând intervalul de separație dar menținând aceeași eroare admisibilă obținem soluția dorită după un număr mai mare de iterații, așa cum se observă din datele următoare:



V.3. METODA COARDEI

Fie *f* continuă şi derivabilă de două ori pe $[a,b]$, astfel încât $f(a)f(b)<0$ şi $f''(x)$ păstrează semn constant pe intervalul considerat. De asemena presupunem că în $[a,b]$ a fost separată o soluţie a ecuaţiei: $f(x)=0$.

Metoda coardei sau a falsei poziţii constă în construirea unui şir de aproximaţii care să conveargă către soluţia ecuaţiei separată în acest interval, în care termenii şirului reprezintă abscisele punctelor de intersecţie ale axei *Ox* cu diverse drepte determinate de două puncte situate pe graficul funcţiei *f,* convenabil alese.

Prima aproximaţie, notată x_1, reprezintă abscisa punctului în care *Ox* taie dreapta ce trece prin punctele $(a, f(a))$ şi $(b, f(b))$:

$$\frac{x-a}{b-a}=\frac{y-f(a)}{f(b)-f(a)}.$$

Alegând $y=0$ obţinem:

$$x_1=a-\frac{f(a)(b-a)}{f(b)-f(a)}.$$

Se construieşte apoi o nouă dreaptă ce trece prin $(x_1, f(x_1))$ şi un al doile punct în funcţie de intervalul în care se află rădăcina ecuaţiei: (a, x_1) sau (x_1, b), continuându-se poi în acelaşi mod.

Dacă $f(a)f(x_1)<0$ atunci al doilea punct al dreptei va fi $(a, f(a))$, şi obţinem un şir descrescător şi mărginit, deci convergent, dat de relaţia de recurenţă:

$$x_{n+1}=x_n-\frac{f(x_n)(x_n-a)}{f(x_n)-f(a)}.$$

Dacă $f(x_1)f(b)<0$ atunci al doilea punct al dreptei va fi $(b, f(b))$, şi obţinem un şir crescător şi mărginit, deci convergent dat prin relaţia de recurenţă:

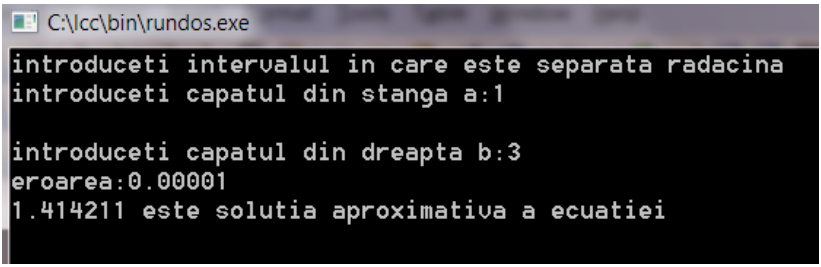
$$x_{n+1}=x_n-\frac{f(x_n)(b-x_n)}{f(b)-f(x_n)}.$$

Oprirea algoritmului se face în mod analog cazului precedent, adică prin folosirea condiţiilor de localizare de tip (1) sau (2), sau prin folosirea unui număr maxim de iteraţii admis sau a oricărei combinaţii între acestea.

Următorul program în limbajul C implementează algoritmul acestei metode pentru aflarea rădăcinilor ecuaţiei $x^2-2=0$, în cazul în care oprirea algoritmului foloseşte o condiţie de tip (1).

```
/* Program Metoda Coardei 1-condiție de tip(1) */
#include<conio.h>
#include<stdio.h>
#include <math.h>
float a,b,eps,x,A, C;
float F(float u)
{return u*u-2;}
void main ()
{
printf("introduceti intervalul in care este separata radacina\n");
printf("introduceti capatul din stanga a:");
scanf("%f",&a);
printf("\nintroduceti capatul din dreapta b:");
scanf("%f",&b);
printf("eroarea:");
scanf("%f",&eps);
x=a-F(a)*(b-a)/(F(b)-(Fa));
A=F(x);
  while (fabs(A)>=eps)
            {
            C=F(x)*F(a);
            if (C<0)
              x=x-F(x)*(x-a)/(F(x)-F(a));
                    else
{
x=x-F(x)*(b-x)/(F(b)-F(x));
                    a=b;
}
A=F(x);
}
  if (A==0)
       printf ("%f este solutia exacta a ecuatiei", x);
  else
       printf ("%f este solutia aproximativa a ecuatiei", x);
getch();
}
```

Rezultatele obținute la rulare sunt:

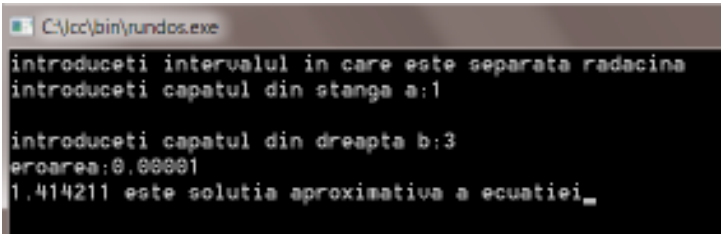


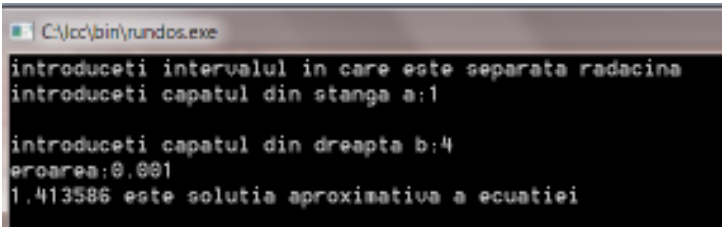
Varianta care folosește o condiție de oprire de tip (2) este:

```
/* Program Metoda Coardei 2 –conditie de tip (2) */
#include<conio.h>
#include<stdio.h>
#include <math.h>
float a,b,eps,x,y,A,C,ER;
float F(float u)
{return u*u-2;}
void main ()
{
printf("introduceti intervalul in care este separata radacina\n");
printf("introduceti capatul din stanga a:");
scanf("%f",&a);
printf("\nintroduceti capatul din dreapta b:");
scanf("%f",&b);
printf("eroarea:");
scanf("%f",&eps);
x=a-F(a)*(b-a)/(F(b)-F(a));
ER=b-a;
  while (fabs(ER)>=eps)
            {
            C=F(x)*F(a);
            if (C<0)
              y=x-F(x)*(x-a)/(F(x)-F(a));
                    else
{
y=x-F(x)*(b-x)/(F(b)-F(x));
                    a=b;
```

```
}
ER=y-x;
x=y;
}
A=F(x);
  if (A==0)
          printf ("%f este solutia exacta a ecuatiei", x);
  else
  printf ("%f este solutia aproximativa a ecuatiei", x);
getch();
}
```

Rezultate obținute la rulare:





V.4. METODA SECANTEI

Considerăm din nou ecuaţia: $f(x)=0,\ x\in[a,b]$ cu f continuă şi derivabilă de două ori pe $[a,b]$, astfel încât în $[a,b]$ a fost separată o soluţie a ecuaţiei, pe care o notăm cu α.

Presupunând că $f'(x) \neq 0$ pe $[a,b]$, se poate demonstra că şirul de numere reale $(x_k)_{k\geq 0}$ definit de relaţia de recurenţă: $x_{k+1} = \dfrac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$, cu $x_0, x_1 \in [a,b]$ este un şir convergent către α.

Metoda secantei constă în construcţia acestui şir de aproximaţii, ce se poate demonstra că este un şir convergent la α.

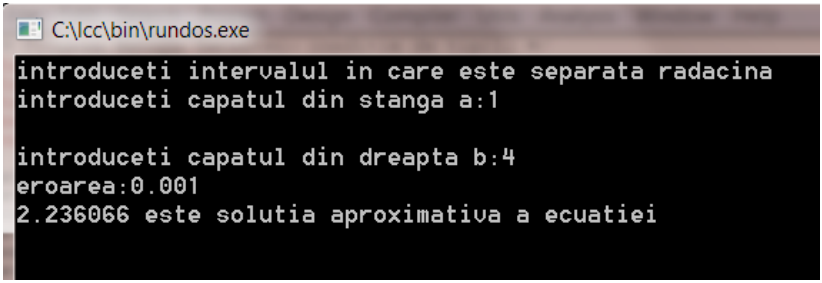
Algoritmul foloseşte ca date de intrare valorile iniţiale ale şirului aproximaţiilor, adică valorile x_0, x_1 şi expresia lui f, la care se adaugă fie numărul maxim admis de iteraţii, fie o eroare admisibilă dată, ε_{adm}, fie ambele, în funcţie de criteriul ales pentru oprirea acestuia (se poate folosi oricare din criteriile amintite la celelalte metode). Valorile iniţiale se pot alege chiar capetele intervalului de separaţie.

Observaţie: valorile din şirul recursiv ce se construieşte nu reprezintă nimic altceva decât punctele în care secanta la graficul funcţiei *f*, dusă prin punctele de pe graficul ei, de coordonate $(x_{k-1}, f(x_{k-1}))$ şi $(x_k, f(x_k))$ intersectează axa O*x*.

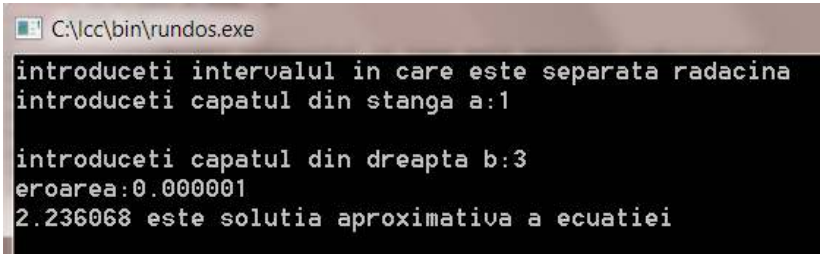
Pentru varianta algoritmului metodei secantei ce foloseşte eroarea admisibilă dată şi o condiţie de oprire de tip (1) s-a realizat următorul program în C, care permite calculul valorii aproxiamtive a lui $\sqrt{5}$ cu eroarea ε_{adm}, introdusă de la tastatură. El poate fi uşor modificat pentru a permite rezolvarea oricărei ecuaţii neliniare analog ca în cazul celorlalte metode.

```
/* Program Metoda Secantei- conditie de tip(1) */
#include<conio.h>
#include<stdio.h>
#include<math.h>
float a,b,eps,x, A;
float F(float u)
{return u*u-5;}
void main ()
{
printf("introduceti intervalul in care este separata radacina\n");
printf("introduceti capatul din stanga a:");
scanf("%lf",&a);
printf("\nintroduceti capatul din dreapta b:");
scanf("%lf",&b);
printf("eroarea:");
scanf("%lf",&eps);
do
  {
        x=(a*F(b)-b*F(a))/(F(b)-F(a));
        a=b;
        b=x;
        A=F(x);
  }
  while (fabs(A)>=eps);
  if (fabs(A)==0)
        printf ("%f este solutia exacta a ecuatiei", x);
  else
        printf ("%f este solutia aproximativa a ecuatiei", x);
getch();
 }
```

Considrând intervalul $[1;4]$ şi eroarea admisibilă dată 0.001, s-a obţinut pentru $\sqrt{5}$ valoarea aproximativă: 2.236066.



Pentru alte date de intrare se obţin rezultatele următoare:



V.5. METODA APROXIMAŢIILOR SUCCESIVE PENTRU REZOLVAREA ECUAŢIILOR ŞI SISTEMELOR NELINIARE

Fie ecuaţia: $f(x)=0$ (*), în care $f:R\to R$ este o funcţie dată, căreia ne propunem să-i găsim rădăcina dintr-un anumit interval [a,b].

Dacă rescriem ecuaţia sub forma echivalentă:

$$x=\varphi(x)$$

cu $\varphi:[a,b]\to[a,b]$ contracţie, cu coeficientul de contracţie $q < 1$, putem s-o rezolvăm cu ajutorul metodei aproximaţiilor succesive.

Rădăcina din intervalul [a,b] reprezintă unicul punct fix al aplicaţiei φ. Acesta se obţine ca limită a şirului aproximaţiilor succesive, date de:

$x_n = \varphi(x_{n-1}), n \geq 1$, unde $x_0 \in [a,b]$ arbitrar.

Relaţiile:

$$|x_n - x| \leq \frac{q^n}{1-q}|x_1 - x_0|, n \geq 0$$

$$|x_n - x| \leq \frac{q}{1-q}|x_n - x_{n-1}|, n \geq 1$$

ne indică cât de bună este valoarea aproximativă calculată la pasul *n*.

Cel mai des întâlnit caz în care putem aplica metoda aproximaţiilor succesive este:

1. Dacă $\varphi : [a,b] \to [a,b]$ este continuă pe [*a*,*b*], derivabilă pe (*a*, *b*), şi $|\varphi'(x)| \leq q < 1$ oricare ar fi $x \in (a,b)$, atunci φ este o contracţie.

Exemplul V.2. Să se găsească valoarea aproximativă a rădăcinii din intervalul $[2.2\,;4]$ ecuaţiei $x^3 - 2x^2 = 5$ folosind metoda aproximaţiilor succesive şi executând un număr maxim de 5 iteraţii.

Soluţie:

Evident $f(x) = x^3 - 2x^2 - 5$ are proprietatea că $f(2.2)f(4) < 0$, iar $f'(x) = 3x^2 - 4x = x(3x - 4 > 0)$ pe intervalul considerat , deci f admite o singură rădăcină în $[2.2\,;4]$.

Scriem ecuaţia astfel: $x = \frac{5}{x^2} + 2$

Observăm că dacă $x \in [2.2\,;4] \Rightarrow \frac{5}{16} + 2 \leq \frac{5}{x^2} + 2 \leq \frac{5}{4.4} + 2$.

Astfel considerând $\varphi : [2.2;4] \to [2.2;4]$, $\varphi(x) = \frac{5}{x^2} + 2$, aceasta este o contracție, deoarece $\varphi'(x) = -\frac{10}{x^3}$, iar $|\varphi'(x)| \le \frac{10}{2.2^3} = \frac{10}{10.648} < 1$.

Alegând $x_0 = 2.5$ găsim după cele cinci iterații valorile:

$x_1 = \varphi(2.5) = 2.8$

$x_2 = \varphi(2.8) = 2.637755$

$x_3 = \varphi(2.637755) = 2.718623$

$x_4 = \varphi(2.718623) = 2.676507$

$x_5 = \varphi(2.676507) = 2.697965$.

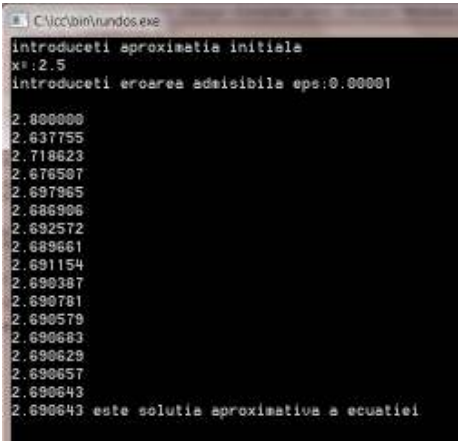
Un program în C bazat pe metoda aproximațiilor succesive, ce folosește o condiție de oprire de tip (1), pentru problema anterioară este:

```
/* Program Contractia 1 */
#include<conio.h>
#include<stdio.h>
#include <math.h>
float x,eps, A;
float F(float u)
{return 5/ (u*u)+2;}

void main ()
{
printf("introduceti aproximatia initiala \n");
printf("x=:");
scanf("%f",&x);
printf("introduceti eroarea admisibila eps:");
scanf("%f",&eps);
A=x-F(x);
while (fabs(A)>=eps)
```

```
  {
          x=F(x);
printf("\n%f",x);   /*afiseaza valorile intermediare*/
A=x-F(x);
}
if (A==0)
  printf ("\n%f este solutia exacta a ecuatiei", x);
else
  printf ("\n%f este solutia aproximativa a ecuatiei", x);
getch();
}
```

Linia evidențiată din cadrul lui permite ca, după rularea sa, să obținem și valorile intermediare calculate la fiecare iterație. Rezultatele rulării codului sursă de mai sus sunt prezentate în continuare.



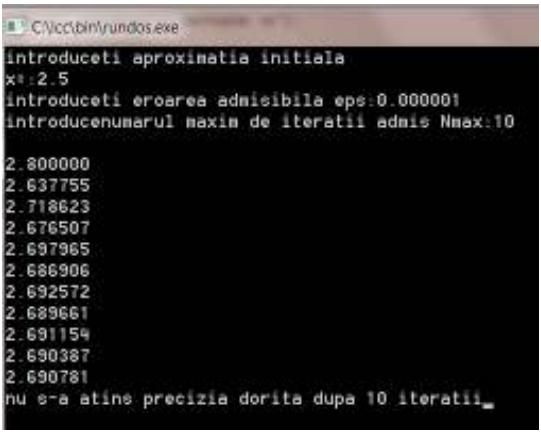
Un program bazat pe metoda aproximațiilor succesive, ce folosește o condiție de oprire de tip (1) combinată cu cea care folosește un număr maxim de iterații admis introdus de la tastatură , notat Nmax, pentru problema anterioară, este prezentat în continuare.

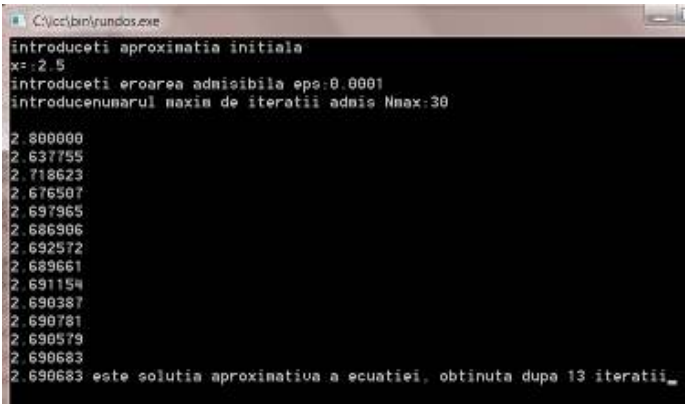
Variabila întregă Nit înregistrează numărul iteraţiilor efectuate până la atingerea preciziei dorite. Dacă nu este atinsă precizia dorită programul afişează acest lucru.

```
/* Program Contractia 2 */
#include<conio.h>
#include<stdio.h>
#include <math.h>
float x,eps,A;
int Nmax, Nit;
float F(float u)
{return 5/ (u*u)+2;}
void main ()
{
printf("introduceti aproximatia initiala \n");
printf("x=:");
scanf("%f",&x);
printf("introduceti eroarea admisibila eps:");
scanf("%f",&eps);
printf("introducenumarul maxim de iteratii admis Nmax:");
scanf("%d",&Nmax);
A=x-F(x);
Nit=0;
while ((fabs(A)>=eps) && (Nit<=Nmax))
  {
          x=F(x);
printf("\n%f",x);   /*afiseaza valorile intermediare*/
A=x-F(x);
Nit=Nit+1;
}
if (Nit>Nmax)
  printf ("\nnu s-a atins precizia dorita in %d iteratii", Nmax);
else
  {
if (A==0)
  printf ("\n%f este solutia exacta a ecuatiei", x);
```

```
else
  printf ("\n%f este solutia aproximativa a ecuatiei, obtinuta dupa %d iteratii", x, Nit);
}
getch();
}
```

Rularea acestui program pentru diverse date de intrare a condus la rezultatele următoare.





V.6. METODA APROXIMAŢIILOR SUCCESIVE PENTRU REZOLVAREA SISTEMELOR DE ECUAŢII NELINIARE

Vom considera sistemul de ecuaţii neliniare:

$$\begin{cases} F(x,y)=0 \\ G(x,y)=0 \end{cases}$$

transcris sub forma echivalentă:

$$\begin{cases} x=f(x,y) \\ y=g(x,y) \end{cases}$$

Fie $D=\{(x,y)\in R^2 : a\le x\le b, c\le y\le d\}$.Dacă $(f(x,y),g(x,y))\in D$ oricare ar fi $(x,y)\in D$, putem defini aplicaţia:

$$\varphi : D\to D, \varphi(x,y)=\big(f(x,y),g(x,y)\big), x,y\in D .$$

Sistemul dat este echivalent cu ecuaţia:

$$\varphi(x,y)=(x,y) \qquad (**)$$

Cele mai importante situaţii în care φ este o contracţie.

1. Să presupunem că există q < 1 astfel încât în raport cu norma $\|h\|=\max\{|h(x,y)| : (x,y)\in D\}$ să avem:

$$\left\|\frac{\partial f}{\partial x}\right\|+\left\|\frac{\partial g}{\partial x}\right\|\le q ; \qquad \left\|\frac{\partial f}{\partial y}\right\|+\left\|\frac{\partial g}{\partial y}\right\|\le q$$

În aceste condiţii φ este o contracţie, dacă ne raportăm la metrica definită prin: $\rho[(x,y),(z,t)]=|x-z|+|y-t|$, care face ca ($D$, ρ) să fie un spaţiu metric complet.

2. Dacă există q < 1 astfel încât:

$$\left\|\frac{\partial f}{\partial x}\right\|+\left\|\frac{\partial f}{\partial y}\right\|\le q ; \qquad \left\|\frac{\partial g}{\partial x}\right\|+\left\|\frac{\partial g}{\partial y}\right\|\le q$$

atunci φ este o contracţie dacă ne raportăm la metrica definită prin:

$$\rho[(x,y),(z,t)] = max\{|x-z|,|y-t|\},$$

care face ca (D, ρ) să fie un spaţiu metric complet.

Astfel, în oricare din aceste situaţii, ecuaţia (**) are o singură soluţie în D, soluţie care poate fi obţinută prin metoda aproximaţiilor succesive; aceasta va fi şi unica soluţie în D a sistemului dat;

Soluţia sistemului neliniar poate fi obţinută prin metoda aproximaţiilor succesive pornind de la $(x_0, y_0) \in D$ şi luând $(x_n, y_n) = \varphi(x_{n-1}, y_{n-1}), n \geq 1$, adică:

$$\begin{cases} x_n = f(x_{n-1}, y_{n-1}) \\ y_n = g(x_{n-1}, y_{n-1}) \end{cases}, n \geq 1$$

V.7. METODA NEWTON

Considerăm ecuaţia: $f(x)=0$, unde $f : R \to R$ derivabilă, cu derivata nenulă într-un interval $[a,b]$ şi presupunem că am separat o rădăcină α în acest interval.

Fie $x_0 \in [a,b]$ o aproximaţie iniţială a rădăcinii α.

Notăm cu x_1 intersecţia tangentei la graficul funcţiei în x_0 cu axa Ox. Ecuaţia tangentei este:

$$y - f(x_0) = f'(x_0)(x - x_0) \Rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Notăm cu x_2 intersecţia tangentei la graficul funcţiei în x_1 cu axa Ox şi continuăm analog obţinând procedeul ce poartă numele de *metoda Newton Raphson* sau *metoda tangentei*.

Metoda Newton constă astfel în a găsi un şir de aproximaţii $(x_n)_{n \geq 1}$ pentru rădăcină, în felul următor: pentru fiecare $n \geq 1$, x_n este

punctul de intersecţie dintre axa *Ox* şi tangenta la graficul lui *f* în punctul de abscisă x_{n-1}.

Ecuaţia acestei tangente este:

$$y - f(x_{n-1}) = f'(x_{n-1})(x - x_{n-1})$$

Intersecţia ei cu axa *Ox* este punctul de abscisă:

$$x = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

Prin urmare procesul iterativ al metodei lui Newton poate fi descris astfel:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, n \geq 0$$

Observaţie: Dacă f'' are semn constant pe $[a,b]$ se poate alege x_0 un punct în care are loc: $f(x_0)f''(x_0) > 0$.

Un program în C, ce foloseşte o condiţie de oprire de tip (1), pentru metoda Newton este prezentat în continuare.

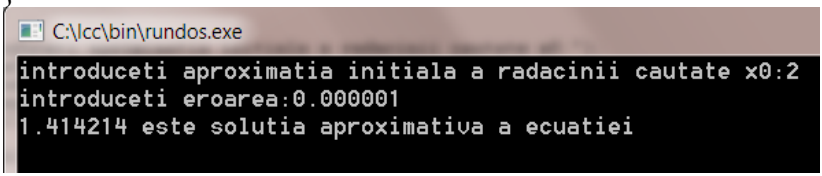
```
/* Program Newton*/
#include<conio.h>
#include<stdio.h>
#include<math.h>
float x0, x, eps, A;
float f(float u)
{return f(u);}
float Df(float u)
{return fderivat(u);}
 void main ()
{
printf("introduceti aproximatia initiala a radacinii cautate x0:");
scanf("%f",&x0);
printf("introduceti eroarea:");
scanf("%f",&eps);
x=x0-f(x0)/Df(x0);
```

```
A=f(x);
while (fabs(A)>=eps)
  {
  x0=x;
  x=x0-f(x0)/Df(x0);
  A=f(x);
}
  if (A==0)
          printf ("%f este solutia exacta a ecuatiei", x);
  else
  printf ("%f este solutia aproximativa a ecuatiei", x);
getch();
}
```

Pentru rezolvarea unei probleme concrete trebuie înlocuite expresiile funcțiilor f , respectiv *Df*, $Df(x) = f'(x)$, în punctele corespunzătoare. Astfel pentru a găsi valoarea lui $\sqrt{2}$ putem rula programul următor:

```
/* Program Newton*/
#include<conio.h>
#include<stdio.h>
#include<math.h>
float x0,x,eps,A;
float f(float u)
{return u*u-2;}
float Df(float u)
{return 2*u;}
void main ()
{
printf("introduceti aproximatia initiala a radacinii cautate
x0:");
scanf("%f",&x0);
printf("introduceti eroarea:");
scanf("%f",&eps);
```

```
x=x0-f(x0)/Df(x0);
A=f(x);
while (fabs(A)>=eps)
  {
  x0=x;
x=x0-f(x0)/Df(x0);
A=f(x);
}
  if (A==0)
          printf ("%f este solutia exacta a ecuatiei", x);
  else
  printf ("%f este solutia aproximativa a ecuatiei", x);
getch();
}
```



Observaţia V.7. Metoda poate fi aplicată şi la rezolvarea sistemelor neliniare de forma:

$$\begin{cases} f_1(x_1,\ldots,x_m)=0 \\ f_2(x_1,\ldots,x_m)=0 \\ \ldots \\ f_m(x_1,\ldots,x_m)=0 \end{cases}.$$

În acest caz locul funcţiei reale f este luat de funcţia vectorială de variabilă vectorială:

$$F: R^m \to R^m, \quad F(\bar{x})=\begin{pmatrix} f_1 \\ \ldots \\ f_m \end{pmatrix}, \bar{x} \in R^m.$$

Dacă $f_i : R^m \to R$ au derivate parţiale de ordinul întâi continue atunci există $F'(\bar{x})$ şi acesta este operator liniar de la R^m în R^m, dat de matricea (jacobianul):

$$F'(\bar{x}) = \left(\frac{\partial f_i}{\partial x_j} \right)_{1 \le i, j \le m} .$$

Dacă $(\exists)\left[F'(\bar{x})\right]^{-1} \Rightarrow \bar{x}^{(n+1)} = \bar{x}^{(n)} - \left[F'\left(\bar{x}^{(n)}\right)\right]^{-1} F\left(\bar{x}^{(n)}\right)$, aceasta fiind formula recursivă pentru calculul elementelor din şirul aroximaţiilor. Indicele (n) aşezat sus desemnează elementul calculat la pasul *n*.

Observaţia V.8.

Metoda Newton modificată înlocuieşte pe $\left[F'\left(\bar{x}^{(n)}\right)\right]^{-1}$ cu $\left[F'\left(\bar{x}^{(0)}\right)\right]^{-1}$, ceea ce implică un volum de calcule mult mai mic deoarece la fiecare pas se foloseşte aceeaşi inversă –cea de la primul pas. Procesul iterativ astfel rezultat este convergent, dar converge mai lent decât cel din cazul metodei Newton.

Exemplul V.5. Să considerăm ecuaţia:

$x^3 + x + 1 = 0$.

Să se aplice metoda lui Newton pentru aflarea aproximativă a unei rădăcini din intervalul $\left[-1, -\frac{1}{2}\right]$. Se foloseşte o condiţie de oprire de tip (1), cu $\varepsilon = 10^{-3}$

Soluţie:

Notând $f(x) = x^3 + x + 1$ avem:

$$f(-1) = -1 \text{ şi } f\left(\frac{-1}{2}\right) = -\frac{1}{8} - \frac{1}{2} + 1 = \frac{3}{8}.$$

De asemenea avem:

$$f'(x) = 3x^2 + 1 > 0 \text{ şi } f''(x) = 6x .$$

Deoarece $f'(x) > 0\ (\forall)x \in \left[-1, -\frac{1}{2}\right]$, f este strict crescătoare pe intervalul considerat. Cum $f(-1)f\left(-\frac{1}{2}\right) < 0$ atunci ecuaţia dată are o singură rădăcină în intervalul considrat.

Deoarece $(\forall)x \in \left[-1, -\frac{1}{2}\right]$ avem: $f''(x) < 0$, putem aplica metoda Newton alegând o aproximaţie iniţială x_0 care satisface condiţia: $f(x_0)f''(x_0) > 0$, deci alegem $x_0 = -1$.

Obţinem succesiv:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = -1 + \frac{-1}{4} = -0.75 .$$

Evaluăm $f(x_1) = -0.171875$.

Cum $|f(x_1)| = 0.171875 > \varepsilon$, trecem la pasul următor

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = -0.75 + \frac{-0.171875}{2.6875} = -0.686047 .$$

Evaluăm $f(x_2) = -0.008949$.

Cum $|f(x_2)| = 0.008949 > \varepsilon$, trecem la pasul următor

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = -0.686047 + \frac{-0.008949}{2.411994} = -0.682340 .$$

Avem $f(x_3) = f(0.682340) = -0.000029$ şi

$|f(x_3)| = 0.000029 < \varepsilon = 0.0001$.

Cndiţia (1) fiind îndeplinită algoritmul se opreşte. Astfel valoarea aproximativă a rădăcinii este: $\alpha \cong -0.682340$

V.8. Metoda şirului Sturm pentru separarea rădăcinilor reale ale unei ecuaţii algebrice

Această metodă se aplică pentru determinarea numărului rădăcinilor reale ale unei ecuaţii algebrice, a semnelor acestora şi chiar pentru separarea lor.

Fie $P \in R[X], \quad P(X) = a_0 X^n + a_1 X^{n-1} + ... + a_{n-1} X + a_n$.

Considerăm mai întâi cazul în care polinomul *P* nu are rădăcini reale multiple, adică are rădăcini reale distincte, (fiecare cu ordinul de multiplicitate unu). Pentru orice astfel de polinom $P \in R[X]$ construim un şir de polinoame ataşate astfel:

$P_0(X) = P(X), \quad P_1(X) = P'(X), \quad P_k(X) = -R_k(X), \quad 2 \le k \le s$,
unde $P_{k-2}(X) = P_{k-1}(X) \cdot Q_{k-1}(X) + R_k(X)$.

Observaţia V.9. Acest şir de polinoame este cel pe care-l obţinem în algoritmul lui Euclid pentru aflarea unui cel mai mare divizor comun al polinoamelor *P*, P' notat (P, P'), cu observaţia că resturile se înmulţesc cu (-1).Cum *P* nu are radacini multiple rezultă că *P* si P' sunt prime între ele, deci rezultăcă $(P, P') = const \neq 0$. Această constantă este chiar P_s.

Observaţia V.10. Acest şir se numeşte şir Sturm asociat polinomului *P*. El nu este unicul şir Sturm asociat polinomului *P*.

Observaţia V.11. Polioamele din şirul Sturm construit pe baza algoritmului lui Euclid pot fi înlocuite de polinoame cu coeficienţii mai simpli, asociate în divizibilitate cu primele dar care să păstreze semnele coeficienţilor.

Definiţia V.4. Pentru polinomul P şi şirul Sturm asociat lui P definim funcţia $S: R \to N$, care asociază oricărui număr real x numărul de variaţii de semn din şirul de numere

$$P_0(x), P_1(x), ..., P_s(x).$$

Teorema V.2. Fie $a, b \in R$, $a<b$, astfel încât $P(a) \neq 0$ şi $P(b) \neq 0$. Avem:

1) $S(a) \geq S(b)$;

2) Numărul rădăcinilor reale ale ecuaţiei $P(x) = 0$, situate în (a,b) este egal cu $S(a) - S(b)$.

Observaţia V.12. Dacă P are rădăcini multiple atunci P şi P' au rădăcini commune şi astfel rezultă că $(P, P') = P_s$, cu $\operatorname{grad} P_s \geq 1$. În acest caz şirul Sturm asociat lui P se poate alege ca fiind format din:

$$F_0 = \frac{P_0}{P_s},\ F_1 = \frac{P_1}{P_s},\ \ldots\ldots\ldots,\ F_s = \frac{P_s}{P_s} = 1.$$

Consecinţa V.1. Dacă $P(0) \neq 0$ se poate determina numărul rădăcinilor pozitive şi negative, N_+, respectiv N_- astfel:

$N_+ = S(0) - S(\infty)$, reprezintă numărul rădăcinilor pozitive

$N_- = S(-\infty) - S(0)$, reprezintă numărul rădăcinilor negative.

Consecinţa V.2. Fie $\alpha, \beta \in R, \alpha < \beta$, astfel încât $P(\alpha) \neq 0$, $P(\beta) \neq 0$ şi, $S(\alpha) - S(\beta) = 1$, atunci în intervalul (α, β) polinomul P are o singură rădăcină reală.

Exemplul V.7. Fie ecuaţia $x^3 - 6x + 1 = 0$. Să se determine câte rădăcini pozitive şi câte negative are această ecuaţie şi să se determine intervale de separaţie pentru acestea.

Soluţie:

Construim un şir Sturm asociat.

$P_0(X) = X^3 - 6X + 1$, $P_1(X) = 3X^2 - 6$.

Observăm că putem înlocui polinomul $P_1(X)$ cu un altul, mai simplu, asociat în divizibilitate cu el: $X^2 - 2$

Obţinem: $P_2(X) = 4X - 1$, $P_3(X) = 31/16$.

Calculăm $S(-\infty)$.

Şirul de semne este: $-, +, -, +$.

Observăm că apar trei variaţii de semn, şi deci:

$S(-\infty) = 3$.

Şirul de semne corespunzător lui $+\infty$ este: $+, +, +, +$.

Astfel $S(\infty) = 0$.

Analog obţinem: $S(0) = 2$.

Deducem de aici că: N_-=1, N_+=2

Calculând alte valori ale funcţiei S găsim:

$S(-2) = 2, S(1) = 1, S(3) = 0$

Astfel $S(-2) - S(0) = 3 - 2 = 1$, deci există o unică rădăcină în intervalul $(-2,0)$.

Cum $S(0) - S(1) = 2 - 1 = 1$, deci există o unică rădăcină şi în intervalul $(0,1)$.

Analog $S(1) - S(3) = 1 - 0 = 1$, deci există o unică rădăcină în intervalul $(1,2)$.

Astfel au fost separate cele trei rădăcini reale ale polinomului.

Pentru a putea separa rădăcinile unei ecuaţii în care apare o funcţie derivabilă se poate utiliza şi metoda şirului Rolle pe care o vom aminti în continuare.

Fie $f : I \to R$, o funcţie derivabilă pe I. Metoda şirului Rolle se bazează pe o proprietate importantă a funcţiilor derivabile pe care

o amintim în continuare: între două zerouri consecutive ale derivatei, există cel mult un zerou al funcţiei.

Observaţia V.13. Cu ajutorul şirului lui Rolle, determinăm numărul rădăcinilor reale ale ale ecuaţiei $f(x)=0$, indicând şi intervalele în care se află aceste rădăcini.

Etapele formării şirului lui Rolle.

- Se stabileşte intervalul I de studiu, al ecuaţiei $f(x)=0$, funcţia $f: I \to R$, fiind presupusă derivabilă pe I.
- Se rezolvă ecuaţia: $f'(x)=0$ şi se ordonează (crescător), rădăcinile reale, din I, ale ecuaţiei: $x_m < ... < x_1 < x_2 < ... < x_M$.
- Se calculează, valorile funcţiei în aceste puncte, la care se adaugă limitele funcţiei, notate l_1, l_2 la capetele intrvalului I; obţinem şirul de valori:

$$l_1, f(x_m), ..., f(x_1), f(x_2), ..., f(x_M), l_2 .$$

- Datele obţinute se trec într-un tabel/tablou, pentru x, $f(x)$; şirul lui Rolle este şirul semnelor acestor valori (poate apărea şi zero) :
- Comentarii:

▪ Dacă $f(x_1)f(x_2)<0 \Rightarrow$ ecuaţia: $f(x)=0$, are în intervalul (x_1, x_2), o singură rădăcină reală.

▪ Dacă $f(x_1)f(x_2)>0 \Rightarrow$ ecuaţia: $f(x)=0$, nu are în intervalul (x_1, x_2), nici o soluţie reală.

▪ Dacă $f(x_i)=0$, atunci x_i este rădăcină multiplă, a ecuaţiei $f(x)=0$, şi în intervalul (x_{i-1}, x_i), (x_i, x_{i+1}), ecuaţia $f(x)=0$, nu mai are soluţii reale.

Observaţia V.14. Metoda şirului lui Rolle, este eficientă în cazul în care, există posibilitatea, rezolvării efective a ecuaţiei $f'(x)=0$.

Observaţia V.15. Numărând schimbările de semn şi zerourile, se determină, numărul de soluţii reale ale ecuaţiei date, şi intervalele în care se află acestea.

Exemplul V.9. Să se determine folosind şirul Rolle numărul de rădăcini reale ale ecuaţiei $x^5+2x-1=0$, şi să se separe acestea. Să se determine, folosind metoda bisecţiei, o valoare aproximativă pentru cea mai mare rădăcina a sa, efectuând 4 iteraţii.

Soluţie:

Se consideră $f:R\to R,\ f(x)=x^5+2x-1$

Calculăm limitele funcţiei la $\pm\infty$ şi obţinem:

$\lim\limits_{x\to\infty} f(x)=\infty,\ \lim\limits_{x\to-\infty} f(x)=-\infty$.

Derivata acestei funcţii este: $f':R\to R,\ f'(x)=5x^4+2>0$.

Deducem că funcţia este strict crescătoare pe R.

Calculăm valorile funcţiei în două puncte:

$f(0)=-1,\ f(1)=2$

Realizăm următorul tablou:

x	$-\infty$		0		1		$+\infty$
$f(x)$	$-\infty$	↗	-1	↗	2	↗	$+\infty$
$f'(x)$	$+$	$+$	$+$	$+$	$+$	$+$	$+$

Se observă astfel că ecuaţia admite o singură rădăcină reală situată în intervalul $(0,1)$.

Aplicând metoda bisecţiei găsim :

$a = 0, b = 1 \Rightarrow x = \frac{1}{2} = 0.5$, valoarea găsită la prima iteraţie

$f(0.5) = 0.031250$

$f(0.5) \cdot f(0) < 0 \Rightarrow b = 0.5$

$a = 0, b = 0.5 \Rightarrow x = \frac{0.5}{2} = 0.25$, valoarea găsită la a doua iteraţie

$f(0.25) = -0.499023$

$f(0.75) \cdot f(0) > 0 \Rightarrow a = 0.25$

$a = 0.25, b = 0.5 \Rightarrow x = \frac{0.25 + 0.5}{2} = 0.375$, valoarea găsită la a treia iteraţie.

$f(0.375) = -0.242584$

$f(0.375) \cdot f(0.25) > 0 \Rightarrow a = 0.375$

$a = 0.375, b = 0.5 \Rightarrow x = \frac{0.375 + 0.5}{2} = 0.4375$

Astfel soluţia aproximativă cerută este 0.4375.

VI. METODE NUMERICE PENTRU DETERMINAREA POLINOMULUI CARACTERISTIC, A VECTORILOR ŞI VALORILOR PROPRII

VI.1. POLINOM CARACTERISTIC, VECTORI ŞI VALORI PROPRII

Definiţia VI.1. Fie $A \in \mathrm{M_n(R)}$. Polinomul definit prin: $P_A(\lambda) = \det(\lambda I - A)$ se numeşte polinomul caracteristic al lui A.

Observaţia VI.1. Polinomul caracterisitic este un polinom de grad n, cu coeficienţi reali, şi cu coeficientul dominant egal cu unu.

Definiţia VI.2. Ecuaţia: $P_A(\lambda) = 0$ se numeşte ecuaţia caracteristică a lui A.

$\lambda^n + c_1\lambda^{n-1} + ... + c_{n-1}\lambda + c_n = 0$ (*)

Polinomul caracteristic are cel mult „n" rădăcini distincte (reale sau complexe).

(**) $(A - \lambda \mathrm{I}) \cdot \{\mathrm{x}\} = 0$ (vectorul nul din R^n); are şi soluţii nebanale.

Definiţia VI.3. Rădăcinile lui P_A se numesc valori proprii (sau valori caracteristice) ale matricei A.

Definiţia VI.4. Un vector nenul, $\{x\} \neq 0$, se numeşte vector porpriu asociat valorii proprii λ dacă satisface relaţia: $(\lambda I - A)\{x\} = 0$.

TeoremaVI.1. (Gerschgorin- localizarea valorilor proprii) Fie A o matrice pătratică de ordinul n. Dacă λ este o valoare proprie, arbitrară, a lui A atunci:

$$|\lambda - a_{ii}| \leq r_i \ , \ r_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \ , 1 \leq i \leq n \ .$$

VI.2. METODE NUMERICE PENTRU CALCULUL VALORILOR ŞI VECTORILOR PROPRII

1. Metoda minorilor diagonali

Dacă $A \in M_n(R)$ este de forma $A = (a_{ij}), 1 \le i, j \le n$, atunci $P_A(\lambda)$ se poate scrie astfel:

$$P_A(\lambda) = \lambda^n - \tau_1 \lambda^{n-1} + \tau_2 \lambda^{n-2} + ... + (-1)^n \tau_n ,$$

unde: $\tau_1 = \sum_{i=1}^{n} a_{ii}$ (τ_1 = suma minorilor diagonali de ordin 1)

$\tau_2 = \sum_{1 \le i < j \le n} \begin{vmatrix} a_{ii} & a_{ij} \\ a_{ji} & a_{jj} \end{vmatrix}$ (τ_2 =suma minorilor diagonali de ordin 2)

…

…

τ_n = det A

Exemplul VI.1. Să se calculeze, cu ajutorul minorilor diagonali, polinomul caracteristic al matricei:

$$A = \begin{pmatrix} 1 & 0 & -1 & 2 \\ 1 & -1 & 0 & 0 \\ 3 & -1 & 2 & 1 \\ 2 & -1 & 1 & 2 \end{pmatrix};$$

Soluţie: Calculăm minorii diagonali şi obţinem:

$\tau_1 = 1 - 1 + 2 + 2 = 4$

$$\tau_2 = \begin{vmatrix} 1 & 0 \\ 1 & -1 \end{vmatrix} + \begin{vmatrix} 1 & -1 \\ 3 & 2 \end{vmatrix} + \begin{vmatrix} 1 & 2 \\ 2 & 2 \end{vmatrix} + \begin{vmatrix} -1 & 0 \\ -1 & 2 \end{vmatrix} + \begin{vmatrix} -1 & 0 \\ -1 & 2 \end{vmatrix} + \begin{vmatrix} 2 & 1 \\ 1 & 2 \end{vmatrix} = 1$$

$$\tau_3 = \begin{vmatrix} 1 & 0 & -1 \\ 1 & -1 & 0 \\ 3 & -1 & 2 \end{vmatrix} + \begin{vmatrix} 1 & 0 & 2 \\ 1 & -1 & 0 \\ 2 & -1 & 2 \end{vmatrix} + \begin{vmatrix} 1 & -1 & 2 \\ 3 & 2 & 1 \\ 2 & 1 & 2 \end{vmatrix} + \begin{vmatrix} -1 & 0 & 0 \\ -1 & 2 & 1 \\ -1 & 1 & 2 \end{vmatrix} = -2$$

$\tau_4 = \det A = -6$

$$P_A(\lambda) = \lambda^4 - 4\lambda^3 + \lambda^2 + 2\lambda - 6$$

Exemplul VI.2. Să se calculeze, cu ajutorul minorilor diagonali, polinomul caracteristic al matricei:

$$A = \begin{pmatrix} 0 & -1 & 1 & 0 \\ 3 & 1 & 0 & 0 \\ 1 & -1 & 3 & 4 \\ 3 & -1 & 1 & 2 \end{pmatrix};$$

Soluţie: Calculăm minorii diagonali şi obţinem:

$\tau_1 = 0 + 1 + 3 + 2 = 6$

$$\tau_2 = \begin{vmatrix} 0 & -1 \\ 3 & 1 \end{vmatrix} + \begin{vmatrix} 0 & 1 \\ 1 & 3 \end{vmatrix} + \begin{vmatrix} 0 & 0 \\ 3 & 2 \end{vmatrix} + \begin{vmatrix} 1 & 0 \\ -1 & 3 \end{vmatrix} + \begin{vmatrix} 1 & 0 \\ -1 & 2 \end{vmatrix} + \begin{vmatrix} 3 & 4 \\ 1 & 2 \end{vmatrix} = 9$$

$$\tau_3 = \begin{vmatrix} 0 & -1 & 1 \\ 3 & 1 & 0 \\ 1 & -1 & 3 \end{vmatrix} + \begin{vmatrix} 0 & -1 & 0 \\ 3 & 1 & 0 \\ 3 & -1 & 2 \end{vmatrix} + \begin{vmatrix} 0 & 1 & 0 \\ 1 & 3 & 4 \\ 3 & 1 & 2 \end{vmatrix} + \begin{vmatrix} 1 & 0 & 0 \\ -1 & 3 & 4 \\ -1 & 1 & 2 \end{vmatrix} = 23$$

$\tau_4 = \det A = 22$

$$\Rightarrow P_A(\lambda) = \lambda^4 - 6\lambda^3 + 9\lambda^2 - 23\lambda + 22$$

2. Metoda Leverrier

Fie $P_A(\lambda) = \lambda^n - \tau_1\lambda^{n-1} + \tau_2\lambda^{n-2} + ... + (-1)^n\tau_n$.

În cadrul acestei metode, pentru a determina coeficienţii polinomului caracteristic parcurgem două etape:

1) Determinăm $s_k = Tr(A^k)$, $1 \le k \le n$

2) Coeficienţii τ_k, $1 \le k \le n$, se calculează recursiv cu relaţiile:

$$\tau_1 = s_1;\ \tau_2 = \frac{s_1\tau_1 - s_k}{2};$$

$$\tau_k = \frac{s_1 \cdot \tau_{k-1} - s_2 \cdot \tau_{k-2} + ... + (-1)^{k+1} \cdot s_k}{k};\ 2 \le k \le n.$$

Exemplul VI.3. Să se determine polinomul caracteristic al matricei următoare, folosind metoda Leverrier.

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 0 & -1 & 2 \\ 1 & 0 & 0 \end{pmatrix};$$

1) Determinăm mai întâi elementele $s_k = Tr(A^k)$.

$s_1 = Tr(A) = 1 - 1 + 0 = 0$

Calculăm puterile matricei *A* şi apoi urmele acestora. Avem:

$$A^2 = \begin{pmatrix} 2 & 0 & 3 \\ 2 & 1 & -2 \\ 1 & 1 & 1 \end{pmatrix} \qquad \Rightarrow s_2 = \text{Tr}\,(A^2) = 2 + 1 + 1 = 4$$

$$A^3 = \begin{pmatrix} 5 & 2 & 2 \\ 0 & 1 & 4 \\ 2 & 0 & 3 \end{pmatrix} \Rightarrow s_3 = \text{Tr}\,(A^3) = 5+1+3 = 9$$

2) Determinăm coeficienţii τ_k.

$$\tau_1 = s_1 = 0\,;\ \tau_2 = \frac{(s_1 \cdot \tau_1 - s_2)}{2} = \frac{(0 \cdot 0 - 4)}{2} = -2$$

$$\tau_3 = \frac{(s_1\tau_2 - s_2\tau_1 + s_3)}{3} = \frac{(0 \cdot (-2) - 4 \cdot 0 + 9)}{3} = 3$$

$$\Rightarrow\ P_A(\lambda) = \lambda^3 - 2\lambda - 3\,.$$

Exemplul VI.4. Să se determine polinomul caracteristic al matricei următoare folosind metoda Leverrier.

$$A = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 2 & 1 \\ 1 & 0 & 1 \end{pmatrix};$$

1) Determinăm mai întâi elementele $s_k = Tr(A^k)$.

$s_1 = Tr(A) = 2 + 1 + 1 = 4$

Calculăm puterile matricei A şi apoi urmele acestora. Avem:

$$A^2 = \begin{pmatrix} 1 & 3 & 3 \\ -2 & 3 & 2 \\ 2 & 1 & 2 \end{pmatrix} \qquad \Rightarrow s_2 = \text{Tr}\,(A^2) = 3 + 2 + 1 = 6$$

$$A^3 = \begin{pmatrix} 1 & 7 & 7 \\ -3 & 4 & 3 \\ 3 & 4 & 5 \end{pmatrix} \qquad \Rightarrow s_3 = \text{Tr}\,(A^3) = 1 + 4 + 5 = 10$$

2) Determinăm coeficienţii τ_k.

$$\tau_1 = s_1 = 4\,;\ \tau_2 = \frac{(s_1 \cdot \tau_1 - s_2)}{2} = \frac{(4 \cdot 4 - 6)}{2} = 5$$

$$\tau_3 = \frac{(s_1\tau_2 - s_2\tau_1 + s_3)}{3} = \frac{(4\cdot 5 - 6\cdot 4 + 10)}{3} = 2$$

$\Rightarrow P_A(\lambda) = \lambda^3 - 4\lambda^2 + 5\lambda - 2$. $P_A(\lambda) = (\lambda - 1)^2(\lambda - 2)$

Rădăcinile sale sunt: $\lambda_1 = \lambda_2 = 1$, $\lambda_3 = 2$.

Astfel au fost găsite două valori proprii distincte ale matricei considerate, una având ordinul de multiplicitate doi iar cealaltă unu.

3. Metoda Krylov

Fie $A \in M_n(R)$ şi $P_A(\lambda) = \lambda^n + c_1\lambda^{n-1} + ... + c_{n-1}\lambda + c_n$ olinomul ei caracteristic. Folosind teorema Cayley-Hamilton care afirmă că: matricea A verifică ecuaţia caracteristică (forma matriceală), putem scrie: $P_A(A) = 0_n$.

Deci: $A^n + c_1 \cdot A^{n-1} + ... + c_{n-1} \cdot A + c_n \cdot I_n = 0_n$.

Fie $y^{(0)} \in R^n$, oarecare, nenul. Prin înmulţirea relaţiei precedente la dreapta cu $y^{(0)}$ obţinem relaţia:

$$A^n \cdot y^{(0)} + c_1 A^{n-1} \cdot y^{(0)} + ... + c_x \cdot A \cdot y^{(0)} + c_n \cdot y^{(0)} = \overline{0}.$$

Relaţia de mai sus reprezintă un sistem de *n* ecuaţii liniare cu *n* necunoscute $c_1, c_2, ..., c_n$.

Dacă alegerea lui $y^{(0)} \in R^n$ conduce la obţinerea unui determinant nenul soluţia unică a acestuia reprezintă coeficienţii polinomului caracteristic. Dacă determinantul este nul, se reiau calculele cu un alt vector iniţial $y^{(0)}$.

Pentru simplitate se notează:

$A^k \cdot y^{(0)} = y^{(k)} = A\left(A^{k-1} y^{(0)}\right) = Ay^{(k-1)}$, $k = \overline{1,n} \Rightarrow$

sistemul se poate scrie astfel:

$$c_1 \cdot y^{(n-1)} + c_2 y^{(n-2)} + ... + c_{n-1} \cdot y^{(1)} + c_n \cdot y^{(0)} = -y^{(n)}.$$

Determinantul său este $\left|y^{(n-1)}\ y^{(n-2)}\ldots y^{(0)}\right|$ şi trebuie să fie nenul pentru a continua calculele.

Exemplul VI.3. Folosind metoda Krylov să se determine $P_A(\lambda)$, unde:

$$A=\begin{pmatrix} 1 & 1 & 1 \\ -1 & 2 & 1 \\ 1 & 0 & 1 \end{pmatrix}.$$

Alegem $y^{(0)}=\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$.

Găsim:

$$y^{(1)}=\begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix},\ y^{(2)}=\begin{pmatrix} 1 \\ -2 \\ 2 \end{pmatrix},\ y^{(3)}=\begin{pmatrix} 1 \\ -1 \\ 9 \end{pmatrix}.$$

Sistemul pe care îl obţinem în acest caz are determinantul nul, deci alegem alt vector iniţial $y^{(0)}$.

Fie $y^{(0)}=\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$. Găsim în acest caz:

$$y^{(1)}=\begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix},\ y^{(2)}=\begin{pmatrix} 3 \\ 3 \\ 1 \end{pmatrix},\ y^{(3)}=\begin{pmatrix} 7 \\ 4 \\ 4 \end{pmatrix}.$$

$$\Rightarrow \begin{cases} 3\cdot c_1 + c_2 = -7 \\ 3\cdot c_1 + 2c_2 + c_3 = -4 \\ c_1 = -4 \end{cases} \Rightarrow$$

Soluţia unică a acestui sistem este:

$c_1 = -4, c_2 = 5, c_3 = -2$

Astfel polinomul caracteristic ce se obţine este:

$$P_A(\lambda) = \lambda^3 - 4 \cdot \lambda^2 + 5 \cdot \lambda - 2 .$$

$P_A(\lambda) = (\lambda - 1)^2 (\lambda - 2)$

Rădăcinile sale sunt: $\lambda_1 = \lambda_2 = 1$, $\lambda_3 = 2$.

Observaţia VI.7. Dacă polinomul caracteristic are rădăcini dinstincte atunci, cu ajutorul metodei Krylov se pot determinarea vectorii proprii.

Notând

$$q_j(\lambda) = \frac{P_A(\lambda)}{\lambda - \lambda_j} = \lambda^{n-1} + q_{1j}\lambda^{n-2} + ... + q_{n-1j}, \; j = \overline{1,n},$$

atunci un vector propriu corespunzator valorii caracteristice λ_j este dat de relaţia:

$$y^{(n-1)} + q_{1j} y^{(n-2)} + ... + q_{n-1j} y^{(0)} \quad .$$

Exemplu VI.7. Să se calculeze, folosind metoda Krylov, polinomul caracteristic şi vectorii proprii ai matricei A, unde:

$$A = \begin{pmatrix} 2 & 1 & 1 \\ -1 & 2 & -1 \\ 1 & -1 & 2 \end{pmatrix}$$

Soluţie:

Fie $y^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \Rightarrow \; y^{(1)} = \begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix}, \quad y^{(2)} = \begin{pmatrix} 3 \\ -5 \\ 6 \end{pmatrix}, \; y^{(3)} = \begin{pmatrix} 7 \\ -19 \\ 20 \end{pmatrix}.$

Cum $\begin{vmatrix} 3 & 1 & 0 \\ -5 & -1 & 0 \\ 6 & 2 & 1 \end{vmatrix} = -3+5 = 2 \neq 0$,sistemul ce trebuie rezolvat este:

$$c_1\begin{pmatrix} 3 \\ -5 \\ 6 \end{pmatrix} + c_2\begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix} + c_3\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = -\begin{pmatrix} 7 \\ 19 \\ 20 \end{pmatrix}$$

Soluția acestuia este: $c_1 = -6, c_2 = 11, c_3 = -6$

Ecuaţia caracteristică a matricei A este:

$$\lambda^3 - 6\cdot\lambda^2 + 11\cdot\lambda - 6 = 0$$

Rădăcinile ei, adică valorile proprii, sunt: $\lambda_1 = 1, \quad \lambda_2 = 2, \quad \lambda_3 = 3$.

Fiind distincte două câte două vom determina vectorii proprii.

Pentru $\lambda_1 = 1$ avem:

$$q_1(\lambda) = \frac{P_A(\lambda)}{\lambda - \lambda_1} = \lambda^2 - 5\cdot\lambda + 6$$

$$y^{(2)} - 5\cdot y^{(1)} + 3\cdot y^{(0)} = \begin{pmatrix} 3 \\ -5 \\ 6 \end{pmatrix} - 5\cdot\begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix} + 6\cdot\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 \\ 0 \\ 2 \end{pmatrix} = 2\cdot\begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}$$

este un vector propriu corespunzător valorii proprii $\lambda_1 = 1$.

Pentru $\lambda_2 = 2 \Rightarrow q_2(\lambda) = \frac{P_A(\lambda)}{\lambda - \lambda_2} = \lambda^2 - 4\cdot\lambda + 3$.

Astfel un vector propriu corespunzător ei este:

$$y^{(2)} - 4\cdot y^{(1)} + 3\cdot y^{(0)} = \begin{pmatrix} 3 \\ -5 \\ 6 \end{pmatrix} - 4\cdot\begin{pmatrix} 1 \\ -1 \\ 2 \end{pmatrix} + 3\cdot\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}$$

Pentru $\lambda_3 = 3 \Rightarrow q_3(\lambda) = \frac{P_A(\lambda)}{\lambda - \lambda_3} = \lambda^2 - 3\cdot\lambda + 2$, deci un vector propriu corespunzător ei este:

$$y^{(2)} - 3 \cdot y^{(1)} + 2 \cdot y^{(0)} = \begin{pmatrix} 0 \\ -2 \\ 2 \end{pmatrix} = 2 \cdot \begin{pmatrix} 0 \\ -1 \\ 1 \end{pmatrix}.$$

4. Metoda Fadeev

Considerăm $P_A(\lambda) = \lambda^n + c_1\lambda^{n-1} + ... + c_{n-1}\lambda + c_n$

Coeficienţii polinomului caracteristic, cât şi inversa A^{-1} se pot obține din relațiile ce caracterizează algoritmul acestei metode:

$$A_1 = A, \quad c_1 = -Tr(A_1), \quad B_1 = A_1 + c_1 I_n$$

$$A_2 = AB_1, \quad c_2 = -\frac{1}{2} Tr(A_2), \quad B_2 = A_2 + c_2 I_n$$

.....

$$A_k = AB_{k-1}, \quad c_k = -\frac{1}{k} Tr(A_k), \quad B_k = A_k + c_k I_n$$

.....

$$\Rightarrow \begin{cases} B_n = O_n \\ A^{-1} = -\dfrac{1}{c_n} \cdot B_{n-1}, daca \;\; c_n \neq 0 \end{cases}$$

Exemplul VI.5. Folosind metoda Fadeev să se determine polinomul caracteristic, $P_A(\lambda)$, pentru matricea:

$$A = \begin{pmatrix} -1 & 2 & 0 \\ 1 & -1 & 2 \\ 3 & 1 & -3 \end{pmatrix},$$

şi dacă este inversibila şi inversa ei A^{-1}.

Soluţie:

Urmăm paşii din algoritm şi avem:

$$A_1 = A = \begin{pmatrix} -1 & 2 & 0 \\ 1 & -1 & 2 \\ 3 & 1 & -3 \end{pmatrix} \Rightarrow c_1 = 5, \quad B_1 = \begin{pmatrix} 4 & 2 & 0 \\ 1 & 4 & 2 \\ 3 & 1 & 2 \end{pmatrix}$$

$$A_2 = A \cdot B_1 = \begin{pmatrix} -2 & 6 & 4 \\ 9 & 0 & 2 \\ 4 & 13 & -4 \end{pmatrix} \Rightarrow c_2 = 3, \quad B_2 = \begin{pmatrix} 1 & 6 & 4 \\ 9 & 3 & 2 \\ 4 & 7 & -1 \end{pmatrix}$$

$$A_3 = A \cdot B_2 = \begin{pmatrix} 17 & 0 & 0 \\ 0 & 17 & 0 \\ 0 & 0 & 17 \end{pmatrix} \Rightarrow c_3 = -17, \quad B_3 = O_3 .$$

Astfel obţinem următoarea expresie pentru polinomul caracteristic

$$P_A(\lambda) = \lambda^3 + 5\lambda^2 + 3\lambda - 17 .$$

Deoarece $c_3 = -17 \neq 0$, se poate deduce şi inversa lui A.

Obţinem:

$$A^{-1} = \frac{1}{17} \cdot B_2 = \frac{1}{17} \begin{pmatrix} 1 & 6 & 4 \\ 9 & 3 & 2 \\ 4 & 7 & -1 \end{pmatrix}.$$

VII APROXIMAREA FUNCŢIILOR

VII.1. APROXIMAREA PRIN INTERPOLARE

Fie o funcţie $f:[a,b]\to R$, cunoscută numai într-un număr limitat de puncte (sau noduri): $x_0, x_1, \ldots, x_n$, ce poarta numele de *suportul interpolarii*, prin valorile: $f(x_0), f(x_1), \ldots, f(x_n)$, notate în general: $f_0, f_1, \ldots, f_n$.

De obicei aceste date sunt prezentate într-un *tabel de interpolare:*

x	x_0	x_1	...	x_n
$f(x)$	$f(x_0)$	$f(x_1)$	...	$f(x_n)$

Dacă punctele $x_0, x_1, \ldots, x_n$ sunt echidistante, se spune că dispunem de *o discretizare uniformă,* iar dacă nu sunt echidistante *discretizarea este neuniformă.*

Dacă presupunem că cele *n*+1 noduri reprezintă abscisele unor puncte din plan, iar $f_0, f_1, \ldots, f_n$ reprezintă ordonatele lor se pune problema determinării unei curbe, imaginea geometrică a unei funcţii reale, care trece prin acestea.

Vom aproxima comportarea funcţiei printr-un *polinom generalizat de interpolare*, de forma:

$$P_n(\mathrm{x}) = a_0\, u_0(\mathrm{x}) + a_1\, u_1(\mathrm{x}) + \ldots + a_n u_n(\mathrm{x})$$

în care funcţiile *liniar indepente*: $u_0(\mathrm{x}), u_1(\mathrm{x}), \ldots, u_n(\mathrm{x})$ sunt cunoscute şi constituie *baza interpolării.* Aceasta poate fi formată din funcţii simple: polinoame, funcţii trigonometrice, exponenţiale, etc.

Determinarea polinomului generalizat de interpolare presupune practic determinarea coeficienţilor $a_0, a_1, \ldots, a_n$. Se impune practic ca, pe suportul interpolării, polinomul de interpolare să coincidă cu funcţia f. Astfel se obţin relaţiile:

$$P_n(x_i) = f(x_i),\ i = \overline{0,n} \qquad (*)$$

Relaţiile (*) sunt cunoscute sub numele de *condiţii de interpolare.*

Se poate considera o abordare mai generală a aproximării unei funcţii, impunând condiţii de interpolare de forma:

$$P_n(x_i) = f(x_i),\quad P_n'(x_i) = f'(x_i), \ldots,\quad P_n^{(k)}(x_i) = f^{(k)}(x_i).$$

Condiţiile de interpolare (*) conduc la sistemul de ecuaţii liniare:

$$\sum_{i=0}^{n} a_k u_k(x_i) = f(x_i),\ i = \overline{0,n} \qquad .$$

1.1. INTERPOLARE POLINOMIALĂ. POLINOMUL DE INTERPOLARE LAGRANGE

Dacă se alege baza polinomială:

$$u_0(x)=1 \quad u_1(x)=x \ \ldots \ u_n(x)=x^n$$

sistemul determinat de condiţiile de interpolare devine:

$$\sum_{k=0}^{n} a_k \cdot x_i^k = f(x_i) \ .$$

El reprezintă un sistem linar de n+1 ecuaţii cu n+1 necunoscute caracterizat de un detrminant Vandermonde nenul, dacă punctele x_i sunt distincte, caz în care sistemul este compatibil determinat, având soluţie unică, ceea ce implică un polinom de interpolare unic.

Teorema 1. Date fiind n+1 puncte distincte $x_0, x_1, \ldots, x_n$ şi n+1 valori reale arbitrare $f_0, f_1, ..., f_n$, există şi este unic un polinom de interpolare corespunzător tabelului de interpolare construit pe baza lor.

Polinomul general de interpolare, de grad cel mult *n*, se poate obţine sub forma:

$$P(x)=\sum_{k=0}^{n} f(x_k)\cdot\prod_{\substack{i=0\\ i\neq k}}^{n}\frac{x-x_i}{x_k-x_i}=\sum_{k=0}^{n} f_k l_k(x)$$

(**)

şi poartă numele de polinom de interpolare Lagrange asociat tabelului de interpoale dat, notat $L_n(x)$, iar polinoamele $l_k(x)=\prod_{i=0\ j\neq k}^{n}\frac{x-x_i}{x_k-x_i}$ ce apar în formulă se numesc polinoame fundamentale de interpolare Lagrange.

Ele au proprietatea:

$$l_k(x_i)=\begin{cases}1, & i=k\\ 0, & i\neq k\end{cases}$$

Exemplul 1. Să se determine polinomul de interpolare Lagrange pentru o funcţie *f* ştiind că suportul interpolării este: $x_0=-1, x_1=0, x_2=1, x_3=3$, iar valorile funcţiei în aceste puncte sunt: 7, 1, 2, 1. Să se determine apoi valoarea acestuia în punctul 2.

Soluţie:

Polinomul de interpolare Lagrange este dat de:

$$P(x)=\sum_{k=0}^{3} f(x_k)\cdot l_k(x).$$

Notăm $\omega(x)=(x-x_0)(x-x_1)(x-x_2)(x-x_3)$.

Avem: $\omega(x)=(x+1)x(x-1)(x-3)$.

Notăm: $\omega_k(x)=\dfrac{\omega(x)}{x-x_k}$, $k=\overline{0,3}$. Obţinem următoarele relaţii:

$$\omega_0(x)=x(x-1)(x-3),\ \omega_1(x)=(x+1)(x-1)(x-3),$$
$$\omega_2(x)=(x+1)x(x-3),\ \omega_3(x)=(x+1)x(x-1).$$

Cu ajutorul lor construim polinoamele fundamentale Lagrange date de relaţia: $l_k(x)=\dfrac{\omega_k(x)}{\omega_k(x_k)}$, $k=\overline{0,3}$.

Obţinem deci:

$$l_0(x)=\frac{x(x-1)(x-3)}{-8},$$

$$l_1(x)=\frac{(x+1)(x-1)(x-3)}{3},$$

$$l_2(x)=\frac{(x+1)x(x-3)}{-4},$$

$$l_3(x)=\frac{(x+1)x(x-1)}{24}.$$

Deducem deci următoarea formă pentru polinomul de interpolare Lagrange:

$$L_3(x)=7\frac{x(x-1)(x-3)}{-8}+\frac{(x+1)(x-1)(x-3)}{3}+2\frac{(x+1)x(x-3)}{-4}+\frac{(x+1)x(x-1)}{24}$$

Se obţine apoi valoarea polinomului în punctul 2: $L_3(2)=4$.

Exemplul 2. Considerând variaţia densităţii unui material în intervalul de temperatura $t = 60^o C ... 120^o C$ din tabelul următor să se determine densitatea acestuia la 70^o C, folosindu-se un polinom Lagrange de gradul 2.

$t\left[^o C\right]$	60	80	100	120
$\rho\left[Kg/m^3\right]$	435,60	413,40	402,65	398.50

Soluţie:

Folosind toate datele din tebelul de interpolare dat s-ar obţine un polinom Lagrange de grad 3. Polinomul cerut trebuie să aibă gradul doi, deci selectăm un suport de interpolare mai restrâns: ca suport de interpolare punctele $t_1 = 60$, $t_2 = 80$, $t_3 = 100$ şi valorile corespuzătoare ale funcţiei densitate.

Se obţin polinoamele fundamentale Lagrange:

$$l_1(t) = \frac{(t-t_2)(t-t_3)}{(t_1-t_2)(t_1-t_3)} = \frac{(t-80)(t-100)}{(60-80)(60-100)} = \frac{t^2-180t+8000}{800}$$

$$l_2(t) = \frac{(t-t_1)(t-t_3)}{(t_2-t_1)(t_1-t_3)} = \frac{(t-60)(t-100)}{(80-60)(80-100)} = \frac{t^2-160t+6000}{-400}$$

$$l_3(t) = \frac{(t-t_1)(t-t_2)}{(t_3-t_1)(t_3-t_2)} = \frac{(t-60)(t-80)}{(100-60)(100-80)} = \frac{t^2-140t+4800}{800}$$

Astfel polinomul de interpolare Lagrange este :

$$L(t) = 435{,}60 \cdot l_1(t) + 413.40 \cdot l_2(t) + 402{,}65 \cdot l_3(t).$$

Pentru $t = 70^o C$ se obţine următoarea valoare : $L(70) = 423{,}06875$.

1.2. DIFERENŢE DIVIZATE

O altă metodă de a obține polinomul de interpolare corespunzător unui tabel de intepolare dat este cea bazată pe așa-numitele diferențe divizate, care reprezintă în fapt niște notații asociate unor rapoarte construite după anumite regului cu datele din tabelul de interpolare.

Diferenţele divizate pot fi introduse prin recurenţă, cu ajutorul definiţiilor următoare:

- Diferenţele divizate de ordin zero, notate $F_0[\ ]$ coincid cu valorile funcţiei în nodurile date. Există astfel *n*+1 diferenţe divizate de ordinul zero pentru datele din tabloul de interpolare considerat;

$$F_0[x_i] = f(x_i)$$

- Diferenţele divizate de ordinul unu, notate cu $F_1[\ ]$, se definesc prin egalitatea:

$$F_1[x_i, x_{i+1}] = \frac{F_0[x_i] - F_0[x_{i+1}]}{x_i - x_{i+1}}$$

şi sunt în număr de *n*: $F_1[x_0, x_1], F_1[x_1, x_2], \ldots, F_1[x_{n-1}, x_n]$;

……..

- Diferenţele divizate de ordinal *p* se definesc cu ajutorul diferenţelor divizate de ordin *p*-1 prin relaţia:

$$F_p[x_0, x_1, \ldots, x_p] = \frac{F_{p-1}[x_0, \ldots, x_{p-1}] - F_{p-1}[x_1, \ldots, x_p]}{x_0 - x_p};$$

.....

- Există o singură diferenţă divizată de ordinul *n* dată de:

$$F_n[x_0, x_1, \ldots, x_n] = \frac{F_{n-1}[x_0, \ldots, x_{n-1}] - F_{n-1}[x_1, \ldots, x_n]}{x_0 - x_n}$$

Pentru a simplifica evaluarea diferenţelor divizate se poate utiliza aşezarea acestora într-un tabel de forma următoare:

x_i	F_0	F_1	F_2	...	F_n
x_0	$F_0[x_0] = f_0$	$F_1[x_0, x_1]$	$F_2[x_0, x_1, x_2]$		$F_n[x_0, .., x_n]$
x_1	$F_0[x_1] = f_1$	$F_1[x_1, x_2]$	$F_2[x_1, x_2, x_3]$		-
x_2	$F_0[x_2] = f_2$	$F_1[x_2, x_3]$	$F_2[x_2, x_3, x_4]$		-
...	...	...	...		-
x_{n-2}	$F_0[x_{n-2}] = f_{n-2}$	$F_1[x_{n-2}, x_{n-1}]$	$F_2[x_{n-2}, x_{n-1}, x_n]$		-

x_{n-1}	$F_0[x_{n-1}]=f_{n-1}$	$F_1[x_{n-1},x_1]$	-		-
x_n	$F_0[x_n]=f_n$	-	-		-

Notând cu d_{ij} elementele acestui tablou observăm că sunt valabile relaţiile:

$$d_{i0}=f_i,\ i=\overline{0,n},$$

$$d_{ij}=\frac{d_{i+1,j-1}-d_{i,j-1}}{x_{j+i-1}-x_i},\ j=\overline{1,n};i=\overline{0,n-j}.$$

O altă formă a polinomului de interpolare de gred cel mul n, se poate obține cu ajutorul ediferenţelor divizate şi poartă numele de polinom Newton de interpolare, notat cu $N_n(x)$:

$$N_n(x)=f(x_0)+(x-x_0)F_1[x_0,x_1]+(x-x_0)(x-x_1)F_2[x_0,x_1,x_2]...+(x-x_0)..(x-x_{n-1})F_n[x_0,...,x_n]$$

Exemplul 3 Folosind diferenţele divizate să se determine polinomul Newton de interpolare ce interpolează datele din tabelul următor:

x	-1	1	2	3
f	4	2	1	-2

Soluţie:

Construim tabelule diferenţelor divizate şi apoi cu ajutorul elementelor de pe prima linie a tabelului polinomul Newton de interpolare.

x	F_0	F_1	F_2	F_3
-1	4	-1	0	-1/4
1	2	-1	-1	
2	1	-3		
3	-2			

Utilizând relaţia (14) obţinem:

$$N_3(x) = F_0(x_0) + (x - x_0)F_1[x_0, x_1] + (x - x_0)(x - x_1)F_2[x_0, x_1, x_2] + \\ + (x - x_0)(x - x_1)(x - x_2)F_3[x_0, ..., x_3]$$

Astfel polinomul Newton de interpolare este:

$$N_3(x) = 4 + (x+1)(-1) + (x+1)(x-1)(x-2)\left(-\frac{1}{4}\right).$$

Exemplul 4.

Să se determine polinomul Newton de interpolare de grad cel mult 3, ce interpolează funcţia $f : R \to R,\ f(x) = 2x^2 + 3x - 1$ în nodurile $x = 0,1,2,3$. Care este gradul polinomului găsit?

x	0	1	2	3
f	-1	4	13	26

Soluţie:

Construim tabelule diferenţelor divizate şi apoi polinomul Newton de interpolare.

x	F_0	F_1	F_2	F_3
0	-1	5	2	0
1	4	9	2	
2	13	13		
3	26			

$$N_3(x) = F_0(x_0) + (x - x_0)F_1[x_0, x_1] + (x - x_0)(x - x_1)F_2[x_0, x_1, x_2] + (x - x_0)(x - x_1)(x$$

Astfel obţinem:

$$N_3(x) = -1 + 5x + 2x(x-1).$$

Remarcăm că el are gradul doi şi coincide cu funcţia dată:

$$N_3(x) = f(x)$$

În ceea ce priveşte eroarea pe care o introducem aproximând o funcţie prin polinomul de interpolare de grad cel mult *n,* se poate demonstra că în cazul unei funcţii de clasă $C^{n+1}([a,b])$ aceasta este dată de:

$$E_n(x) = f(x) - P_n(x) = f^{(n+1)}(z)\frac{\omega(x)}{(n+1)!}, \text{ cu } z \in (a,b).$$

Dacă în plus nodurile din intervalul $[a,b]$sunt echidistante, atunci

$$|E_n(x)| = |f(x) - P_n(x)| \le \frac{1}{4(n+1)} M \left(\frac{b-a}{n} \right)^{n+1}, \quad \text{unde}$$

$M = \sup_{[a,b]} |f^{(n+1)}(x)|$.

Exemplul 5.

Să se determine cât de mare este eroarea pe care o introducem când aproximăm funcția $f(x) = \sin x$ pe intervalul $[0,2]$cu un polinom de interpolare corespunzător unui tabel de intrpolare cu 21 noduri echidistante.

Avem $n = 20$, $a = 0, b = 2$, $|f^{(21)}(x)| = |\cos x| \le 1 = M$.

Astfel obținem o eroare foarte mică:

$$|E_{20}(x)| = |f(x) - P_{20}(x)| \le \frac{1}{4 \cdot 21} \left(\frac{2-0}{20} \right)^{21} \cong 0{,}012 \cdot 10^{-21}$$

1.3. DIFERENȚE FINITE

Diferenţele finite sunt utile în evaluarea numerică a derivatelor şi în formulele de interpolare. Ele sunt de mai multe feluri şi se aplică unor funcţii definite în puncte echidistante: $x_i = x_0 + ih$:

-diferenţe progresive:

$$\Delta f(x_i) = \frac{f(x_i + h) - f(x_i)}{h} = \frac{f(x_{i+1}) - f(x_i)}{h}$$

$$\Delta^k f(x_i) = \frac{\Delta^{k-1} f(x_{i+1}) - \Delta^{k-1} f(x_i)}{h}$$

-diferenţe regresive:

$$\nabla f(x_i) = \frac{f(x_i) - f(x_i - h)}{h} = \frac{f(x_i) - f(x_{i-1})}{h}$$

$$\nabla^k f(x_i) = \frac{\nabla^{k-1} f(x_i) - \nabla^{k-1} f(x_{i-1})}{h} \qquad (15)$$

-diferenţe centrate:

$$\delta f(x_i) = \frac{f(x_i + \frac{h}{2}) - f(x_i - \frac{h}{2})}{h} = \frac{f(x_{i+\frac{1}{2}}) - f(x_{i-\frac{1}{2}})}{h}$$

$$\delta^k f(x_i) = \frac{\delta^{k-1} f(x_{i+\frac{1}{2}}) - \delta^{k-1} f(x_{i-\frac{1}{2}})}{h}.$$

Pentru a trece de la diferenţele finite la diferenţele divizate se foloseşte relaţia:

$$\Delta^n f(x_i) = \nabla^n f(x_{i+n}) = \delta^n f(_{i+n/2}) = = n! h^n F^n [x_i, x_{i+1}, ..., x_{i+n}].$$

Astfel, cu ajutorul lor pot fi construite polinoamele de interpolare, în cazul funcţiilor definite pe un suport de interpolare cu noduri echidistante.

Fie deci funcţia f cunoscută printr-un tabel de interpolare în care abscisele x_i sunt echidistante.

Dorim să evaluăm funcţia în punctul intermediar x, $(x \neq x_i)$. Vom considera în mod simplificator că acest punct poate fi situat:

-la începutul tabloului $x_0 < x < x_1$

-la sfârşitul tabloului $x_{n-1} < x < x_n$.

Dacă $x_i < x < x_{i+1}$ cu $i \neq 0$ şi $i \neq n-1$ vom neglija un număr de puncte de la începutul sau sfîrşitul tabloului, pentru a ne situa într-unul din cazurile de mai sus.

Formulele de interpolare construite cu ajutorul diferenţelor finite poartă numele de formulele Newton – Gregory.

Prima formulă Newton-Gregory realizează interpolare la început de tablou, adică consideră: $x = x_0 + uh$ cu 0< u < 1.

Pornind de la expresia polinomului Newton de interpolare în punctul x, şi exprimând diferenţele divizate cu ajutorul diferenţelor finite progresive: $F_k[x_0, x_1, ..., x_k] = \frac{\Delta^k f(x_0)}{k! \cdot h^k}$ se obţine:

$$P_1(x) = P_1(x + u \cdot h) = p_1(u) =$$
$$f(x_0) + \frac{x - x_0}{1! \cdot h} \cdot \Delta f(x_0) + \frac{(x - x_0) \cdot (x - x_1)}{2! \cdot h^2} \cdot \Delta^2 f(x_0) + ...$$
$$+ \frac{(x - x_0) \cdot (x - x_1) ... (x - x_{n-1})}{n! \cdot h^n} \cdot \Delta^n f(x_0)$$

Ţinând cont de faptul că:
$x - x_k = x - x_0 - (x_k - x_0) = uh - kh = (u - k)h$, obţinem:

$$p_1(u) = f_0 + \frac{u}{1!} \cdot \Delta f_0 + \frac{u \cdot (u-1)}{2!} \cdot \Delta^2 f_0 + ... + \frac{u \cdot (u-1) ... (u-n+1)}{n!} \cdot \Delta^n f_0$$

.

Următoarea formulă de interpolare Newton-Gregory se referă la interpolarea la sfârşit de tablou şi se obţin exprimând diferenţele divizate prin diferenţe regresive. Astfel dacă se ia: $x = x_n - u \cdot h$, 0< u <1 , şi se parcurg aceleaşi etape, se obţine:

$$p_2(u) = f_0 - \frac{u}{1!} \cdot \nabla f_n + \frac{u(u-1)}{2!} \cdot \nabla^2 f_n + \ldots + (-1)^n \frac{u(u-1)\ldots(u-n+1)}{n!} \cdot \nabla^n f_n$$

.

1.3. INTERPOLARE CU FUNCŢII SPLINE

Polinomul de interpolare Lagrange prezintă un grad înalt pentru un număr mare de puncte. Aceasta ne determină uneori să alegem polinoame de interpolare de grad mic, valabile pe subintervale ale intervalului $[a,b]$. În astfel de situații folosim funcțiile spline pentru aproximare.

Definiția 1.

O funcție S se numește funcție spline de grad k dacă ea îndeplinește condițiile:

Este definită pe $[a,b]$,

Derivatele ei de ordin r sunt continue pe $[a,b]$, pentru $0 \le r \le k-1$

Restricțiile ei la subintervalele $[x_i, x_{i+1}]$, $i = \overline{1, n-1}$ sunt polinoame de grad cel mult *k*.

Funcţii spline liniare

Fie $n+1$ puncte: $x_0 < x_1 < \ldots < x_n$ în care se cunosc valorile funcţiei: $f(x_0), f(x_1), ..., f(x_n)$, vom considera în acest caz funcţii de interpolare liniare locale, pe subintervalele:

$$[x_0, x_1], [x_1, x_2], ..., [x_i, x_{i+1}], ..., [x_{n-1}, x_n]$$

de forma:

$$s_i(x) = a_i x + b_i,\ i = 0...n-1\ ,$$

în care cei $2n$ parametri a_i şi b_i se determină impunând satisfacerea condiţiilor interpolare:

$$s_i(x_i) = f(x_i) \quad i = 0...n-1 \qquad\qquad s_{n-1}(x_n) = f(x_n)$$

şi a condiţiilor de racordare în punctele interioare:

$$s_i(x_{i+1}) = s_{i+1}(x_{i+1}) \quad i = 0...n-2$$

Interpolarea liniară ne oferă o funcţie continuă şi derivabilă pe porţiuni, dar care prezintă totuşi dezavantajul discontinuităţii derivatelor în punctele interioare.

Condiţiile de interpolare ne dau coeficienţii:

$$a_i = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}, i = 0...n-1$$

$$b_i = \frac{x_{i+1} f(x_i) - x_i f(x_{i+1})}{x_{i+1} - x_i}.$$

Observăm că pe fiecare din subintervalele formate se construiesc practic segmente de drepte ce unesc punctele de

coordonate $(x_i, f(x_i))$ și $(x_{i+1}, f(x_{i+1}))$ și astfel se realizează o aproximare a curbei reale printr-o linie poligonală.

Funcția spline se poate prelungi și în afara intervalului $[a,b]$ astfel:

$$s(x) = \begin{cases} s_1(x), x < a \\ s_{n-1}(x), x > b \end{cases}$$

Funcții spline pătratice

Dacă în cazul funcțiilor spline liniare pe fiecare subinterval format am construit practic segmente, în cazul funcțiilor spline ptratice se vor construi arce de parabole prin intermediul funcțiilor de gradul doi.. Astfel pe fiecare subinterval $[x_i, x_{i+1}]$ se alege

$s_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2$, cu a_i, b_i, c_i coeficienți ce urmează a fi determinați.

Astfel funcțiile spline trebuie să îndeplinească relațiile:

$$\begin{aligned} &s_i(x) = f_i \\ &s_i(x_{i+1}) = f_{i+1} \\ &s_i'(x_i) = d_i \\ &s_i'(x_{i+1}) = d_{i+1} \end{aligned},$$

cu d_i construite recursiv cu ajutorul relației de ordinul unu:

$$d_{i+1} = -d_i + 2\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i},\ i = \overline{1, n-1},\ \text{iar } d_1 \text{ arbitrar.}$$

Se obțin relațiile:

$$s_i(x) = f(x_i) + d_i(x - x_i) + \frac{d_{i+1} - d_i}{2(x_{i+1} - x_i)}(x - x_i)^2$$

Cu ajutorul funcțiilor spline pătratice de mai sus se obține o funcție de aproximare care are derviata contină pe $[a,b]$.

Exemple:

Să se determine funcția spline pătratică ce interpolează datele din tabelul:

x	0	1	2	3
f	1	4	8	10

Construim funcțiile spline calculând mai întâi coeficienții d_i pornind de la $d_1 = 0$.

Astfel obținem:

$$d_2 = 2\frac{4-1}{1} = 6,\ d_3 = -6 + 2\frac{8-4}{1} = 2,\ d_4 = -2 + 2\frac{10-8}{1} = 2.$$

Funcțiile spline pătratice atașate subintervalelor formate sunt:

$$s_1(x) = 1 + \frac{6}{2}x^2 = 1 + 3x^2,$$

$$s_2(x) = 4 + 6(x-1) + \frac{2-6}{2}(x-1)^2 = -2x^2 + 10x - 4$$

$$s_3(x) = 8 + 2(x-2) = 2x + 4.$$

Astfel funcția spline corespunzătoare tabelului dat are următoarea expresie:

$$S(x)=\begin{cases}1+3x^2,\ x\in[0,1]\\ -2x^2+10x-4,\ x\in[1,2]\\ 2x+4,\ x\in[2,3]\end{cases}$$

Se observă ca ea este nu numai continuă, ci și derivabilă, iar derivata sa este continuă pe $[0,3]$.

Funcții spline cubice

Prin alegerea unor funcţii de interpolare de gradul 3 se poate realiza o interpolare care presupune şi fixarea valorii derivatelor pe suportul interpolării:

$$f'(x_0), f'(x_1),..., f'(x_n)$$

O funcţie spline cubică se poate exprima astfel:

$$S_i(x)=a_i+b_i(x-x_i)+c_i(x-x_i)^2+d_i(x-x_i)^3$$

Cei $4n$ coeficienţi a_i, b_i, c_i, d_i ai funcţiilor spline se determină impunând $2n+2$ condiţii de interpolare:

$$S_i(x_i)=f(x_i)\ ,\qquad S_i'(x_i)=f'(x_i)\ \text{cu}\ i=0....n$$

şi $2n-2$ condiţii de racordare, asigurând continuitatea şi derivabilitatea funcţiilor de interpolare vecine în punctele interioare:

$$S_i(x_{i+1})=S_{i+1}(x_{i+1})\ ,\quad S_i'(x_{i+1})=S_{i+1}'(x_{i+1})\ \text{cu}\ i=\overline{0,n-2}$$

.

2. APROXIMAREA ÎN SENSUL CELOR MAI MICI PĂTRATE

În paragrafele precedente s-au construit fucții care aproximează datele dintr-un tabel prin interpolare, adică funcții ce trec prin toate punctele din plan corespunzătoare tabelului dat.

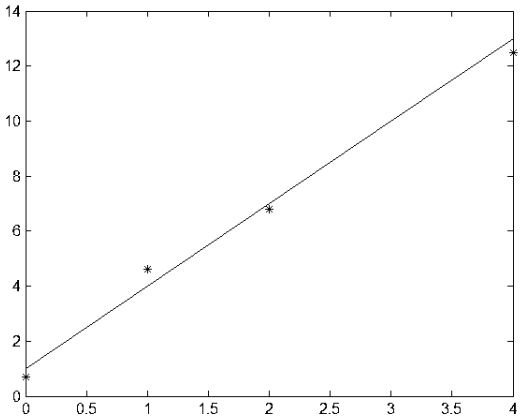
Uneori se pune problema găsirii unei funcții care să reprezinte cel mai bine datele din tabel, dar care să nu treacă neapărat prin toate punctele date.

Primul tip de aproximare se folosește când datele din tabel sut date cu o mare acuratețe, iar cel de-al doilea în cazul în care în determinarea lor pot să apară mici erori (de exemplu de măsurare).

Cea mai des utilizată tehnică pentru a găsi o aproximantă ca re se potrivește cel mai bine datelor dintr-un tabel este metoda celor mai mici pătrate.

Primul pas ce trebuie făcut pentru a aplica această metodă este reprezentarea punctelor în plan, pentru a putea vizualiza forma norului de puncte și pentru a intui astfel ce model de funcție se potrivește pentru aproximantă.

În cazul în care se observă pe grafic că norul de puncte urmează forma unei drepte, precum în figura următoare, se folosește pentru aproximantă un model liniar: $f(x) = ax + b$.



Erorile care apar în fiecare nod, denumite și reziduuri, $e_i = f(x_i) - y_i$, trebuie minimizate.

În cazul metodei celor mai mici pătrate coeficienții *a* și *b* se determină impunând condiția ca suma pătratelor erorilor ce apar să fie minimă. De aceea metoda folosită se numește metoda celor ai mici pătrate.

Astfel se formează suma: $E(a,b) = \sum_{i=1}^{N} \left((ax_i + b) - y_i\right)^2$ căreia vrem să-i determinăm minimul. Vom calcula mai întâi punctele ei staționare.

$$\begin{cases} \dfrac{\partial E}{\partial a} = 0 \\ \dfrac{\partial E}{\partial b} = 0 \end{cases}, \text{ cu } \begin{cases} \dfrac{\partial E}{\partial a} = 2\sum_{i=1}^{N}\left((ax_i + b) - y_i\right)x_i \\ \dfrac{\partial E}{\partial b} = 2\sum_{i=1}^{N}\left((ax_i + b) - y_i\right) \end{cases}.$$

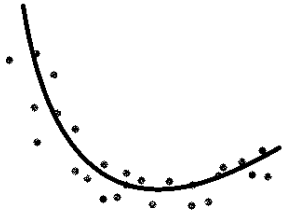
Se obține astfel sistemul:

$$\begin{cases} a\sum_{i=1}^{N} x_i^2 + b\sum_{i=1}^{N} x_i - \sum_{i=1}^{N} x_i y_i = 0 \\ a\sum_{i=1}^{N} x_i + bN - \sum_{i=1}^{N} y_i = 0 \end{cases}.$$

Coeficienții din ecuația funcției de gradul întâi care aproximează cel mai bine datele în sensul celor mai mici pătrate sunt:

$$a = \frac{\left(\sum_{i=1}^{N} x_i\right)\left(\sum_{i=1}^{N} y_i\right) - N\sum_{i=1}^{N} x_i y_i}{\left(\sum_{i=1}^{N} x_i\right)^2 - N\sum_{i=1}^{N} x_i^2}, \quad b = \frac{\sum_{i=1}^{N} y_i - a\sum_{i=1}^{N} x_i}{N}.$$

În cadrul aproximărilor pe care le facem locul funcției de gradul întâi poate fi luat de orice altă funcție polinomială și nu numai. În cazul în care se observă pe grafic că norul de puncte urmează forma unei parabole, precum în figura următoare, se folosește o funcție de gradul doi.



Pentru exprimarea unui model parabolic de forma: $y = a+bx+cx^2$, cu ajutorul metodei celor mai mici pătrate, construim mai întâi funcția E, ce însumează pătratele reziduurilor care apar : $E(a,b,c)=\sum_{i=1}^{N}\left(\left(cx_i^2+bx_i+a\right)-y_i\right)^2$.

Coeficienţii a, b, c se obţin din condiţia ca funcţia E să admită un minimum, adică:

$$\begin{cases} \dfrac{\partial E}{\partial a} = 0, \\ \dfrac{\partial E}{\partial b} = 0, \\ \dfrac{\partial E}{\partial c} = 0. \end{cases}$$

Calculând derivatele parţiale obţinem expresiile:

$$\begin{cases} \dfrac{\partial E}{\partial a} = 2\sum_{i=1}^{N}\left(\left(cx_i^2+bx_i+a\right)-y_i\right), \\ \dfrac{\partial E}{\partial b} = 2\sum_{i=1}^{N}\left(\left(cx_i^2+bx_i+a\right)-y_i\right)x_i, \\ \dfrac{\partial E}{\partial c} = 2\sum_{i=1}^{N}\left(\left(cx_i^2+bx_i+a\right)-y_i\right)x_i^2. \end{cases}$$

Simplificând scrierea sumelor obținem sistemul următor:

$$\begin{cases} na + b\sum x_i + c\sum x_i^2 = \sum y_i \\ a\sum x_i + b\sum x_i^2 + c\sum x_i^3 = \sum x_i y_i \\ a\sum x_i^2 + b\sum x_i^3 + c\sum x_i^4 = \sum x_i^2 y_i \end{cases}$$

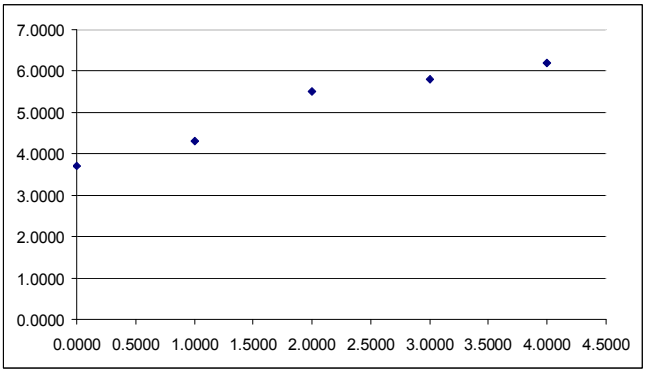
Rezolvând acest sistem obținem practic parabla căutată.

Exemple:

1. Să se determine curba care aproximează cel mai bine în sensul celor mai mici pătrate datele din tabelul următor:

x	0	1	2	3	4
f	3,7	4,3	5,5	5,8	6,2

Reprezentând grafic punctele observăm că ele se așează în jurul unei drepte



Astfel vom determina o aproximantă de tipul unei funcții de gradul întâi: $f(x)=ax+b$, cu a și b dați de relațiile:

$$a=\frac{\left(\sum_{i=1}^{N}x_i\right)\left(\sum_{i=1}^{N}y_i\right)-N\sum_{i=1}^{N}x_iy_i}{\left(\sum_{i=1}^{N}x_i\right)^2-N\sum_{i=1}^{N}x_i^2},\ b=\frac{\sum_{i=1}^{N}y_i-a\sum_{i=1}^{N}x_i}{N},$$

unde N=5.

Pentru ușurință calculele sunt trecute într-un tabel:

i	x_i	y_i	x_iy_i	x_i^2
1	0.0000	3.7000	0.0000	0.0000
2	1.0000	4.3000	4.3000	1.0000
3	2.0000	5.5000	11.0000	4.0000
4	3.0000	5.8000	17.4000	9.0000
5	4.0000	6.2000	24.8000	16.0000
Σ	10.0000	25.5000	57.5000	30.0000

Obținem: $a=\frac{10\cdot 25.5-5\cdot 57.5}{10^2-5\cdot 30}=0.75$, $b=\frac{25.5-0.75\cdot 10}{5}=3.7$

Astfel funcția care aproximează cel mai bine datele din tabel are expresia: $f(x)=0.75x+3.7$.

VIII. EVALUAREA NUMERICĂ A INTEGRALELOR

Prin evaluarea numerică a integralelor înțelegem calculul aproximativ al integralelor definite, $\int_a^b f(x)dx$, cu ajutorul unor formule care folosesc valorile funcției în anumite noduri din intervalul $[a,b]$. Necesitatea evaluării numerice a integralelor apare în specail atunci când primitivele funcțiior de integrat sunt dificil de aflat sau când acestea sunt date tabelar. Metodele cel mai des utilizate pentru evaluarea numerică a integralelor sunt: metoda trapezului, metoda Simpson şi metoda Newton.

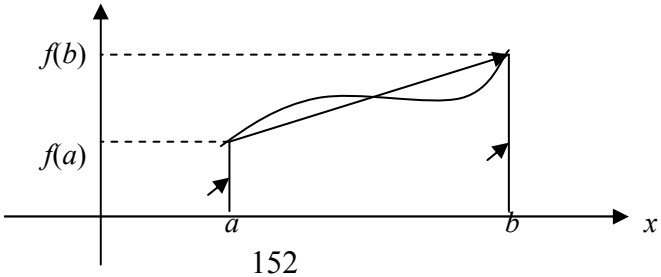
VIII.1. METODA TRAPEZULUI

Această metodă se obţine aproximând funcţia de integrat cu un polinom de interpolare Lagrange construit pe nodurile a şi b, adică printr-un polinom Lagrange de gradul întâi.

x	a	b
f	$f(a)=f_1$	$f(b)=f_2$

Pe baza tabelului de mai sus obţinem polinomul Lagrange:

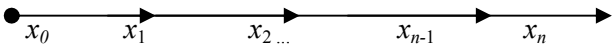
$$L_1(x)=\frac{x-b}{a-b}f(a)+\frac{x-a}{b-a}f(b)\Rightarrow \int_a^b f(x)dx=\frac{b-a}{2}\left[f(a)+f(b)\right]$$



Polinomul Lagrange de gradul 1, ne oferă o funcție care, restricționată la $[a,b]$, are ca imagine geometrică un segment de dreaptă care aproximează curba pe acest interval. Aria subgraficului acesteia este tocmai aria trapezului cu bazele *f(a), f(b)* şi înălțimea $b-a$. De aici rezultă şi denumirea metodei drept *metoda trapezului*.

Observația VIII.1. Dacă $f \in C^2[a,b]$, eroarea care apare aproximând integrala definită cu formula precedentă este: $-\frac{1}{12} f''(\xi)(b-a)^3$, $\xi \in (a,b)$.

Se observă că eroarea este mare dacă $[a,b]$ este de lungime mare. Pentru a micşora această eroare se consideră o diviziune a intervalului $[a,b]$ prin puncte echidistante. Se determină pasul h dat de: $h = \frac{b-a}{n}$ şi apoi nodurile echidistante: $x_i = x_0 + ih$, $i = \overline{0,n}$, unde $x_0 = a$, $x_n = b$.



Aplicând proprietatea de aditivitate (față de domeniu) a integralei definite şi formula precedentă pe fiecare subinterval format, obținem:

$$\int_a^b f(x)dx \cong \frac{h}{2}\left[f(x_0) + f(x_n) + 2\sum_{i=1}^{n-1} f(x_i) \right] \quad (*)$$

Aproximând $\int_a^b f(x)dx$ cu formula precedentă eroarea care se obține, notată $E(f)$, verifică inegalitatea:

cu $|E(f)| \le \frac{(b-a)^3 M}{12n^2}$, unde $M = \max_{x\in[a,b]} |f^{(2)}(x)|$.

Observaţia VIII.2. Mărginirea erorii ne permite să evaluăm numeric integrala cu o anumită precizie, stabilită prin eroarea admisibilă dată, ε_{adm}.

Impunând condiţia:

$$\frac{(b-a)^3}{12n^2}M < \varepsilon_{adm} \Rightarrow n^2 > \frac{(b-a)^3}{12\varepsilon_{adm}}M \Rightarrow$$

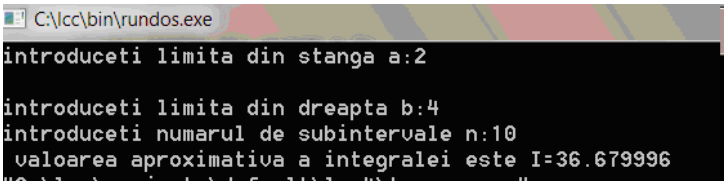
putem determina $n^* = \left[\frac{(b-a)^3}{12\varepsilon_{adm}}M\right] + 1$.

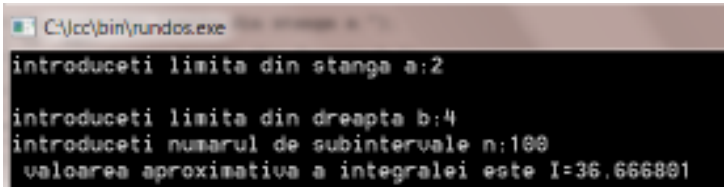
Putem evalua integrala pe baza formulei (*) considerând $n \geq n^*$ şi vom obţine o eroare mai mică decât eroarea admisibilă dată.

În continuare este prezentat un cod sursă în C, realizat pe baza metodei trapezelor, pentru evaluarea numerică a integralei funcţiei $f : R \to R,\ f(x) = x^2 + 3x$, pe un interval introdus de la tastatură.

```
/* Metoda Trapezelor */
#include<conio.h>
#include<stdio.h>
float a,b,x[1000],h, s;
float f(float);
int i,n;
void main()
{
printf("introduceti limita din stanga a:");
scanf("%f",&a);
printf("\nintroduceti limita din dreapta b:");
scanf("%f",&b);
printf("introduceti numarul de subintervale n:");
scanf("%d",&n);
h=(b-a)/n;
for(i=1;i<=n-1;i++)
```

```
        x[i]=a+i*h;
s=f(a)+f(b);
for(i=1;i<=n-1;i++)
        s=s+2*f(x[i]);
s=s*h/2;
printf(" valoarea aproximativa a integralei este I=%f",s);
}
float f(float x)
{return x*x+3*x;}
```





Valoarea exactă a integralei este:

$$\int_2^4 \left(x^2+3x\right)dx = \frac{110}{3} = 36,(6)$$

Valoarea numerică obținută cu ajutorul sursei precedente în cazul n=10 este 36.379996

Cea obținută în cazul n=100 este 36.666801

VIII.3. METODA SIMPSON

În cadrul acestei metode, formula de calcul a valorii aproximative a integralei, ce se obţine aproximând funcţia f printr-un

polinom Lagrange de gradul cel mult doi, care interpolează funcţia f în a, $c=\dfrac{a+b}{2}$ şi b, este:

$$\int_a^b f(x)dx=\frac{b-a}{6}\left(f(a)+4f(c)+f(b)\right)+E(f).$$

Eroarea este dată de:

$$E(f)=-\frac{1}{2880}(b-a)^5 f^{(4)}(\xi),\xi\in(a,b)$$

Fiind o eroare mare, în cazul unui interval de lungime supraunitară, se recurge şi în acest caz la discretizarea intervalului prin noduri echidistante ($x_i=x_0+ih$, $i=\overline{0,n}$, unde $x_0=a$, $x_n=b$, $h=\dfrac{b-a}{n}$) şi aplicarea formulei de mai sus pe fiecare subinterval format. Însumându-se acestea se obţine formula lui Simpson generalizată:

$$\int_a^b f(x)dx\cong\frac{h}{6}\left[f(x_0)+f(x_n)+2\sum_{i=1}^{n-1}f(x_i)+4\sum_{i=1}^{n}f(\frac{x_{i-1}+x_i}{2})\right]$$

Observaţia VIII.3. Aproximarea integralei $\int_a^b f(x)dx$ prin formula lui Simpson generalizată, în cazul unei funcţii de clasă C^4, se face cu eroarea $E(f)$, cu

$$|E(f)\ |\le\frac{(b-a)^5}{2880n^4}M',\text{ unde }M'=\max_{x\in[a,b]}|f^4(x)|.$$

Utilizând limbajul de programare C s-a realizat următorul program de calcul.

```
/* Metoda Simpson */
#include<conio.h>
#include<stdio.h>
float f(float);
float a,b,x[1000],y[1000], p, h, s;
int i,n;
void main ()
{
printf("introduceti limita din stanga a:");
scanf("%f",&a);
printf("\nintroduceti limita din dreapta b:");
scanf("%f",&b);
printf("introduceti numarul de subintervale n:");
scanf("%d",&n);
h=(b-a)/n;
for(i=0;i<=n;i++)
        x[i]=a+i*h;
s=f(a)+f(b);
for(i=1;i<=n-1;i++)
        s=s+2*f(x[i]);
p=0;
for(i=1;i<=n;i++)
        y[i]=(x[i-1]+x[i])/2;
for(i=1;i<=n;i++)
        p=p+4*f(y[i]);
s=(s+p)*h/6;
printf(" valoarea aproximativa a integralei este I=%f",s);
}
float f(float x)
{return x*x+3*x;}
```

VIII. 4. METODA NEWTON

Fie $f \in C^4[a,b]$ şi diviziunea:

$$a < x_1 < x_2 < b\,,\ x_i = a + i\frac{b-a}{3}, i = 1,2.$$

Folosind pentru aproximarea funcţiei un polinom de interpolare construit pe baza suportului de interpolare corespunzător diviziunii precedente se obţine, în mod analog, o nouă formulă de evaluare numerică a unei integrale definite, ce poartă numele de formula lui Newton: $\int_a^b f(x)dx = \frac{b-a}{8}\left[f(a)+3f(x_1)+3f(x_2)+f(b)\right]+E(f)$

$$E(f) = -\frac{1}{6480}(b-a)^5 f^{(4)}(\xi), \xi \in (a,b).$$

În cazul formulei generalizate se realizează o diviziune a intervalului $[a,b]$ în n părţi egale prin nodurile echidistante x_i, $i = \overline{0,n}$, $x_0 = a$, $x_n = b$ şi se aplică formula lui Newton pe fiecare interval $[x_{i-1}x_i]$. Însumând relaţiile obţinem formula lui Newton generalizată:

$$\int_a^b f(x)dx \cong \frac{h}{8}\left\{f(a)+f(b)+2\sum_{i=1}^{n-1} f(x_i)+3\sum_{i=0}^{n-1}\left[f(x_i+\frac{h}{3})+f(x_i+\frac{2h}{3})\right]\right\}$$

O margine superioară pentru valoarea absolută a erorii ce apare este dată de relaţia:

$$\left|E(f)\right| \le \frac{(b-a)^5}{6480n^4}M'', \text{ unde } M'' = \sup_{\xi\in[a,b]}\left|f^{(4)}(\xi)\right|$$

Observaţia VIII.4. M' din formula lui Simpson generalizată şi M'' din fomula lui Newton generalizată coincid.

Deoarece eroarea obţinută este mai mică atunci când se utilizează formula lui Newton

$$\left(\frac{(b-a)^5}{6480n^4}M'' \le \frac{(b-a)^5}{2880n^4}M'\right),$$

pentru un acelaşi *n*, este preferabil să se folosească în aplicaţii formula lui Newton. Programul în C corespunzător pentru cazul funcţiei $f : R \to R,\ f(x) = x^2 + 3x$ este prezentat în continuare.

```
/* Metoda Newton */
#include<conio.h>
#include<stdio.h>
#include<math.h>
float f(float);
float a,b,x[1000],y[1000],z[1000], p, h, s;
int i,n;
void main ()
{
printf("introduceti limita din stanga a:");
scanf("%f",&a);
printf("\nintroduceti limita din dreapta b:");
scanf("%f",&b);
printf("introduceti numarul de subintervale n:");
scanf("%d",&n);
h=(b-a)/n;
for(i=0;i<=n;i++)
        x[i]=a+i*h;
s=f(a)+f(b);
for(i=1;i<=n-1;i++)
        s=s+2*f(x[i]);
p=0;
for(i=1;i<=n;i++)
        y[i]=x[i-1]+h/3;
for(i=1;i<=n;i++)
        z[i]=x[i-1]+2*h/3;
for(i=1;i<=n;i++)
        p=p+3*(f(y[i])+f(z[i]));
s=(s+p)*h/8;
printf(" valoarea aproximativa a integralei este I=%f",s);
}float f(float x)
{return x*x+3*x;}
```

IX. METODE NUMERICE PENTRU REZOLVAREA ECUAŢIILOR DIFERENŢIALE

IX.1. INTRODUCERE

În acest capitol sunt prezentate câteva metode numerice pentru găsirea soluţiei unei probleme de forma:

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases} \tag{1}$$

Ea reprezintă de fapt o ecuaţie diferenţială însoţită de o condiţie iniţială şi poartă numele de problemă Cauchy (sau cu date iniţiale). Ele sunt foarte des întâlnite în practică deoarece majoritatea fenomenelor fizice studiate sunt scrise sub forma unor ecuaţii diferenţiale, căci este mai uşor de determinat cum variază o mărime decât mărimea în sine.

Aplicarea metodelor numerice pentru rezolvarea acestor tipuri de probleme se face în cazul în care condiţia de unicitate a soluţiei este asigurată.

Teorema IX.1. (stabileşte condiţii suficiente pentru ca problema precedentă să admită soluţie unică)

Fie U o vecinătate a punctului (x_0, y_0) de forma: $U = \{(x, y) / x \in [x_0 - a, x_0 + a], y \in [y_0 - b, y_0 + b]\}$. Dacă f este mărginită şi lipschitziană în raport cu variabila y (adică $|f(x, y_1) - f(x, y_2)| \le L|y_1 - y_2|$), atunci există un număr $h > 0$ şi o

mulţime $U_1 \subset U$ de forma: $U_1 = \{(x,y)/\, x \in [x_0, x_0 + h], |y - y_0| \le b\}$ astfel încât, pe această mulţime, soluţia problemei (1) există şi este unică.

Observaţia IX.1. Cele mai frecvente cazuri în care f satisface cerinţele teoremei precedente sunt:

1) Dacă f este astfel încât $\left|\frac{\partial f}{\partial y}\right| \le L$ *pe* $D \Rightarrow f$ este lipschitziană în raport cu y pe D.

2) Dacă $f \in C'(D)$=> derivatele parţiale sunt mărginite şi deci f este lipschitziană.

Există două direcţii principale de abordare a rezolvării numerice a ecuaţiei (1):

- se caută o funcţie elementară $\tilde{y}(x)$, care să aproximeze suficient de bine soluţia problemei (1)

$$\tilde{y}(x) = y_0 \text{ şi } |y(x) - \tilde{y}(x)| < \varepsilon \ (\forall) x \in [x_0, x_0 + h]$$

- se determină mărimile y_i, care aproximează valorile funcţiei $y(x)$ în punctele x_i, i=0,1,2,..., adică $|y(x) - y_i| < \varepsilon \quad i = 0,1,2,....$.

Cele mai simple metode numerice ce se pot utiliza în rezolvarea ecuaţiilor diferenţiale sunt: metoda integrării iterative, metoda dezvoltării în serie Taylor, metoda Euler şi Runge Kutta. Metoda de integrare iterativă şi cea de dezvoltare în serie fac parte din prima categorie iar metodele Euler şi Runge-Kutta fac parte din cea de a doua categorie.

IX.2. METODA INTEGRĂRII ITERATIVE

Fie $f:U\to R$, continuă pe U şi lipschitziană în raport cu y, de coeficient L. Se notează $M=\sup_U|f(x,y)|$. Dacă se alege $h=\min\left(a,\frac{1}{L},\frac{b}{M}\right)$, atunci şirul de aproximaţii succesive:

$$y_{n+1}=y_0+\int_{x_0}^{x}f(t,y_n(t))dt \qquad n=0,1,2,\ldots., \qquad (2)$$

converge uniform pentru $x\in[x_0,x_0+h]$ către soluţia problemei (1).

Astfel $y(x)=\lim_{n\to\infty}y_n(x)$ este soluţia problemei (1).

Are loc inegalitatea:

$$|y(x)-y_n(x)|<ML^n\frac{(x-x_0)^{n+1}}{(n+1)!}$$

Metoda integrării iterative mai poartă numele de metoda Picard şi, deoarece construieşte la fiecare iteraţie o aproximare a soluţiei problemei, se mai întâlneşte şi sub denumirea de metoda aproximațiilor succesive.

Algoritmul de aproximare este următorul:

Pasul 1: Se determină intervalul pe care procesul iterativ este convergent, adică valoarea lui h şi intervalul $[x_0,x_0+h]$.

Pasul 2: Se determină succesiv funcţiile $y_1(x),y_2(x),\ldots$ cu ajutorul formulelor (2).

Pasul 3: Se determină indicele n pentru care:

$$|y(x)-y_n(x)|<\varepsilon \quad (\forall)x\in[x_0,x_0+h],$$

unde ε este eroarea admisibilă, folosind inegalitatea:

$$|y(x)-y_n(x)|<ML^n\frac{(x-x_0)^{n+1}}{(n+1)!}.$$

Concluzie: y_n este aproximarea dorită.

Exemplul IX.1. Utilizând metoda aproximărilor succesive, să se determine o soluţie aproximativă pentru problema:

$$\begin{cases} y' = 3x+\frac{1}{2}y \\ y(0)=1 \end{cases} \Rightarrow x_0=0,\ y_0=1.$$

P1: Funcţia $f(x,y)=3x+\frac{1}{2}y\in C^1(D)$ pentru orice $D\subset R^2$. Fie $U=\{(x,y)\setminus |x|\le 1, |y-1|\le 1\}$($a$=1, b=1). Determinăm

$$M=\sup_{(x,y)\in U}|f(x,y)|=\sup_{(x,y)\in U}\left|3x+\frac{1}{2}y\right|=4,$$

şi $$L=\sup_{(x,y)\in U}\left|\frac{\partial f}{\partial y}(x,y)\right|=\sup_{(x,y)\in U}|1/2|=1/2.$$

Alegem $h=\min\left(1,2,\frac{1}{4}\right)=\frac{1}{4}$.

P2: Aproximaţiile sunt:

$$Y_1(x)=1+\int_0^x(3t+\frac{1}{2})dt=1+\frac{3x^2}{2}+\frac{x}{2}\ .$$

$$Y_2(x)=1+\int_0^x(3t+\frac{1}{2}\left(1+\frac{t}{2}+\frac{3t^2}{2}\right))dt=1+\frac{x}{2}+\frac{13}{8}x^2+\frac{x^3}{4}.$$

$$Y_3(x)=1+\int_0^x(3t+\frac{1}{2}\left(1+\frac{t}{2}+\frac{13}{8}t^2+\frac{t^3}{4}\right))dx=1+\frac{x}{2}+\frac{13x^2}{8}+\frac{13x^3}{48}+\frac{x^4}{32}$$

.

P3: Pentru $x \in [0,0+1/4] = [0,\ 0.25]$ deducem că:

$$|y(x) - y_3(x)| <$$

$$4 \cdot \left(\frac{1}{2}\right)^3 \cdot \frac{(0.25)^4}{4!} = \frac{1}{3} \cdot \left(\frac{1}{2}\right)^{12} \approx 8.138 \cdot 10^{-5} \approx 0.8 \cdot 10^{-4} < 10^{-4}$$

De exemplu pentru

$\varepsilon = 10^{-4} \Rightarrow y(x) = y_3(x) = 1 + \frac{x}{2} + \frac{13x^2}{8} + \frac{13x^3}{48} + \frac{x^4}{32}$ este soluţia problemei date. Ea aproximează soluţia exactă cu o eroare de 10^{-4}.

Exemplu IX.2. Să considerăm problema Cauchy

$$\begin{cases} y' = x^2 + 2y \\ y(1) = 0 \end{cases}$$

Să se determine soluția aproximativă a ei, obținută după trei iterații, folosind metoda integrării iterative și să se evalueze un majorant pentru eroarea ce se realizează prin această aproximare.

Soluție:

Alegem a=b=1și mulțimea $U = [0,2] \times [-1,1]$.

Detrminām $M = \sup_{(x,y) \in U} |f(x,y)| = \sup_{(x,y) \in U} |x^2 + 2y| = 6$

Determinăm $L = \sup_{(x,y) \in U} \left|\frac{\partial f}{\partial y}(x,y)\right| = \sup_{(x,y) \in U} |2| = 2$

Deci vom alege un număr $h < \frac{1}{6}$, atunci problema admite o singură soluţie aproximativă y^* definită pe intervalul $\left[\frac{5}{6}, \frac{7}{4}\right]$, ca limită a șirului de aproximări (2).

Pentru $x \in \left[\frac{5}{6}, \frac{7}{4}\right]$ vom avea:

$$y_1(x) = Ty_0(x) = \int_0^x t^2 dt = \frac{x^3}{3}$$

$$y_2(x) = \int_0^x \left(t^2 + \frac{2t^3}{3}\right) dt = \frac{x^3}{3} + \frac{2x^4}{3 \cdot 4} = \frac{x^3}{3} + \frac{x^4}{6}$$

$$y_3(x) = \int_0^x \left(t^2 + 2\left(\frac{t^3}{3} + \frac{t^4}{6}\right)\right) dt =$$

$$= \frac{x^3}{3} + \frac{x^4}{6} + \frac{x^5}{15}$$

Observăm că avem:

$$\left\| y_3 - y^* \right\| \leq 6 \cdot 2^3 \left(\frac{1}{6}\right)^4 \cdot \frac{1}{4!} = \frac{8}{6^3} \frac{1}{3 \cdot 8} = \frac{1}{3 \cdot 6^3} \approx 0{,}00154$$

IX.2. METODA DEZVOLTĂRII ÎN SERIE (METODA TAYLOR)

Dacă se presupune că funcţia *f* este derivabilă de o infinitate de ori într-o vecinătate a punctului (x_0, y_0) de rază *r*, atunci putem ataşa soluţiei problemei (1) seria Taylor corespunzătoare :

$$y(x) = y_0 + \frac{x - x_0}{1!} y'(x_0) + \frac{(x - x_0)^2}{2!} y''(x_0) + \ldots .$$

Coeficienţii se calculează consecutiv şi sunt valori cunoscute:

$y_0 = y(x_0)$, $y'(x_0) = f(x_0, y(x_0)) = f_0$.

Ei se obţin prin derivări succesive. De exemplu, $y''(x) = f'_x + y' f'_y + f'_x + f'_y \cdot f \Rightarrow$

$$y''(x_0) = f'_x(x_0, y_0) + f'_y(x_0, y_0) \cdot f(x_0, y_0) =$$
$$f'_x(x_0, y_0) + f'_y(x_0, y_0) \cdot f_0 .$$

Derivând funcţia precedentă, se poate calcula $y'''(x)$ şi apoi $y'''(x_0)$, etc. Astfel se pot obţine valorile derivatelor de orice ordin în punctul x_0.

Practic cu această metodă se aproximează soluţia $y(x)$ a problemei (1) în vecinătatea punctului (x_0, y_0)printr-un un polinom Taylor de grad *n* (*n*=4 sau 5 de cele mai multe ori).

Această metodă poate să ofere valorile soluţiei în nodurile echidistante ale unui interval.

Înlocuind *x* cu $x_0 + h$ vom deduce y_1, adică valoarea din punctul x_1. Avem relaţia:

$$y_1 = y_0 + h\left(f + \frac{h}{2}\left(f'_x + f \cdot f'_y\right)\right),$$

expresiile din dreapta egalităţii fiind evaluate în punctul (x_0, y_0)

Pentru celelalte valori deducem relaţia recursivă:

$$y_{i+1} = y_i + h\left(f + \frac{h}{2}\left(f'_x + f \cdot f'_y\right)\right),$$ cu termenul drept evaluat în punctul (x_i, y_i), $i = \overline{1, n-1}$.

Formule analoage se pot obţine dacă reţinem mai mulţi termeni din seria Taylor corespunzătoare.

Exemplul IX.3. Fie problema Cauchy următoare:

$$\begin{cases} y' = 2x - y \\ y(0) = -1 \end{cases}.$$

Să se aproximeze pe $[0,1]$ soluţia problemei folosind metoda Taylor, cu $n=3$. Pasul pe axa Ox se alege $h=0.1$. Să se compare rezultatele cu cele date de soluţia exactă:

$\hat{y}(x)=e^{-x}+2x-2$.

Soluţie:

Din datele problemei avem: $y(0)=-1\ y'(0)=1$.

Derivând în raport cu x ecuaţia diferenţială din problemă deducem că:

$$y''=2-y'=2-2x+y.$$

Considerând în relaţia precedentă $x=0$ şi folosind valorile anterior calculate obţinem:

$$y''(0)=2-1=1.$$

Procedând analog şi în continuare obţinem;

$$y'''=-y''=-2+y'=-2+2x-y,\ y'''(0)=-1$$

Astfel soluţia aproximativă a problemei date obţinută cu metoda Taylor este:

$$y(x)\approx y(0)+\frac{x}{1!}y'(0)+\frac{x^2}{2!}y''(0)+\frac{x^3}{3!}y'''(0)=$$

$$=-1+x+\frac{x^2}{2}-\frac{x^3}{6}.$$

Astfel, considerând $h=0.1$ putem deduce valoarea soluției în punctul $x_1=0+0,1=0,1$.

Obținem: $y_1=-1+0,1+\frac{0,1^2}{2}-\frac{0,1^3}{6}=-0,89517$.

Procedând analog în celelalte puncte găsim datele din tabelul următor. Sunt prezentate și erorile care apar între valorile calulate și cele exacte.

x	y	Soluti exacta	Eroarea absoluta
0.000000	-1.000000	-1.000000	0.000000
0.100000	-0.895163	-0.895167	0.000004
0.200000	-0.781269	-0.781277	0.000008
0.300000	-0.659182	-0.659192	0.000010
0.400000	-0.529680	-0.529692	0.000012
0.500000	-0.393469	-0.393483	0.000014
0.600000	-0.251188	-0.251203	0.000015
0.700000	-0.103415	-0.103430	0.000015
0.800000	0.049329	0.049313	0.000016
0.900000	0.206570	0.206553	0.000017
1.000000	0.367879	0.367863	0.000016

Eraoarea absolută maximă este deci 0.000017.

IX.3. Metoda EULER

Pornind de la faptul că soluţia unei ecuaţii diferenţiale de ordinul întâi reprezintă o familie de curbe din plan deducem că soluţia problemei (1) reprezintă şi ea o astfel de curbă.

Metoda Euler propune aproximarea acesteia, adică a soluţiei problemei, printr-o linie poligonală în care fiecare segment este coliniar cu direcţia tangentei la această curbă în punctul care coincide cu extremitatea sa stângă (vezi Fig. IX.1.). Punctele alese în lungul axei O*x* sunt echidistante, iar distanţa dintre oricare două alăturate se notează cu *h*.

Fie astfel nodurile echidistante $x_i = x_0 + ih$.

În punctul $M_0(x_0, y_0)$ se trasează segmentul ce trece prin M_0 şi are panta $y'(x_0) = f(x_0, y_0)$.

Ecuaţia este:

$y - y_0 = y(x_0)(x - x_0)$ sau $y - y_0 = f'(x_0, y_0)(x - x_0)$.

Se aproximează soluţia în punctul x_1 cu ordonata punctului M_1 de intersecţie dintre dreapta precedentă şi $x = x_1$.

Obţinem deci: $y_1 = y_0 = f_0 h$.

Procedăm analog în continuare considerând că M_1 joacă rolul lui M_0, etc. (vezi Fig. IX.1.).

Obţinem astfel formulele următoare pentru aflarea valorilor necunoscutei problemei în nodurile alese:

$$y_{i+1} = y_i + hf_i, \text{ unde } f_i = f(x_i, y_i).$$

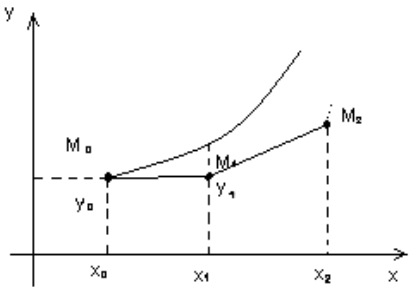


Fig. IX.1.

Astfel soluţia problemei (1) este aproximată prin funcţia tabel:

x	x_0	x_1	.....	x_n
y(x)	y_0	y_1	.....	y_n

Cunoscând valoarea lui *h*, care poartă numele de pasul metodei Euler, precum şi numărul de valori dorite ale soluţiei problemei, adică *n*, se obţine practic algoritmul de calcul numeric caracteristic metodei.

Algoritmul Metodei Euler:
Date de intrare: *n*-numărul de subintervale, *h*-pasul metodei, valorile iniţiale x_0, y_0 şi expresia funcţiei *f*.
Date de ieşire: vectorul ce conţine valorile numerice: $y_0, y_1, ..., y_n$.

P1: Se introduc datele de intrare;

P2: Se realizează diviziunea intervalului $[x_0, x_0 + nh]$ în n intervale prin nodurile $x_0, x_1, ..., x_n$.

$$(x_i = x_0 + ih);$$

P3: Se determină valorile y_i, $i = \overline{0,n}$ cu relaţia:

$$y_{i+1} = y_i + hf_i, \; f_i = f(x_i, y_i);$$

P4: Se afişează valorile funcţiei şi nodurile în care sunt calculate acestea, după care se opreşte algoritmul.

Observaţia IX.3. Pentru micşorarea erorilor ce intervin în metoda Euler se înlocuieşte direcţia segmentului $\overline{M_i M_{i+1}}$ cu direcţia corespunzătoare mijlocului segmentului $[x_i, x_{i+1}]$, rezultând metoda Euler modificată:

Formulele de calcul corespunzătoare acestei metode pot fi sintetizate astfel:

$$\begin{cases} x_i = x_0 + ih, f_i = f(x_i, y_i) \\ x_{i+\frac{1}{2}} = x_i + \dfrac{h}{2}, y_{i+\frac{1}{2}} = y_i + \dfrac{h}{2} f_i, f_{i+\frac{1}{2}} = f(x_{i+\frac{1}{2}}, y_{i+\frac{1}{2}}). \\ y_{i+1} = y_i + hf_{i+\frac{1}{2}} \end{cases}$$

Exemplul IX.4. Fie problema Cauchy:

$$\begin{cases} y' = 2x - y \\ y(0) = -1 \end{cases} \qquad x \in [0,1].$$

Să se determine o soluţie aproximativă a ei folosind metoda Euler cu pasul $h = 0.1$.

Să se compare valorile găsite cu cele ale soluției exacte $\hat{y}(x) = e^{-x} + 2x - 2$.

Soluţie:

Avem următoarele relaţii:

$$x_0 = 0; x_1 = 0{,}1; x_2 = 0{,}2; x_3 = 0{,}3; x_4 = 0{,}4...., x_{10} = 1.$$

$$y_0 = -1 \Rightarrow y_1 = y_0 + hf(x_0, y_0) = -1 + 0{,}1(2 - (-1) = -0.9 .$$

$$y_2 = y_1 + hf(x_1, y_1) \cong -0.79 \text{ , etc.}$$

Obţinem următorul tabel:

x	$y(x)$	$\hat{y}(x)$	eroarea absolută
0	-1.000000	-1	0
0.1	-0.900000	-0.89516	0.004837
0.2	-0.790000	-0.78127	0.008731
0.3	-0.671000	-0.65918	0.011818
0.4	-0.543900	-0.52968	0.01422
0.5	-0.409510	-0.39347	0.016041
0.6	-0.268559	-0.25119	0.017371
0.7	-0.121703	-0.10341	0.018288
0.8	0.030467	0.049329	0.018862
0.9	0.187420	0.20657	0.01915
1	0.348678	0.367879	0.019201

Se observă că cea mai mare eroare absolută obţinută este: 0.019201

Exemplul IX.4. Fie problema Cauchy:

$$\begin{cases} y' = y - \dfrac{y}{x} \\ y(1) = \dfrac{1}{2} \end{cases} \qquad x \in [1,2].$$

Să se determine o soluţie aproximativă a ei folosind metoda Euler cu pasul $h = 0.2$. Să se compare valorile găsite cu valorile soluției exacte: $\hat{y}(x) = \dfrac{e^{x-1}}{2x}$

Soluţie:

Avem următoarele relaţii:

$$x_0 = 1; x_1 = 1{,}2; x_2 = 1{,}4; x_3 = 1{,}6; x_4 = 1{,}8; x_5 = 2.$$

$$y_0 = \frac{1}{2} \Rightarrow y_1 = y_0 + hf(x_0, y_0) = \frac{1}{2} + 0{,}2(\frac{1}{2} - \frac{1}{2}) = 0{,}5.$$

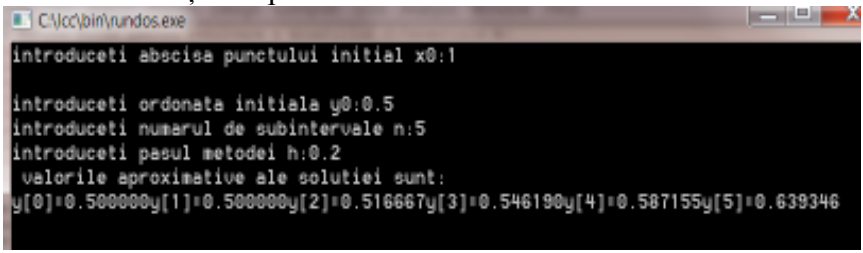
$$y_2 = y_1 + hf(x_1, y_1) = 0.5 + 0{,}2(0{,}5 - 0{,}5/1{,}2) = 0{,}516667 \text{ etc.}$$

Pe baza metodei Euler s-a realizat în C următorul program:

```
/* Metoda Euler pentru rezolvare a ecuatiilor diferentiale
*/
#include<conio.h>
#include<stdio.h>
float fun(float, float);
float x0,y0,x[100],y[100],h;
int i,n;
float fun(float x,float y)
{return y-y/x;}
void main ()
{
printf("introduceti abscisa punctului initial x0:");
scanf("%f",&x0);
printf("\nintroduceti ordonata initiala y0:");
scanf("%f",&y0);
```

```
printf("introduceti numarul de subintervale n:");
scanf("%d",&n);
printf("introduceti pasul metodei h:");
scanf("%f",&h);
x[0]=x0;
y[0]=y0;
for(i=1;i<=n;i++)
        {
                x[i]=x0+i*h;
                y[i]=y[i-1]+fun (x[i-1], y[i-1])*h;
        }
printf(" valorile aproximative ale solutiei sunt:\n");
for(i=0;i<=n;i++)
        printf("y[%d]=%f",i,y[i]);
getch();
}
```

Rezultatele obținute prin rularea lui sunt:



Obţinem următorul tabel:

x	$y(x)$	$\hat{y}(x)$	eroarea absolută
1	0.5	0.5	0
1.2	0.5	0.508918	0.008918
1.4	0.516667	0.532795	0.016128
1.6	0.546190	0.569412	0.023222
1.8	0.587155	0.618206	0.031051
2	0.639346	0.67957	0.040224

Se observă că cea mai mare eroare absolută este: 0.040224

Exemplul IX.4. Să se rezolve aceeaşi problemă folosind metoda Euler modificată.

Soluţie:

Nodurile de pe axa O*x* rămân aceleaşi:

$$x_0 = 1; x_1 = 1{,}2; x_2 = 1{,}4; x_3 = 1{,}6; x_4 = 1{,}8; x_5 = 2.$$

Avem, din datele problemei: $y_0 = \frac{1}{2}$.

Determinăm mijlocul primului subinterval format, valoarea ordonatei corespunzătoare şi apoi valoarea funcţiei în acest nod:

$$x_{1/2} = x_0 + h/2 = 1 + 0{,}2/2 = 1{,}1;$$

$$y_{1/2} = y_0 + (h/2) * f(x_0, y_0) = \frac{1}{2} + 0{,}1 * 0 = 0.5;$$

$$f(x_{1/2}, y_{1/2}) = f(1{,}1; 0.5) = 0.5 - (0.5)/1{,}1 = 0{,}045455 = f_{1/2}.$$

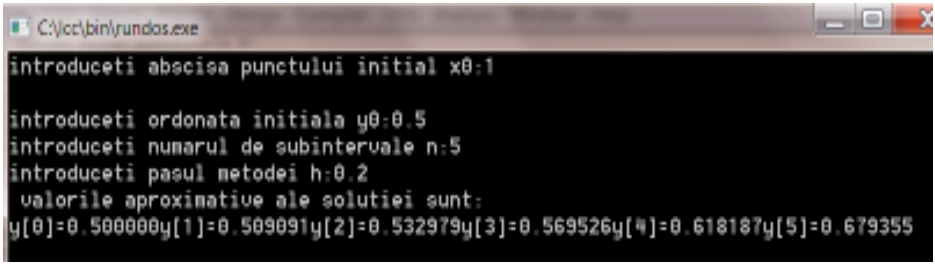
Cu aceste valori calculăm prima valoare numerică a necunoscutei din problemă, adică valoarea y_1:

$$y_1 = y_0 + f_{1/2} * h \Rightarrow y_1 = 0.5 + 0{,}2 * 0{,}045455 = 0{,}509091;$$

Rezultatele au fost obţinute cu ajutorul următorului program realizat în C:

```
/* Metoda Euler modificata */
#include<conio.h>
#include<stdio.h>
#include<math.h>
float fun(float, float);
float x0,y0,x[100],y[100],Y[100],a[100],b[100],h;
int i,n;
float fun(float x,float y)
{return y-y/x;}
void main ()
```

```
{
printf("introduceti abscisa punctului initial x0:");
scanf("%f",&x0);
printf("\nintroduceti ordonata initiala y0:");
scanf("%f",&y0);
printf("introduceti numarul de subintervale n:");
scanf("%d",&n);
printf("introduceti pasul metodei h:");
scanf("%f",&h);
x[0]=x0;
y[0]=y0;
for(i=0;i<=n;i++)
        {x[i+1]=x0+(i+1)*h;
                a[i]=x[i]+h/2;
                b[i]=y[i]+h*fun(x[i],y[i])/2;
                y[i+1]=y[i]+fun (a[i], b[i])*h;
        }
printf(" valorile aproximative ale solutiei sunt:\n");
for(i=0;i<=n;i++)
        printf("y[%d]=%f",i,y[i]);
getch();
}
```



Datele numerice obținute sunt trecute în tabelul următor, tabel în care se compară acestea cu valorile reale.

x	$y(x)$ Euler modificata	$\hat{y}(x)$	eroarea absolută
1	0.5	0.5	0
1.2	0.509091	0.508918	0.009091
1.4	0.532979	0.532795	0.016312
1.6	0.569526	0.569412	0.023336
1.8	0.618187	0.618206	0.031032
2	0.679355	0.67957	0.040009

Se observă că cea mai mare eroare absolută este: 0.040009 şi ea este mai mică decât în cazul aplicării metodei Euler. Se observă o îmbunătăţire a rezultatelor în cazul aplicării metodei Euler-modificată.

IX.4. Metode Runge-Kutta (R-K)

Aceste metode constau în aproximarea soluţiei $y(x)$ a problemei (1) în punctul $x_i + h$, dacă se cunoaşte soluţia în punctul x_i, cu ajutorul unor formule de tipul:

$$y_{i+1} \approx y_i + \Delta y_i ,$$

unde $\Delta y_i = c_1 k_1 + c_2 k_2 + ... + c_m k_m$, coeficienţii k_i, $i = \overline{1, m}$ fiind valorile funcţiei $f(x, y)$ în anumite puncte din intervalul (x_i, x_{i+1}).

Cea mai simplă dintre aceste metode este metoda Runge-Kutta de ordinul doi (RK2). În acest caz expresia lui Δy este:

$\Delta y = c_1 k_1 + c_2 k_2$, cu k_1, k_2 daţi de relaţiile:

$k_1 = hf(x_i, y_i)$, $k_2 = hf(x_i + h, y_i + \beta k_1)$,

unde c_1, c_2, α, β sunt constante ce urmează a fi determinate.
Pentru a determina acești coeficienți evaluăm y_{i+1} cu formula lui Taylor coespunzătoare funcției y în punctul x_i.
Avem:

$$y(x_{i+1}) = y(x_i) + hy'(x_i) + \frac{h^2}{2} y''(x_i) + \ldots = y(x_i) + hf(x_i, y_i) + \frac{h^2}{2} f'(x_i, y(x_i)) + \ldots$$

Dar

$$f'(x_i, y(x_i)) = \frac{\partial f}{\partial x}(x_i, y(x_i)) + \frac{\partial f}{\partial y}(x_i, y(x_i)) y'(x_i) = \frac{\partial f}{\partial x}(x_i, y(x_i)) + \frac{\partial f}{\partial y}(x_i, y(x_i)) f(x_i, y(x_i))$$

Astfel obținem:

$$y(x_{i+1}) = y(x_i) + hf(x_i) + \frac{h^2}{2}\left(\frac{\partial f}{\partial x}(x_i, y(x_i)) + \frac{\partial f}{\partial y}(x_i, y(x_i)) f(x_i, y(x_i))\right) + O(h^3)$$

(*)
Dezvoltând funcția $f(x_i + h, y_i + \beta k_1)$ în jurul lui (x_i, y_i) avem relția:

$$f(x_i + \alpha h, y_i + \beta k_1) = f(x_i, y_i) + \alpha h \frac{\partial f}{\partial x} + \beta k_1 \frac{\partial f}{\partial y} + O(h^2)$$

Dar $y_{i+1} = y_i + c_1 k_1 + c_2 k_2$ și astfel obținem:

$$y(x_{i+1}) = y(x_i) + h(c_1 + c_2) f + c_2 h^2 \left(\alpha \frac{\partial f}{\partial x} + \beta f \frac{\partial f}{\partial y}\right) + O(h^3),$$ funcțiile

din membrul drept fiind evaluate în $(x_i, y(x_i))$.

Comparând relația găsită cu relația (*) deducem că c_1, c_2, α, β verifică sistemul compatibil nedeterminat:

$$\begin{cases} c_1 + c_2 = 1 \\ \alpha = \beta = \dfrac{1}{2c_2} \end{cases},$$

Deducem de aici că există mai multe formule corespunzătoarei metodei RK2.

1. Una din soluțiile acestui sistem este: $c_1 = c_2 = \frac{1}{2},\ \alpha = \beta = 1$.

Astfel formula corespunzătoare ei este:

$$y_{i+1} = y_i + \frac{h}{2}\left(f(x_i, y_i) + f\left(x_i + h, y_i + hf(x_i, y_i)\right)\right),$$

$i = 0,...N-1$

2. O altă alegere a coeficienților c_1, c_2, α, β, și anume:

$c_1 = 0, c_2 = 1,\ \alpha = \beta = \frac{1}{2}$, conduce la obținerea formulei corespunzătoare metode Euler modificată:

$$y_{i+1} = y_i + hf\left(x_i + \frac{h}{2}, y_i + \frac{h}{2} f(x_i, y_i)\right),\ i = 0,...N-1.$$

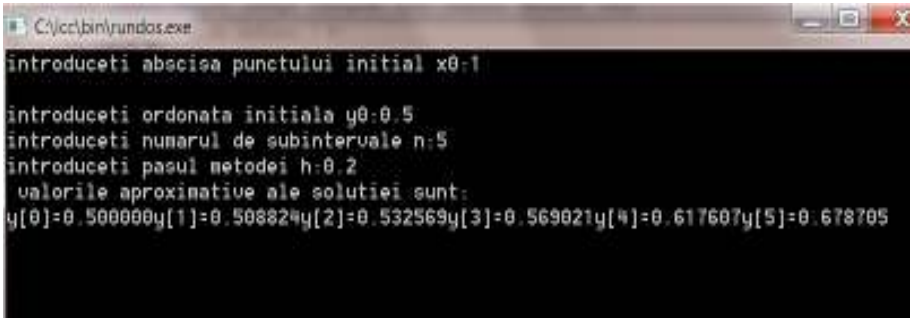
3. Dacă $c_1 = 1/4 \quad c_2 = 3/4 \quad \alpha = \beta = 2/3$, se obține formula:

$$y_{i+1} = y_i + \frac{h}{4}\left(f(x_i, y_i) + 3f\left(x_i + \frac{2}{3}h, y_i + \frac{2}{3}hf(x_i, y_i)\right)\right),$$

Pentru cazul formulei Runge-Kutta de ordinul 2 de mai sus s-a realizat un program în C, pentru rezolvarea problemei considerate în exemplul IX.4.

```
/* Metoda Runge-Kutta de ordinul 2 */
#include<conio.h>
#include<stdio.h>
#include<math.h>
float fun(float, float);
float x0,y0,x[100],y[100],c1[100],c2[100],h;
int i,n;
float fun(float x,float y)
{return y-y/x;}
```

```
void main ()
{
printf("introduceti abscisa punctului initial x0:");
scanf("%f",&x0);
printf("\nintroduceti ordonata initiala y0:");
scanf("%f",&y0);
printf("introduceti numarul de subintervale n:");
scanf("%d",&n);
printf("introduceti pasul metodei h:");
scanf("%f",&h);
x[0]=x0;
y[0]=y0;
for(i=0;i<=n;i++)
        {x[i+1]=x0+(i+1)*h;
                c1[i]=fun(x[i],y[i]);
                c2[i]=fun(x[i]+2*h/3,y[i]+2*h*c1[i]/3);
                y[i+1]=y[i]+h*(c1[i]+3*c2[i])/4;
        }
printf(" valorile aproximative ale solutiei sunt:\n");
for(i=0;i<=n;i++)
        printf("y[%d]=%f",i,y[i]);
getch();
}
```



În urma rulării acestui program pentru pasul $h = 0.2$ şi $n = 5$ s-au obţinut datele din tabelul următor. Tot în acest tabel se face o comparație între valorile numerice și cele exacte evaluându-se erorile absolute ce apar.

x	$y(x)$ RK2	$\hat{y}(x)$	eroarea absolută
1	0.5	0.5	0.000000
1.2	0.508824	0.508918	0.000094
1.4	0.532569	0.532795	0.000226
1.6	0.569021	0.569412	0.000391
1.8	0.617607	0.618206	0.000599
2	0.678705	0.67957	0.000865

Se observă că erorile ce apar sunt într-adevăr mai mici decât $h^3 = 0.008$.

Programul se poate adapta foarte uşor pentru cazul altor formule Runge-Kutta de ordinul doi, şi pentru alte ecuaţii diferenţiale de ordinul întâi cu soluţie unică.

Pentru a obţine însă aproximări mai bune se folosesc de cele mai multe ori formule de ordin superior, cum sunt formule Runge-Kutta de ordinul 3 și 4. Şi în acest caz există mai multe formule deoarece sistemul rezultat este tot compatibil nedeterminat.

Dintre cele mai importante formule Runge-Kutta de ordinul 3 amintim în continuare câteva:

- $y(x+h)=y(x)+1/6(k_1+4k_2+k_3)$,

unde $k_1=hf(x,y)$, $k_2=hf(x+h/2,y+k_1/2)$, $k_3=hf(x+h,y-k_1+2k_2)$

- $y(x+h)=y(x)+1/4(k_1+3k_3)$

unde $k_1=hf(x,y)$, $k_2=hf(x+h/3,y+k_1/3)$, $k_3=hf(x+2h/3,y_1+2k_2/3)$

În cazul folosirii acestor formule se obţine o eroare de ordinul lui h^4 .

Pentru cazul primei formule Runge-Kutta de ordinul 3 avem:

$$y(x+h)=y(x)+h/6(c_1+4c_2+c_3),$$

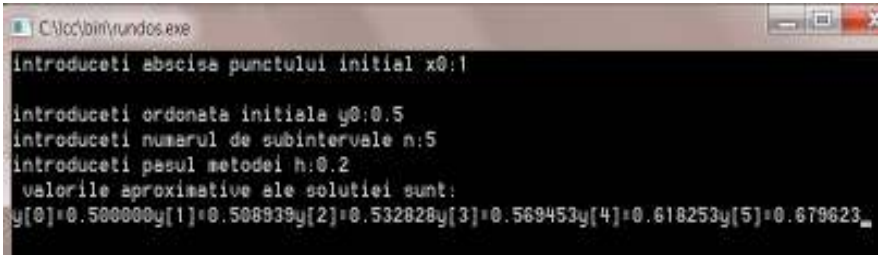
cu $c_1=f(x,y)$, $c_2=f(x+h/2,y+k_1/2)$, $c_3=f(x+h,y-k_1+2k_2)$

Un program realizat în C pe baza acestei formule, pentru rezolvarea problemei considerate în exemplul IX.4, este următorul:

/* Metoda Runge-Kutta de ordinul 3 */

```
#include<conio.h>
#include<stdio.h>
#include<math.h>
float fun(float, float);
float x0,y0,x[1000],y[1000],c1[100],c2[100],c3[100],h;
int i,n;
float fun(float x,float y)
{return y-y/x;}
void main ()
{
printf("introduceti abscisa punctului initial x0:");
scanf("%f",&x0);
printf("\nintroduceti ordonata initiala y0:");
```

```
scanf("%f",&y0);
printf("introduceti numarul de subintervale n:");
scanf("%d",&n);
printf("introduceti pasul metodei h:");
scanf("%f",&h);
x[0]=x0;
y[0]=y0;
for(i=0;i<=n;i++)
        {x[i+1]=x0+(i+1)*h;
                c1[i]=fun(x[i],y[i]);
                c2[i]=fun(x[i]+h/2,y[i]+c1[i]*h/2);
                c3[i]=fun(x[i]+h,y[i]-c1[i]*h+2*c2[i]*h);
                y[i+1]=y[i]+(c1[i]+4*c2[i]+c3[i])*h/6;
        }
printf(" valorile aproximative ale solutiei sunt:\n");
for(i=0;i<=n;i++)
        printf("y[%d]=%f",i,y[i]);
getch();
}
```



În urma rulării acestui program pentru pasul $h = 0.2$ şi $n = 5$ s-au obţinut datele de mai sus. În tabelul următor sunt comparate acetea cu soluția exactă și cu cea numerică obținută cu metoda RK2.

x	$y(x)$ RK3	$\hat{y}(x)$	eroarea absolută	$y(x)$ RK2	eroarea absolută
1	0.5	0.5	0.000000	0.5	0.000000
1.2	0.508824	0.508918	0.000094	0.508939	0.000021
1.4	0.532569	0.532795	0.000226	0.532828	0.000033
1.6	0.569021	0.569412	0.000391	0.569453	0.000041
1.8	0.617607	0.618206	0.000599	0.618253	0.000047
2	0.678705	0.67957	0.000865	0.679623	0.000053

Comparând aceste date cu cele din tabelul ce conţine valorile soluţiei exacte a problemei şi cu cele obținute ca urmare a aplicării metodei RK2, se observă că erorile ce apar sunt evident mai mici în cazul aplicării metodei RK3.

Programul se poate adapta foarte uşor pentru cazul altor formule Runge-Kutta, şi evident pentru cazul altor ecuaţii diferenţiale de ordinul întâi.

Cele mai cunoscute formule Runge-Kutta de ordinul 4 sunt:

- $y(x+h)=y(x)+1/6(k_1+4k_2+k_3+k_4)$

unde $k_1=hf(x,y)$, $k_2=hf(x+h/2,y+k_1/2)$, $k_3=hf(x+h/2,y+k_2/2)$, $k_4=hf(x+h,y+k_3)$

- $y(x+h)=y(x)+1/8(k_1+4k_2+k_3+k_4)$,

unde $k_1=hf(x,y)$, $k_2=hf(x+h/3,y+k_1/3)$, $k_3=hf(x+2h/3,y-k_1/3+k_2)$, $k_4=hf(x+h,y+k_1-k_2+k_3)$.

BIBLIOGRAFIE

1. Abdelwahab Kharab and Ronald B Guenther, An introduction to numerical methods: A MATLAB Approach, 2002
2. Antia H. M. Numerical methods for scientists and engineers, Birkhaussen, 2002
3. Brătianu C., Bostan V., Cojocia L, Negreanu G., Metode numerice,Editura Tehnică, Bucuresti 1996.
4. Ebâncă D., Metode de calcul numeric, Editura Sitech, Craiova 1994.
5. Iorga V., Jora B., etc. Programare Numerică, Editura Teora Bucureşti 1996.
6. Pavel L, Rizzoli I. Elemente de Analiză Funcţională şi Analiză Numerică, Editura Universităţii din Bucuresti, 2002.
7. Pitea A., Postolache M., Modelare numerică pentru ecuaţii diferenţiale si ecuaţii cu derivate parţiale, Editura Fair Partners, Bucureşti 2007.
8. Popa M., Popa A., Militaru R., Noţiuni de Analiză Numerică, Editura Sitech, Craiova, 2001.
9. Postolache M., Modelare Numerică. Teorie si Aplicaţii, Editura Fair Partners, Bucureşti 2008.
10. Rasa I., Vladislav T., Analiză Numerică. Curbe spline, operatori Bernstein, algoritmul lui Casteljau, Editura Tehnică, Bucureşti 1998.
11. Rinderiu P., Gruionu L., Metode Numerice-Elemente teoretice şi aplicative, Editura Universitaria Craiova 2003.
12. Simionescu I., Dranga M., Moise V., Metode Numerice în Tehnică. Editura Tehnică, Bucureşti 1995.
13. Vladislav T., Rasa I., Analiză Numerică, Editura Tehnică, Bucureşti 1997.
14. Vraciu G., Popa M., Efrem R., Micu S., Analiză Numerică. Culegere de exerciţii şi probleme, (vol.I), Editura Sitech, Craiova 1996.
15. Vraciu G., Popa A., Metode numerice cu aplicaţii în tehnica de calcul, Editura Scrisul Românesc, Craiova, 1982.