

**Ioan RUSU**

# **METODE NUMERICE**

**ALGORITMI ÎN LIMBAJ C**

**2006**

## CUPRINS

<b>1. ERORI .....</b>	<b>1</b>
1.1. Erori absolute și relative .....	1
1.2. Clasificarea erorilor .....	2
1.2.1. Erori inițiale .....	2
1.2.2. Erori de trunchiere .....	3
1.2.3. Erori de metoda .....	4
1.2.4. Erori de calcul .....	4
1.3. Propagarea erorilor .....	7
1.4. Grafuri de procedură .....	9
1.5. Reguli pentru mărirea preciziei calculelor .....	11
1.6. Aplicații .....	11
 <b>2. REZOLVAREA NUMERICĂ A ECUAȚIILOR ALGEBRICE .....</b>	 <b>15</b>
2.1. Separarea rădăcinilor .....	15
2.1.1. Metoda șirului lui Rolle .....	15
2.1.2. Metoda șirului lui Sturm .....	16
2.1.3. Metoda lui Budan-Fourier .....	18
2.2. Calculul rădăcinilor ecuațiilor algebrice .....	19
2.2.1 Metoda biseției .....	19
2.2.1.1. Algoritmul 2.2. Biseția pentru polinoame .....	20
2.2.1.2. Implementarea algoritmului 2.2 .....	21
2.2.1.3. Algoritmul 2.3. Biseția pentru ecuații transcendente .....	22
2.2.1.4. Implementarea algoritmului 2.3 .....	23
2.2.2. Metoda aproximațiilor succesive .....	23
2.2.2.1. Algoritmul 2.4. Aproximații succesive .....	26
2.2.2.2. Implementarea algoritmului 2.4 .....	27
2.2.3. Metoda aproximațiilor succesive cu viteză mare de convergență .....	28
2.2.4. Metoda Newton-Raphson .....	29
2.2.4.1. Algoritmul 2.5 .....	31
2.2.4.2. Implementarea algoritmului 2.5 .....	32
2.2.5. Metoda Newton—Raphson pentru polinoame .....	33
2.2.5.1. Algoritmului 2.6. Newton-Raphson pentru polinoame .....	33
2.2.5.2. Implementarea algoritmului 2.6 .....	34
2.2.6. Metoda Newton-Raphson pentru rădăcini foarte apropiate .....	35
2.2.7. Metoda lui Bairstow .....	36
2.2.7.1. Algoritmul 2.7 .....	39
2.2.7.2. Implementarea algoritmului .....	41
2.3. Aplicație .....	44
 <b>3. REZOLVAREA NUMERICĂ A SISTEMELOR DE ECUAȚII .....</b>	 <b>47</b>
3.1. Rezolvarea numerică a sistemelor liniare .....	47
3.1.1. Metode directe .....	48
3.1.1.1. Sisteme inferior triunghiulare .....	48
3.1.1.1.1. Algoritmul 3.1 .....	48
3.1.1.1.2. Implementarea algoritmului 3.1 .....	49
3.1.1.2. Sisteme superior triunghiulare .....	50
3.1.1.2.1. Algoritmul 3.2 .....	50
3.1.1.2.2. Implementarea algoritmului 3.2 .....	50
3.1.1.3. Metoda lui Gauss de eliminare .....	51
3.1.1.3.1. Algoritmul 3.3 .....	52
3.1.1.3.2. Implementarea algoritmului .....	53
3.1.1.4. Metoda lui Crout .....	53
3.1.1.4.1. Algoritmul 3.4 .....	53

3.1.1.4.2. Implementarea algoritmului 3.4 .....	57
3.1.1.5. Metoda lui Cholesky .....	58
3.1.1.5.1. Algoritmul 3.5 .....	58
3.1.1.5.2. Implementarea algoritmului 3.5 .....	58
3.1.1.6. Metoda Gauss-Jordan sau matriceal formală .....	59
3.1.1.6.1. Algoritmul 3.6.....	60
3.1.1.6.2. Implementarea algoritmului 3.6.....	61
3.1.1.7. Metoda factorizării QR .....	75
3.1.1.7.1. Algoritmul 3.7.....	64
3.1.1.7.2. Implementarea algoritmului 3.7 .....	65
3.1.1.8. Metoda de rezolvare a sistemelor tridiagonale .....	67
3.1.1.8.1. Algoritmul 3.8.....	69
3.1.1.8.2. Implementarea algoritmului 3.8 .....	70
3.1.2. Metode indirecte .....	71
3.1.2.1. Metoda lui Jacobi.....	71
3.1.2.1.1. Algoritmul 3.9.....	72
3.1.2.1.2. Implementarea algoritmului 3.9 .....	73
3.1.2.2. Metoda Gauss-Seidel .....	75
3.1.2.2.1. Algoritmul 3.10 .....	75
3.1.2.2.2. Implementarea algoritmului 3.10 .....	76
3.2. Rezolvarea numerică a sistemelor neliniare.....	78
3.2.1. Metoda lui Newton de rezolvare a sistemelor neliniare .....	78
3.2.1.1. Algoritmul 3.11.....	80
3.3. Aplicații .....	82
<b>4. DERIVAREA NUMERICĂ .....</b>	<b>84</b>
4.1. Derivata numerică prin două puncte .....	84
4.2. Derivata numerică prin trei puncte.....	85
4.3. Derivata numerică prin cinci puncte .....	89
4.4. Calculul derivatei funcțiilor tabelate .....	91
4.5. Aplicații .....	91
<b>5. INTEGRAREA NUMERICĂ .....</b>	<b>93</b>
5.1. Integrarea funcțiilor proprii de o singură variabilă. Metode cu divizare constantă .....	93
5.1.1. Metoda dreptunghiului .....	93
5.1.1.1. Algoritmul 5.1.....	94
5.1.1.1.1. Implementarea algoritmului 5.1 .....	95
5.1.2. Metoda trapezului de integrare numerică .....	95
5.1.2.1. Algoritmul 5.2.....	100
5.1.2.2. Implementarea algoritmului .....	100
5.1.3. Metoda lui Richardson .....	100
5.1.3.1. Algoritmul 5.3.....	101
5.1.3.2. Implementarea algoritmului 5.3 .....	102
5.1.4. Metoda lui Simpson .....	102
5.1.4.1. Algoritmul 5.4.....	103
5.1.4.2. Implementarea algoritmului 5.4 .....	104
5.2. Metode cu diviziune variabilă.....	104
5.2.1. Metoda cuadraturii gaussiene cu două puncte.....	104
5.2.2. Metoda cuadraturii gaussiene pentru mai multe puncte .....	107
5.2.2.1. Eroarea de trunchiere .....	108
5.2.2.2. Algoritmul 5.5.....	108
5.2.2.3. Implementarea algoritmului 5.5 .....	109
5.3. Compararea metodelor de integrare numerică .....	116
5.4. Calculul numeric al integralelor improprii.....	116
5.5. Calculul numeric al integralelor duble.....	118

5.5.1. Formula de cubatură a trapezului.....	118
5.5.1.1. Algoritmul 5.6.....	119
5.5.1.2. Implementarea algoritmului 5.6.....	120
5.5.2. Formula lui Simpson de cubatură.....	120
5.5.2.1. Algoritmul 5.7.....	121
5.6. Aplicații .....	122
<b>6. INTERPOLAREA .....</b>	<b>123</b>
6.1. Interpolarea polinomială .....	123
6.2. Polinomul de interpolare de speța întâi al lui Newton .....	129
6.3. Polinomul de interpolare de speța a doua al lui Newton .....	131
6.4. Polinomul lui Newton de interpolare cu diferențe divizate.....	133
6.5. Metoda lui Aitken de interpolare .....	135
6.6. Interpolarea cu funcții spline.....	137
6.7. Interpolarea funcțiilor periodice.....	142
6.8. Interpolarea funcțiilor de mai multe variabile.....	144
6.9. Aplicație.....	145
<b>7. OPTIMIZĂRI .....</b>	<b>148</b>
7.1. Metoda celor mai mici pătrate .....	148
7.1.1. Regresia liniară .....	149
7.1.1.1. Algoritmul 7.1.....	150
7.1.1.2. Implementarea algoritmului 7.1 .....	151
7.1.2. Regresia polinomială .....	151
7.1.2.1. Algoritmul 7.2.....	152
7.1.2.2. Implementarea algoritmului 7.2 .....	153
7.1.3. Regresia hiperbolică.....	153
7.1.3.1. Algoritmul 7.3.....	154
7.1.3.2. Implementarea algoritmului 7.3 .....	155
7.1.4. Regresia exponențială .....	156
7.1.4.1. Algoritmul 7.4.....	156
7.1.4.2. Implementarea algoritmului 7.4 .....	157
7.1.5. Regresia geometrică.....	158
7.1.5.1. Algoritmul 7.5.....	158
7.1.5.2. Implementarea algoritmului 7.5 .....	159
7.1.6. Regresia trigonometrică .....	160
7.1.6.1. Algoritmul 7.6.....	160
7.1.6.2. Implementarea algoritmului 7.6 .....	161
7.1.7. Regresia multiplă .....	162
7.1.7.1. Algoritmul 7.7.....	163
7.1.7.2. Implementarea algoritmului 7.7 .....	164
7.2. Optimizarea neliniară fără restricții .....	165
7.2.1. Metode aleatoare de căutare.....	166
7.2.1.1. Algoritmul 7.8.....	166
7.2.1.2. Implementarea algoritmului .....	167
7.2.2. Metoda căutării unidimensionale .....	168
7.2.2.1. Algoritmul 7.9.....	168
7.3. Aplicații .....	170
<b>8. REZOLVAREA ECUAȚIILOR DIFERENȚIALE .....</b>	<b>171</b>
8.1. Rezolvarea ecuațiilor diferențiale ordinare de ordinul întâi .....	171
8.1.1. Metoda seriilor lui Taylor .....	172
8.1.2. Metodele Runge-Kutta .....	173

8.1.3. Metodele Runge- Kutta de ordinul doi.....	175
8.1.3.1. Metoda lui Euler îmbunătățită.....	175
8.1.3.2. Metoda lui Euler modificată .....	178
8.1.4. Metoda Runge-Kutta de ordinul patru .....	181
8.1.5. Metoda predictor-corector.....	184
8.1.6. Metoda lui Milne.....	188
8.2. Integrarea numerică a sistemelor de ecuații diferențiale de ordinul întâi și a ecuațiilor diferențiale de ordinul doi .....	190
8.3. Rezolvarea numerică a ecuațiilor diferențiale cu derivate parțiale .....	193
8.3.1. Metoda diferențelor finite .....	194
8.3.1.1. Ecuația lui Laplace.....	215
8.3.1.2. Ecuații diferențiale cu derivate parțiale de tip parabolic.....	298
8.3.1.3. Ecuații cu derivate parțiale de tip hiperbolic.....	200
8.4. Aplicație.....	203
<b>9. REZOLVAREA NUMERICĂ A ECUAȚIILOR INTEGRALE .....</b>	<b>204</b>
9.1. Integrarea ecuației Fredholm neomogenă de speța a doua .....	204
9.2. Integrarea ecuației Volterra neomogenă de speța a doua .....	207
9.3. Aplicație.....	209
<b>10. VECTORI SI VALORI PROPRII.....</b>	<b>205</b>
10.1. Tipuri de matrice.....	206
10.2. Localizarea valorilor proprii .....	208
10.3. Metode de determinare a vectorilor și valorilor proprii .....	209
10.3.1. Metoda puterii.....	209
10.3.2. Metoda lui Krilov.....	211
10.3.3. Metoda lui Householder.....	213
10.3.4. Metoda RT .....	221
10.3.5. Metoda LR .....	224
10.4. Aplicații .....	229
<b>11. FUNCȚII SPECIALE.....</b>	<b>236</b>
11.1. Funcția gamma.....	236
11.2. Funcția factorial .....	238
11.3. Coeficienții binomiali .....	240
11.4. Funcția beta.....	242
11.5. Funcțiile Bessel.....	244
11.6. Aplicații .....	250
<b>12. TRANSFORMATĂ FOURIER DISCRETĂ .....</b>	<b>252</b>
12.1. Analiza în frecvență a semnalelor în timp discret .....	254
12.1.1. Reprezentarea secvențelor cu transformate Fourier .....	254
12.1.1.1. Algoritmi rapizi pentru FFT .....	255
12.1.2. Algoritmul Goertzel .....	264
12.3. Aplicații .....	267
<b>13. CĂUTĂRI .....</b>	<b>270</b>
13.1. Metode de inserție.....	270
13.1.1. Sortarea prin inserție directă .....	270
13.1.2. Sortarea prin inserție directă a două mulțimi .....	272
13.1.3. Metoda Shell .....	273
13.2. Metode de sortare prin interschimbare.....	275
13.3. Sortarea rapidă .....	276
13.4. Aplicație.....	278

---

<b>ANEXA 1 .....</b>	<b>279</b>
<b>BIBLIOGRAFIE.....</b>	<b>281</b>

# 1

## ERORI\*

Rezolvarea problemelor moderne de o mare complexitate din științele aplicate, cum ar fi: astronomia, geodezia, electronica etc., nu se poate realiza fără calculator asistat de un pachet de metode numerice matematice.

Rezultatele numerice obținute în acest gen de aplicații, cu ajutorul calculatorului, necesită o analiză a erorii, datorită faptului că datele inițiale pentru determinarea numerică a rezultatului sunt obținute de obicei prin observații sau măsurători afectate de erori și operațiile numerice efectuate de calculator sunt aproximative atunci când numerele nu au o reprezentare internă exactă. Măsurătorile experimentale nu pot fi realizate exact, prin repetarea lor obținându-se rezultate diferite. Ca urmare, datele obținute prin măsurători conțin erori care se reflectă în rezultatul final. Calculatoarele numerice au o reprezentare internă exactă a numerelor întregi pentru care realizează calcule exacte și o reprezentare internă aproximativă a numerelor reale care influențează exactitatea rezultatului.

Determinarea unor mărimi în problemele tehnice se face de obicei cu finalitate practică. Aici apar erorile inevitabile ale procesului de execuție sau fabricație; pentru fiecare componentă există “toleranțe”, adică limite ale erorilor între care componenta poate fi utilizată.

Studiul acestui capitol se referă la erorile ce apar în operațiile aritmetice realizate de calculator. Se definesc tipurile de erori ce apar în procesele electronice și se explicitează funcțiile eroare.

### 1.1. ERORI ABSOLUTE ȘI ERORI RELATIVE

Deoarece valoarea măsurată sau calculată a unei mărimi nu reprezintă valoarea adevărată sau exactă a mărimii, o numim valoare aproximativă .

**Definiția 1.1.** Numim *eroare* diferența dintre valoarea adevărată (exactă) și valoarea aproximativă. Matematic definiția poate fi exprimată prin formula:

$$e_x = x - \bar{x}, \quad (1.1)$$

unde :  $e_x$  reprezintă eroarea,  $x$  valoarea adevărată, iar  $\bar{x}$  valoarea aproximativă.

---

\*) *Bibliografie:* [6],[7],[12],[21],[22]

Din cele trei valori care intervin în definiția erorii cunoaștem numai valoarea aproximativă. Valoarea adevărată nu o cunoaștem, deoarece dacă am cunoaște-o nu s-ar mai pune problema existenței erorii. Pentru ca ecuația (1.1) să fie rezolvabilă se face o estimare asupra erorii, adică se precizează o margine superioară a acesteia. În acest caz se poate determina valoarea adevărată cu ajutorul valorii aproximative, pentru o margine a erorii dată.

Deoarece unii autori definesc eroarea ca diferența:

$$e_x = \bar{x} - x, \quad (1.2)$$

pentru a reuni cele două definiții ale erorii într-una singură s-a trecut la definirea *erorii absolute*, astfel:

$$|e_x| = |x - \bar{x}|. \quad (1.3)$$

Dacă se cunoaște un minorant  $m$  și un majorant  $M$  al mărimii  $x$  atunci orice element  $e_x^*$  al mulțimii  $\{|e_x|, M - m\}$  se numește *eroare absolută la limită*.

**Definiția 1.2.** Numim *eroare relativă* raportul dintre eroarea absolută și valoarea aproximativă absolută  $\varepsilon_x = |e_x|/\bar{x}$  (1.4)

În cazul erorii relative se poate defini eroarea relativă la limită. Din aplicarea proprietăților modului rezultă:

$$||x| - |\bar{x}|| \leq |x - \bar{x}| = |e_x| \leq e_x^* \quad (1.5)$$

$$|\bar{x}| - e_x^* \leq |x| \leq |\bar{x}| + e_x^*$$

**Definiția 1.3** Numim *eroare relativă la limită* raportul:

$$\varepsilon_x^* = \frac{e_x^*}{|\bar{x}| - \varepsilon_x^*} \quad (1.6)$$

Pentru numerele apropiate de 1, eroarea absolută și eroarea aproximativă sunt aproape egale, deoarece eroarea absolută se împarte la un număr aproximativ egal cu 1. Pentru numere diferite mult de 1, cele două erori absolută și relativă diferă foarte mult. Din acest motiv, atunci când nu se poate deduce tipul erorii este necesar să se specifice dacă aceasta este relativă sau absolută. Erorile absolute pentru mărimi care au unități de măsură sunt exprimate în unitățile de măsură ale mărimilor, iar erorile relative nu au unități de măsură și sunt exprimate în procente.

## 1.2. CLASIFICAREA ERORILOR

Făcând analiza erorilor pentru un rezultat numeric, se constată că eroarea finală poate depinde de o serie de erori cum ar fi: erorile inițiale, erorile de trunchiere, erorile de metodă și erorile de calcul (rotunjire).

### 1.2.1. ERORI INIȚIALE

Aceste erori sunt determinate de erorile care intervin în datele inițiale de calcul cauzate deseori de măsurare, greșeli de citire a datelor, greșeli de introducere a datelor în calculator sau de necesitatea reprezentării unei valori cu un număr finit de cifre. Erorile de măsurare se datorează în primul rând preciziei instrumentului și sunt influențate de aceasta astfel: cu cât precizia instrumentului este mai mare, cu atât eroarea de măsură este mai mică. Greșelile de citire, de introducere a datelor fac parte din erorile de observare, care au un caracter aleator și



sunt caracterizate de funcția de repartiție normală sau gaussiană din expresia (1.7), a cărei reprezentare este dată în fig.1.1.

$$\varphi(\varepsilon) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot \exp(-0.5 \cdot (\varepsilon / \sigma)^2), \quad (1.7)$$

Calculul probabilității ca eroarea de observație a unui procedeu de măsură să fie cuprinsă între două valori date  $x_1$  și  $x_2$  se realizează ca o sumare pe intervalul dat a funcției de repartiție, prezentată în expresia (1.8):

$$P(x_1 \leq \varepsilon < x_2) = \int_{x_1}^{x_2} \varphi(\varepsilon) d\varepsilon \quad (1.8)$$

Multe numere nu pot fi reprezentate exact printr-un număr finit de cifre. Dintre aceste

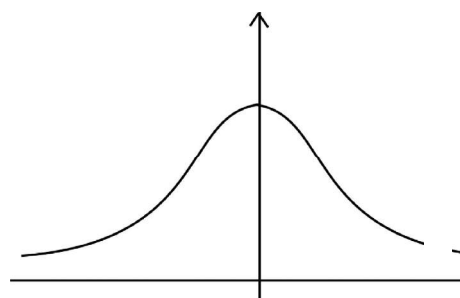


Fig1.1.Funcția de repartiție gaussiană  
 $\varepsilon$ -eroarea de observație; $\sigma$ -dispersia

numere amintim:  $\pi = 3.141592654...$ ,  $\sqrt{2} = 1.414213562$ ,  $e = 2.73...$  Aceste numere cu o infinitate de cifre sunt des utilizate în calcule, dar cu diferite aproximații care dau erori inerente. Numerele transcendente și iraționale sunt reprezentate printr-o infinitate de cifre în orice sistem de numerație. Unele numere, reprezentate cu un număr finit de cifre într-un sistem de numerație, sunt reprezentate cu un număr infinit de cifre în alt sistem de numerație. Ca exemplu poate fi dat

numărul 0.1 în sistemul de numerație zecimal, care în sistemul de numerație binar este dat de numărul cu o infinitate de cifre 0.0001100110011...

### 1.2.2. ERORI DE TRUNCHIERE

Aceste erori sunt introduse de procedura numerică aplicată. Ca exemplu putem da procedura numerică de calcul a valorii funcției trigonometrice  $\cos x$ . Cea mai utilizată procedură numerică de calcul este dezvoltarea în serie Taylor, în jurul punctului 0 a funcției  $\cos x$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, \quad (1.9)$$

unde argumentul  $x$  este dat în radiani. Un calcul exact al funcției  $\cos(x)$  este imposibil, programul de calcul oprindu-se la un anumit termen al seriei, obținându-se în acest mod o valoare a funcției cu o anumită eroare.

Pentru cazul general al unei funcții  $f(x)$  dezvoltată în serie Taylor, în jurul punctului  $x_0$ , oprirea la termenul de rang  $n$  dă o eroare ce poate fi calculată cu ajutorul formulei restului lui Lagrange:

$$R_n(x) = \frac{(x - x_0)^{n+1}}{(n+1)!} f^{(n+1)}(\xi) \quad \xi \in (x_0, x) \quad (1.10)$$

Procedurile utilizate în calculele numerice sunt finite, iar erorile datorate acestor proceduri care întrerup un calcul infinit teoretic, poartă numele de erori de trunchiere. Aceste erori în multe cazuri nu se pot calcula exact, dar se pot estima sau li se pot stabili limitele.

### 1.2.3. ERORI DE METODĂ

Pentru rezolvarea unei probleme se pot aplica mai multe metode care au erori specifice lor. Prin alegerea celei mai adecvate metode în rezolvarea problemei, eroarea poate fi micșorată. Acest lucru se poate realiza dacă cel implicat în rezolvarea problemei cunoaște metodele numerice. Deoarece metodele numerice sunt aproximații ale metodelor analogice de calcul este necesar să se cunoască precizia aproximațiilor. În toate capitolele care urmează sunt tratate metodele numerice împreună cu erorile lor de calcul, iar unde este necesar se face și o comparație între metode.

### 1.2.4. ERORI DE CALCUL PRIN ROTUNJIRE

Pentru o înțelegere mai bună a acestui tip de erori, vom studia mai întâi calculul în virgulă mobilă, calcul realizat de calculator. Un număr  $q$  este reprezentat în virgulă mobilă dacă el este scris sub forma:  $q = f \cdot b^x$  (1.11)

unde  $f$  este fracția, care mai poartă numele de mantisă și are valoare subunitară,  $x$  exponentul, care este întotdeauna întreg și  $b$  baza de numerație.

Calculatoarele utilizează scrierea normalizată a numerelor în virgulă mobilă, deoarece sub această formă se obține o precizie maximă de calcul. Un număr scris în virgulă mobilă este normalizat dacă prima cifră, după virgulă în mantisă este diferită de zero (excepție făcând numărul zero). În bazele 10, 2 și 16, avem următoarele reprezentări normalizate în virgulă mobilă ale unui număr oarecare  $q$ :

$$\begin{aligned} q &= f \cdot 10^x, & 0.1 \leq |f| < 1; \\ q &= f \cdot 2^x, & 1/2 \leq |f| < 1; \\ q &= f \cdot 16^x, & 1/16 \leq |f| < 1 \end{aligned} \quad (1.12)$$

Adunarea și scăderea numerelor în virgulă mobilă se realizează de către calculator după următorul algoritm prezentat în organigrama din fig.1.2. Se consideră că numerele sunt date în baza 10 sub forma  $f_1 \cdot 10^{x_1}$ ,  $f_2 \cdot 10^{x_2}$  iar suma sau diferența  $f_3 \cdot 10^{x_3}$ .

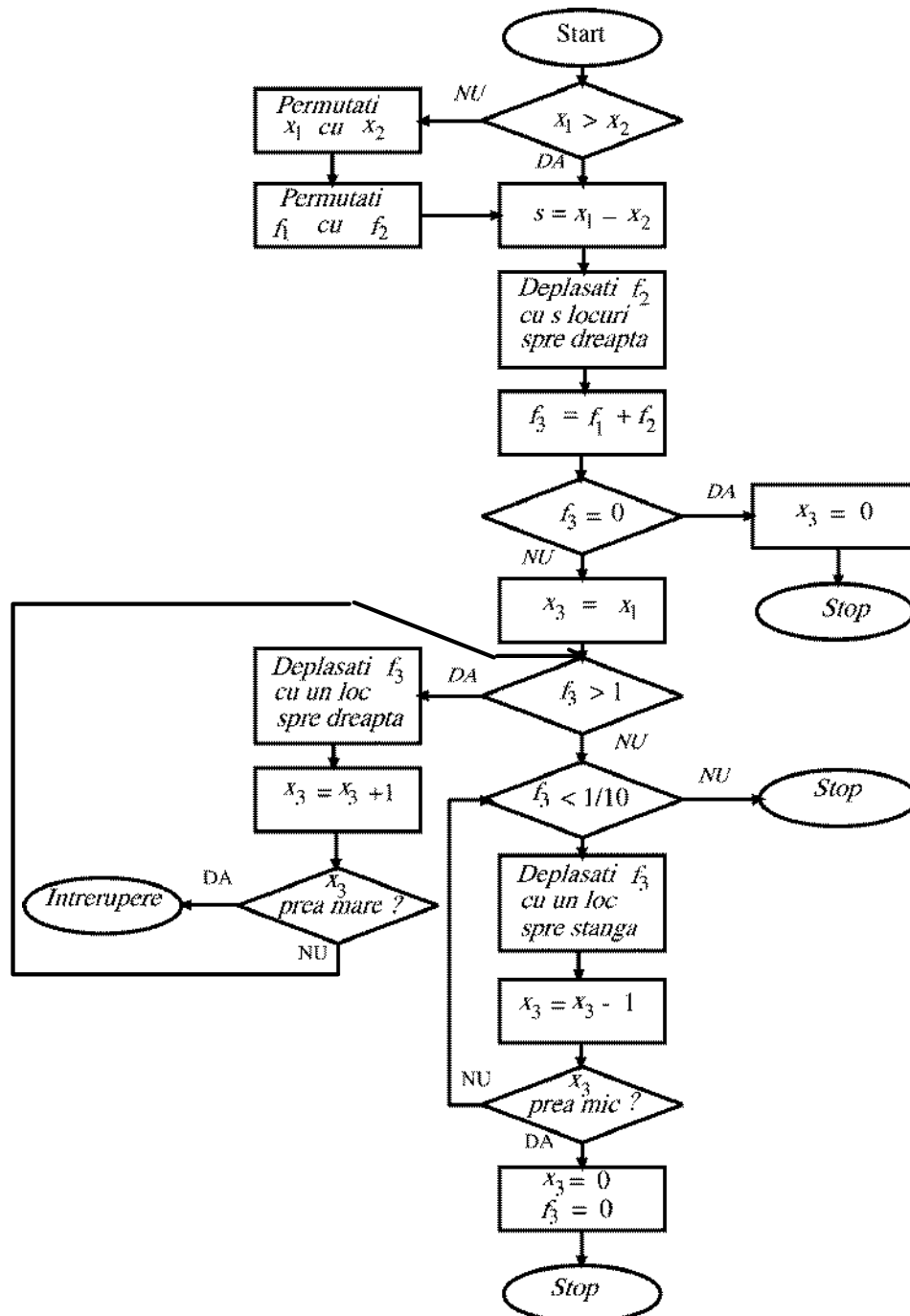


Fig.1 2. Organigrama operației de adunare și scădere în virgulă mobilă, pentru numere în baza zece.

Calculatoarele moderne au cifre de gardă ce reprezintă locurile pe care se deplasează spre dreapta ultima cifră a numărului cu exponent mai mic. După aceste operații de adunare sau scădere, cifra de pe aceste locuri poate fi readusă înapoi în timpul normalizării. În cazul efectuării sumei poate apare situația ca un exponent să fie mai mic decât cel admis de calculator; de exemplu calculatorul poate admite exponenți până la ordinul -99 și dorim să normalizăm numărul prin deplasare spre stânga cu un loc, ceea ce ar însemna un exponent de -100, neadmis de calculator. În acest caz avem o depășire inferioară (*under flow*), iar numărul este considerat zero.

Mai poate apărea situația obținerii după însumare a unui exponent maxim de două cifre, pe care-l poate admite calculatorul, de exemplu 99, și vrem să normalizăm rezultatul prin deplasare cu un loc spre dreapta, deci mărirea exponentului cu o unitate, operație nepermisă de calculator. În acest caz apare o depășire superioară (*over flow*) și programul se întrerupe.

Se consideră un număr în baza 10 reprezentat în virgulă mobilă, cu o mantisă compusă din  $t$  cifre ca fiind rezultatul unor operații aritmetice. Numărul poate fi scris:

$$q = f_q \cdot 10^x + g_q \cdot 10^{x-t}, \quad (1.13)$$

unde  $f_q$  și  $g_q$  satisfac inegalitățile :

$$1/10 \leq f_q < 1 \quad (1.14)$$

$$0 \leq g_q < 1 \quad (1.15)$$

mantisa  $f_q$  fiind compusă din  $t$  cifre.

Prin rotunjire înțelegem influența pe care o are valoarea lui  $g_q$  asupra lui  $q$ .

Calculatoarele care nu au cifre de gardă, neglijează valoarea lui  $g_q$ , așa încât rezultatul este  $q = f_q \cdot 10^x$ . În acest caz  $f_q$  nu se modifică și calculatorul a realizat o trunchiere, numită tăiere, pentru a nu se confunda cu erorile de trunchiere. În acest caz eroarea absolută este dată de expresia  $|e_q| = |g_q| \cdot 10^{x-t}$  iar o limită maximă a erorii relative se poate calcula astfel :

$$\varepsilon_q = \left| \frac{e_q}{q} \right| = \left| \frac{g_q \cdot 10^{x-t}}{f_q \cdot 10^x} \right| \leq \left| \frac{10^{x-t}}{10^{-1} \cdot 10^x} \right| = 10 \cdot 10^{-t} = 10^{1-t} \quad (1.16) \text{ Se}$$

observă că limita maximă a erorii relative de rotunjire depinde numai de numărul de cifre al mantisei numărului. Acest tip de rotunjire introduce o eroare mare, dar timpul de calcul al calculatorului scade.

Un alt tip de rotunjire în cazul existenței cifrelor de gardă este rotunjirea simetrică sau obișnuită. În acest caz valoarea aproximativă prin rotunjire a numărului  $q$  este dată de expresia :

$$|\bar{q}| = \begin{cases} |f_q| \cdot 10^x & \text{pentru } |g_q| < 1/2 \\ |f_q| \cdot 10^x + 10^{x-t} & \text{pentru } |g_q| \geq 1/2 \end{cases} \quad (1.17)$$

$$\text{Dacă } |g_q| < 1/2, \text{ eroarea absolută este } |e_q| \leq (1/2) \cdot 10^{x-t} \quad (1.18)$$

$$\text{și pentru } |g_q| \geq 1/2, \text{ eroarea absolută este } |e_q| \leq (1/2) \cdot 10^{x-t} \quad (1.19)$$

$$\text{Deci în ambele cazuri } |e_q| \leq (1/2) \cdot 10^{x-t} \quad (1.20)$$

Limita maximă a erorii relative o putem calcula astfel :

$$\left| \varepsilon_q \right| = \left| \frac{e_q}{\bar{q}} \right| \leq \left| \frac{0.5 \cdot 10^{x-t}}{10^{-1} \cdot 10^x} \right| = \left| \frac{0.5 \cdot 10^{-t}}{10^{-1}} \right| = 5 \cdot 10^{-t} \quad (1.21)$$

Și în acest caz limita maximă a erorii relative de rotunjire simetrică depinde numai de numărul de cifre al mantisei mărimilor care intră în calcul, pe care l-am considerat același pentru toate mărimile și l-am notat cu  $t$ . Se observă că eroarea simetrică (1.21) este cu un ordin de mărime mai mică decât eroarea de tăiere (1.16). Pentru o precizie mai mare a calculelor se utilizează rotunjirea simetrică.

Aceleași limite maxime ale erorii relative le obținem și pentru cazul reprezentării sistematizate a numărului  $q$  în baza  $b$  :

$$q = a_1 b^n + a_2 b^{n-1} + a_3 b^{n-2} + \dots + a_m b^{n-m+1} + \dots \quad (1.22)$$

Considerăm  $m$  cifre semnificative ale numărului:

$$q = a_1 b^n + a_2 b^{n-1} + \dots + a_m b^{n-m+1} + \dots + b^{n-m} (a_{m+1} + a_{m+2} b^{-1} + \dots) \quad (1.23)$$

$$\text{Notăm cu } c = (a_{m+1} + a_{m+2} b^{-1} + \dots) \quad (1.24)$$

Dacă  $c < (1/2)q$  atunci  $c = 0$  și valoarea aproximativă prin rotunjire a lui  $q$  este

$$\bar{q} = a_1 b^n + a_2 b^{n-1} + a_3 b^{n-2} + \dots + a_m b^{n-m+1} \quad (1.25) \text{ Dacă } c > (1/2)q \text{ atunci } c = q \text{ și}$$

valoarea aproximativă a lui  $q$  este

$$\bar{q} = a_1 b^n + a_2 b^{n-1} + \dots + (a_m + 1) b^{n-m+1} \quad (1.26) \text{ În}$$

$$\text{ambele cazuri eroarea absolută } |e_q| < (1/2) b^{n-m+1} \quad (1.27)$$

O limită maximă a erorii relative se calculează astfel:

$$|\varepsilon_q| = \left| \frac{e_q}{q} \right| \leq \left| \frac{(1/2) \cdot b^{n-m+1}}{1 \cdot b^n} \right| = (1/2) \cdot b^{1-m}, \quad (1.28)$$

unde  $m$  este numărul de cifre semnificative ale numărului. Pentru baza  $b = 10$  se obține același rezultat ca la scrierea în virgulă mobilă (1.21).

### 1.3. PROPAGAREA ERORILOR

În acest paragraf este studiată propagarea erorilor prin operațiile aritmetice realizate de calculatorul numeric: adunări, scăderi, înmulțiri, împărțiri și extrageri de rădăcină pătrată. Se consideră două numere  $x$  și  $y$  cu valorile aproximative, eroarea absolută și eroarea relativă  $\bar{x}, e_x, \varepsilon_x$ , respectiv  $\bar{y}, e_y, \varepsilon_y$ . În scopul simplificării notațiilor vom folosi în continuare pentru eroarea absolută și eroarea relativă a unei mărimi  $x$ , notația  $e_x$  respectiv  $\varepsilon_x$ . Vom deduce formulele prin care se vede efectul erorilor absolute asupra operanzilor.

Adunarea:  $x + y = \bar{x} + e_x + \bar{y} + e_y = \bar{x} + \bar{y} + e_y + e_x$

$$e_{x+y} = x + y - (\bar{x} + \bar{y}) = e_y + e_x \quad e_{x+y} = e_y + e_x \quad (1.29)$$

Scăderea:  $x - y = \bar{x} + e_x - \bar{y} - e_y = \bar{x} - \bar{y} + e_x - e_y$

$$e_{x-y} = x - y - (\bar{x} - \bar{y}) = e_x - e_y \quad e_{x-y} = e_x - e_y \quad (1.30)$$

Înmulțirea:  $x \cdot y = (\bar{x} + e_x) \cdot (\bar{y} + e_y) = \bar{x} \cdot \bar{y} + e_x \cdot \bar{y} + e_y \cdot \bar{x} + e_x \cdot e_y$

Considerând  $e_x \cdot e_y \approx 0$  rezultă:

$$e_{xy} = xy - \bar{x}\bar{y} = e_x \bar{y} + e_y \bar{x} \quad e_{xy} \cong e_x \bar{y} + e_y \bar{x} \quad (1.31)$$

Împărțirea:  $\frac{x}{y} = \frac{\bar{x} + e_x}{\bar{y} + e_y} = \frac{\bar{x} + e_x}{\bar{y}(1 + e_y/\bar{y})} = \frac{\bar{x} + e_x}{\bar{y}} \left( 1 - \frac{e_y}{\bar{y}} + \frac{e_y^2}{\bar{y}^2} - \dots \right) \cong$

$$\frac{\bar{x} + e_x}{\bar{y}} \left( 1 - \frac{e_y}{\bar{y}} \right) = \frac{\bar{x}}{\bar{y}} + \frac{e_x}{\bar{y}} - \frac{\bar{x}e_y}{\bar{y}^2} - \frac{e_x e_y}{\bar{y}^2}.$$

În calcul s-a utilizat dezvoltarea în serie geometrică a factorului  $1 / \left( 1 + \frac{e_y}{\bar{y}} \right), \left| \frac{e_y}{\bar{y}} \right| < 1$  și s-au reținut primii doi termeni. Considerând  $e_x \cdot e_y \approx 0$  rezultă:

$$e_{x/y} = e_x/\bar{y} - \bar{x} \cdot e_y/\bar{y}^2 \quad (1.32)$$

Rădăcina pătrată:

$$\sqrt{x} = \sqrt{\bar{x} + e_x} = \sqrt{\bar{x}(1 + e_x/\bar{x})} = \sqrt{\bar{x}}(1 + (1/2)(e_x/\bar{x}) + \dots)$$

În calcul s-a utilizat dezvoltarea în serie binomială a factorului  $\sqrt{1 + e_x/\bar{x}}$

$$e_{\sqrt{x}} = \sqrt{x} - \sqrt{\bar{x}} = (1/2)(e_x/\bar{x})\sqrt{\bar{x}} \quad (1.33)$$

Propagarea erorilor relative prin operațiile realizate de calculator se deduce din formulele de propagare a erorilor absolute.

Adunarea:

$$\varepsilon_{x+y} = \frac{e_{x+y}}{x+y} = \frac{\bar{x}}{x+y} \cdot \frac{e_x}{x} + \frac{\bar{y}}{x+y} \cdot \frac{e_y}{y} = \frac{\bar{x}}{x+y} \cdot \varepsilon_x + \frac{\bar{y}}{x+y} \cdot \varepsilon_y$$

$$\varepsilon_{x+y} = \frac{\bar{x}}{x+y} \cdot \varepsilon_x + \frac{\bar{y}}{x+y} \cdot \varepsilon_y \quad (1.34)$$

Scăderea:

$$\varepsilon_{x-y} = \frac{e_{x-y}}{x-y} = \frac{\bar{x}}{x-y} \cdot \frac{e_x}{x} - \frac{\bar{y}}{x-y} \cdot \frac{e_y}{y} = \frac{\bar{x}}{x-y} \cdot \varepsilon_x - \frac{\bar{y}}{x-y} \cdot \varepsilon_y$$

$$\varepsilon_{x-y} = \frac{\bar{x}}{x-y} \cdot \varepsilon_x - \frac{\bar{y}}{x-y} \cdot \varepsilon_y \quad (1.35)$$

Înmulțirea:

$$\varepsilon_{xy} = \frac{e_{xy}}{xy} = \frac{\bar{x} \cdot e_y}{xy} + \frac{\bar{y} \cdot e_x}{xy} = \frac{e_y}{\bar{y}} + \frac{e_x}{\bar{x}} = \varepsilon_x + \varepsilon_y$$

$$\varepsilon_{xy} = \varepsilon_x + \varepsilon_y \quad (1.36)$$

Împărțirea:

$$\varepsilon_{x/y} = \varepsilon_{x/y} / (x/y) = \left( (1/\bar{y}) \cdot e_x - (\bar{x}/\bar{y}^2) \cdot e_y \right) / (\bar{x}/\bar{y}) = e_x/\bar{x} - e_y/\bar{y} = \varepsilon_x - \varepsilon_y$$

$$\varepsilon_{x/y} = \varepsilon_x - \varepsilon_y \quad (1.37)$$

Rădăcina pătrată:

$$\varepsilon_{\sqrt{x}} = (1/2) \cdot \sqrt{\bar{x}}(e_x/\bar{x}) / \sqrt{\bar{x}} = (1/2) \cdot (e_x/\bar{x}) = (1/2) \cdot \varepsilon_x$$

$$\varepsilon_{\sqrt{x}} = (1/2) \varepsilon_x \quad (1.38)$$

Formulele de calcul al propagării erorilor relative prin operațiile realizate de calculator sunt simetrice și din aceste motive se pot realiza grafuri pentru calculul lor. Aceste grafuri se numesc grafuri de procedură.

## 1.4. GRAFURI DE PROCEDURĂ

Aceste grafuri sunt binare, adică într-un nod intră cel mult două mărimi și iese o singură mărime. Pentru fiecare operație aritmetică marca grafului se calculează diferit. Marca grafului pentru operația de adunare este raportul între valoarea din nodul care se însumează și suma

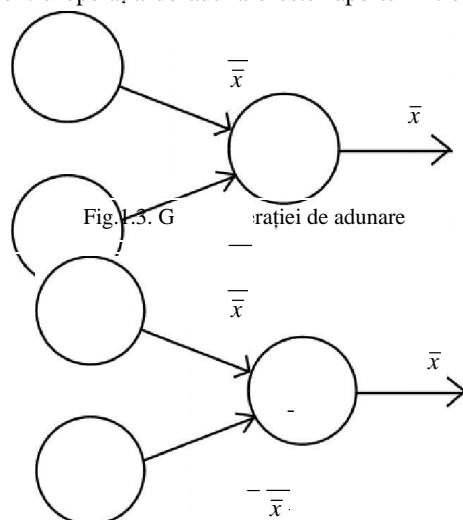


Fig.1.4. Graful operației de scădere

operația corespunzătoare expresiei a cărei valoare a erorii relative o calculăm, pentru a determina marca ramurii grafului. Pentru operațiile de înmulțire, împărțire și ridicare la putere marca grafului este constantă indiferent de valoarea obținută în nodul grafului. Marca grafului

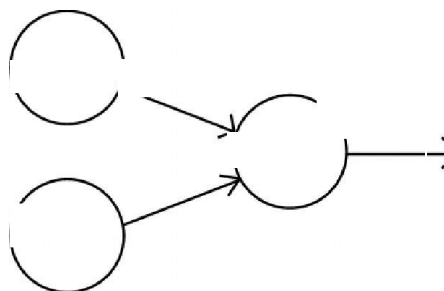


Fig.1.5. Graful operației de înmulțire

valorilor din nodurile care se însumează, procedeul fiind prezentat prin graful din fig.1.3. Se dau valorile erorilor relative  $\varepsilon_x$  și  $\varepsilon_y$  ale mărimilor care se însumează și eroarea de rotunjire  $\varepsilon_r$ .

Eroarea la ieșirea grafului este

$$\varepsilon_{x+y} = \frac{\bar{x}}{\bar{x} + \bar{y}} \varepsilon_x + \frac{\bar{y}}{\bar{x} + \bar{y}} \varepsilon_y + \varepsilon_r$$

(1.39)

Se observă egalitatea expresiei erorii (1.39) cu cea obținută la propagarea erorii prin adunare prezentată în paragraful 1.3.

Eroarea în nodul grafului se calculează după modul expus anterior și nodul poate deveni la rândul său un termen sau un factor al următoarei operații.

În calculul erorii se ține seama de valoarea obținută în fiecare nod prin

calculăm, pentru a determina marca ramurii grafului. Pentru operațiile de înmulțire, împărțire și ridicare la putere marca grafului este constantă indiferent de valoarea obținută în nodul grafului. Marca grafului pentru scăzut la operația de scădere se calculează ca raportul între valoarea scăzutului și diferența valorilor scăzutului și scăzătorului, iar pentru scăzător se calculează ca raportul dintre valoarea scăzătorului și diferența valorilor scăzutului și scăzătorului, luat cu semnul minus. Eroarea la ieșirea grafului este dată în formula (1.40), iar graful este prezentat în fig. 1.4.

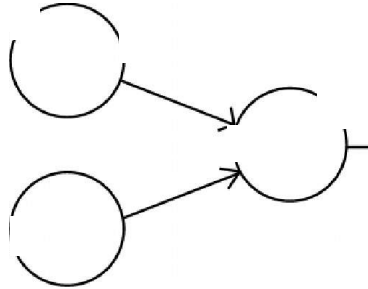


Fig.1.6. Graful operației de împărțire

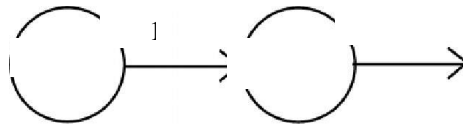


Fig.1.7. Graful operației de extragere de rădăcină pătrată

$$\varepsilon_{x-y} = \frac{\bar{x}}{\bar{x}-\bar{y}} \varepsilon_x - \frac{\bar{y}}{\bar{x}-\bar{y}} \varepsilon_y + \varepsilon_r$$

(1.40) Marca grafului pentru operația de înmulțire este totdeauna constantă egală cu unitatea atât pentru înmulțitor cât și pentru deînmulțit. Graful pentru această operație este prezentat în figura 1.5. Eroarea la ieșirea grafului este dată de expresia următoare:  $\varepsilon_{xy} = \varepsilon_x + \varepsilon_y + \varepsilon_r$

(1.41)

Marca grafului pentru operația de împărțire este totdeauna constantă, având valoarea 1 pentru deîmpărțit și -1 pentru împărțitor. În figura 1.6 este prezentat graful pentru operația aritmetică de împărțire. Eroarea la ieșirea grafului este

$$\varepsilon_{x/y} = \varepsilon_x - \varepsilon_y + \varepsilon_r \quad (1.42)$$

Operația de extragere a rădăcinii pătrate are marca grafului constantă egală cu 1/2 și

este prezentată în figura 1.7. Eroarea mărimii la ieșirea grafului este

$$\varepsilon_{\sqrt{x}} = (1/2)\varepsilon_x + \varepsilon_r \quad (1.43)$$

Se observă că expresia erorilor relative obținute la ieșirea grafulor sunt chiar formulele de calcul ale erorilor relative propagate prin operațiile aritmetice obținute în paragraful 1.3, iar mărcile grafului sunt identice cu expresiile cu care se înmulțesc erorile relative ale componentelor care intervin în operația de calcul. Pentru ușurința calculului mărcilor grafului se ține seama de modul de calcul prezentat anterior pentru fiecare operație aritmetică. Cunoscând erorile relative ale mărimilor care intră într-o expresie în care avem operațiile de bază ale aritmeticii, putem să determinăm eroarea finală a expresiei cu ajutorul grafulor de procedură.

## 1.5. REGULI PENTRU MĂRIREA PRECIZIEI CALCULELOR

Pentru mărirea preciziei calculelor în aplicațiile practice, este bine să ținem seama de următoarele reguli de calcul:

1. În operațiile de adunare și scădere să se înceapă cu cele mai mici numere.
2. Să se evite scăderea a două numere aproximativ egale, prin rescrierea expresiei de calcul.
3. Expresiile de forma  $(a-b) \cdot c$  și  $(a-b)/c$  pot fi rescrise sub forma  $ac - ab$  respectiv  $(a/c) - (b/c)$ . Dacă numerele  $a$  și  $b$  sunt aproximativ egale, este bine să se facă întâi scăderea după aceea înmulțirea sau împărțirea.
4. Dacă nici una din regulile precedente nu pot fi aplicate, pentru minimizarea erorii se va căuta realizarea unui număr cât mai mic de operații.



## 1.6. APLICAȚII

1. Să se determine o limită superioară a erorii relative pentru expresia :

$$E(x) = (ax^2 + bx + c) / (dx + f)$$

în punctul  $x_0$  știind că  $a, b, c, d, f, x_0$  sunt pozitive și au respectiv erorile relative  $\varepsilon_a, \varepsilon_b, \varepsilon_c, \varepsilon_d, \varepsilon_f, \varepsilon_{x_0}$ .

Se cunosc erorile relative ale tuturor elementelor funcției date. Pentru calculul limitei superioare a erorii relative, se realizează graful de procedură al expresiei date, care este prezentat în figura 1.8. Ținând cont de grafurile operațiilor aritmetice, se începe realizarea grafului expresiei  $\varepsilon_E$  cu graful pentru numărător și după aceea cu graful pentru numitor realizând în final un graf pentru operația de împărțire.

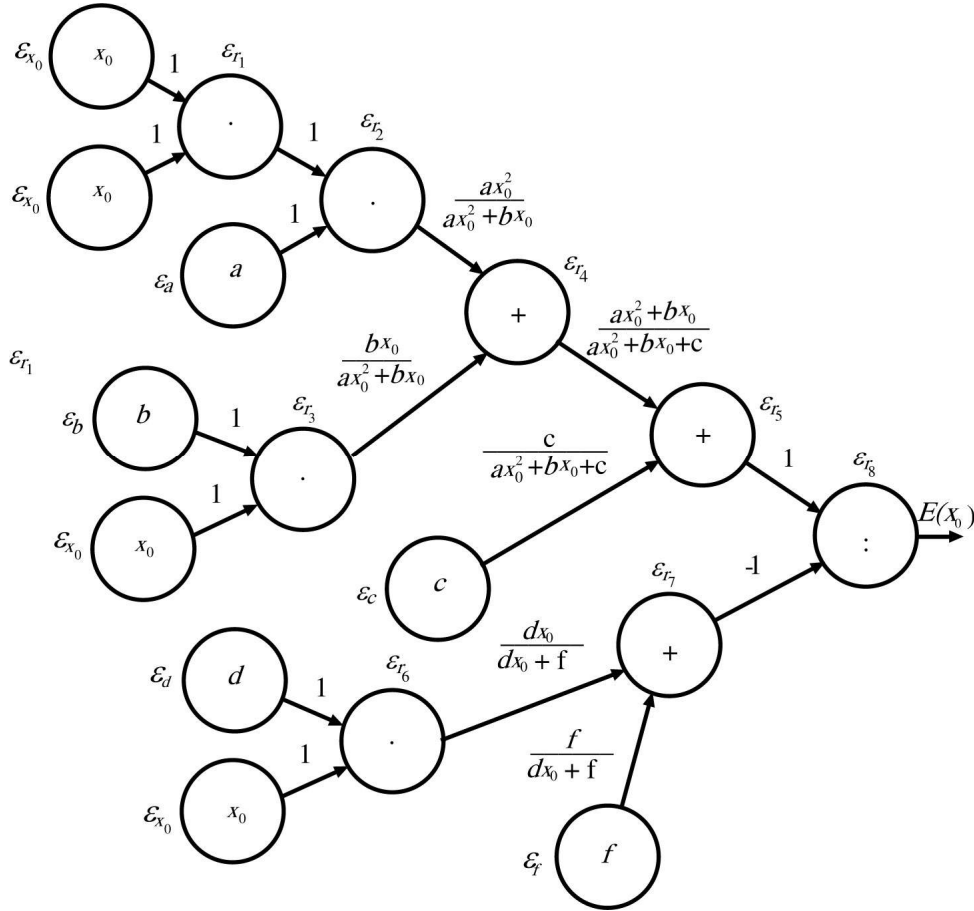
Mărcile grafurilor operațiilor care intervin în expresie se calculează conform expresiilor (1.39), (1.40), (1.41), (1.42) și (1.43). Pentru fiecare operație de calcul se dă eroarea de rotunjire notată cu  $\varepsilon_{r_i}, i=1,2,\dots,8$ .

Erorile din fiecare punct al grafului se înmulțesc cu marca grafului corespunzător iar la ieșire, în punctul în care are loc o operație aritmetică, se însumează, indiferent de operația aritmetică care are loc în nod.

$$\begin{aligned} e_E = & \left[ (\varepsilon_{x_0} + \varepsilon_{x_0} + \varepsilon_{r_1} + \varepsilon_a + \varepsilon_{r_2}) \frac{ax_0^2}{ax_0^2 + bx_0 + c} + (\varepsilon_b + \varepsilon_{x_0} + \varepsilon_{r_3}) \frac{bx_0}{ax_0^2 + bx_0} + \varepsilon_{r_4} \right] \\ & \cdot \frac{ax_0^2 + bx_0}{ax_0^2 + bx_0 + c} + \varepsilon_c \frac{c}{ax_0^2 + bx_0 + c} + \varepsilon_{r_5} + \left[ (\varepsilon_d + \varepsilon_{x_0} + \varepsilon_6) \frac{dx_0}{dx_0 + f} (-1) \right] + \\ & + \left( \varepsilon_f \frac{f}{dx_0 + f} + \varepsilon_{r_7} \right) (-1) + \varepsilon_{r_8} \end{aligned}$$

Dacă considerăm că erorile lui  $a, b, c, d, f, x_0$  sunt obținute ca erori de rotunjire, atunci toate erorile din expresie pot fi majorate cu  $5 \cdot 10^{-t}$ , unde  $t$  este numărul de cifre al variabilelor  $a, b, c, d, f, x_0$  admis de calculator. Ca urmare, expresia limitei erorii relative devine:

$$\varepsilon_E \leq 5 \cdot 10^{-t} \left( \frac{5ax_0^2 + 3bx_0}{ax_0^2 + bx_0 + c} + 5 + \frac{2dx_0}{dx_0 + f} \right)$$

Fig.1.8. Graful pentru calculul erorii relative a expresiei  $E(x)$ 

Considerând că  $x_0 \in [0,1]$ , atunci expresia erorii relative se poate simplifica făcând o nouă majorare și anume înlocuim pe  $x_0$  în expresiile de la numărător cu 1, iar în cele de la numitor cu zero. Marginea erorii relative pentru expresia dată în condițiile impuse devine:

$$\varepsilon_E \leq 5 \cdot 10^{-t} \left( \frac{5a+3b}{c} + \frac{2d}{f} + 5 \right)$$

Se observă că limita erorii relative a expresiei  $E(x)$  depinde de coeficienții  $a, b, c, d, f$  și de  $t$ , mantisa admisă de calculator. S-a considerat că erorile datelor de intrare sunt erori de rotunjire. Când coeficienții nu sunt pozitivi trebuie să se utilizeze valorile absolute ale lor.

2. Se consideră expresiile  $u = a^3 + 3a^2 + 3a + 1$  și  $v = (a+1)^3$  unde  $a$  este un număr pozitiv rotunjit în mod simetric și presupunem că 1 și 3 nu au erori. Să se arate că pentru  $a$

foarte mare marginile erorii relative a lui  $u$  și  $v$  devin aproximativ egale iar pentru  $a$  foarte mic marginea erorii relative a lui  $u$  devine aproximativ de cinci ori mai mică decât marginea erorii relative a lui  $v$ . Numărul maxim de cifre al mantisei admise de calculator este  $t$ . Să se calculeze marginile erorii relative ale expresiilor date și pentru cazul când  $a$  este exact.

Construcția grafurilor de procedură pentru cele două expresii, precum și calculul limitelor superioare rămâne ca un exercițiu pentru cititor.

$$R: \quad a \rightarrow \infty \Rightarrow \varepsilon_u \leq 40 \cdot 10^t; \varepsilon_v \leq 40 \cdot 10^t$$

$$a \rightarrow 0 \Rightarrow \varepsilon_u \leq 5 \cdot 10^t; \varepsilon_v \leq 25 \cdot 10^t$$

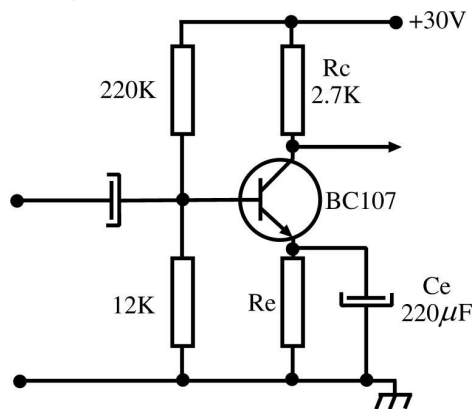


Fig.1.9. Etaj de amplificare cu un tranzistor

3. Să se determine o limita a erorii amplificării etajului din figura 1.9, funcție de toleranțele componentelor pasive electronice. Se consideră că rezistoarele au toleranța de 5% iar condensatoarele de 10%. Amplificarea de tensiune a montajului este dată de expresia:

$$A_u = -\frac{R_s \parallel R_c}{R_e + h_{11b}}, \text{ unde } h_{11b} = \frac{30}{I_E [mA]}$$

$R_s$  = rezistență de sarcină.

Pentru circuitul electronic dat, se poate considera  $R_s = \infty$  și  $R_e = 0$ , iar expresia simplificată a amplificării în tensiune devine:  $A_u = -R_c / h_{11b}$ .

Graful de procedură pentru expresia obținută este prezentat în figura 1.10 și este identic cu cel pentru operația de împărțire. Mărcile grafului pentru cele două mărimi care intră în formula de calcul sunt 1 pentru deîmpărțit și -1 pentru împărțitor. Eroarea calculată pentru amplificare nu include și eroare datorată neglijării rezistențelor de sarcină și emitor. Se calculează o limită a erorii amplificării circuitului electronic datorată toleranțelor componentelor pasive.

Considerând eroarea de rotunjire foarte mică față de eroarea introdusă de toleranța componentelor electronice, rezultă următoarea expresie de calcul:

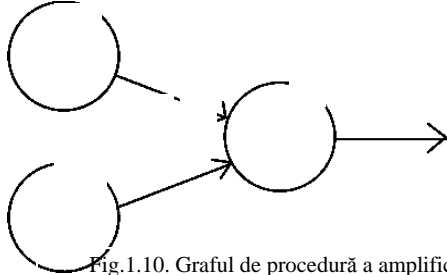


Fig. 1.10. Graful de procedură a amplificării în tensiune

$$\varepsilon_A = \varepsilon_{R_c} - \varepsilon_{h_{11}} + r \leq \varepsilon_{R_c} + \varepsilon_{h_{11}} + r =$$

$$0,05 + 0,05 + 0,5 \cdot 10^{-1} \approx 0,1$$

Amplificarea în tensiune a circuitului electronic poate avea o limită maximă a erorii de 10%.

Analog, se pot calcula erorile maxime pe care le au diferiți parametri ai circuitelor electronice, funcție de erorile elementelor de care depind.

4. Fie  $a, b$  și  $x$  trei numere pozitive și exacte. Să se traseze grafurile de procedură și să se calculeze marginile erorilor relative de rotunjire pentru expresiile  $u = ax + bx^2$  și  $v = x(a + bx)$ . Să se compare cele două margini ale erorilor, obținute. Să se calculeze marginile erorilor relative și pentru cazul când cele trei numere au erori de rotunjire.

$$R: \varepsilon_u \leq 5 \cdot 10^{-t} (3bx + 2a) / (bx + a), \varepsilon_v \leq 5 \cdot 10^{-t} (3bx + 2a) / (bx + a)$$

$$\varepsilon_u \leq 10^{-t+1} (3bx + 2a) / (bx + a), \varepsilon_v \leq 10^{-t+1} (3bx + 2a) / (bx + a)$$

5. Se consideră un circuit  $R, L, C$  alimentat de la o sursă  $U$ . Știind că frecvența sursei are variația  $\varepsilon_f$ , iar  $R, L, C$  au variațiile  $\varepsilon_R, \varepsilon_L, \varepsilon_C$ , să se traseze graful impedanței circuitului și să se determine o margine a erorii relative a ei.

6. Presupunem că  $x$  și  $y$  sunt două numere pozitive rotunjite simetric. Să se traseze grafurile de procedură ale expresiilor  $u = (x + x + x + x) \cdot y$  și  $v = 4xy$  și să se arate că marginea erorii relative a lui  $u$  este mai mare decât marginea erorii relative a lui  $v$ .

# 2

## REZOLVAREA NUMERICĂ A ECUAȚIILOR ALGEBRICE\*

Până la ecuațiile de gradul patru, algebra dispune de metode și formule pentru calculul soluțiilor acestora. Pentru ecuațiile de grad mai mare ca patru, cu rădăcini iraționale, determinarea acestora se poate face numai prin aproximații.

Calculul rădăcinilor unei ecuații se face în două etape: 1) separarea rădăcinilor, 2) calculul lor cu o eroare impusă.

### 2.1. SEPARAREA RĂDĂCINILOR

Considerăm funcția  $f:[a,b] \rightarrow \mathbf{R}$ ,  $x \in [a,b]$  și ecuația algebrică  $f(x) = 0$ . (2.1)

Separarea rădăcinilor unei ecuații  $f(x) = 0$  constă în determinarea unor intervale în domeniul de definiție al funcției, în care să existe o singură rădăcină reală. Pentru separarea rădăcinilor reale există mai multe metode dintre care amintim: metoda șirului lui Rolle, metoda șirului lui Sturm și metoda lui Budan-Fourier.

#### 2.1.1. METODA ȘIRULUI LUI ROLLE

Această metodă se bazează pe o consecință a teoremei lui Rolle care afirmă că: *între două rădăcini consecutive ale derivatei  $f'(x) = 0$  există cel mult o rădăcină reală a ecuației  $f(x) = 0$* . Există cu siguranță o soluție între rădăcinile consecutive ale derivatei  $f'(x) = 0$ , dacă produsul valorilor funcției pentru cele două rădăcini consecutive ale derivatei este negativ.

Considerăm  $x_1 < x_2 < x_3 < \dots < x_n$ , rădăcinile ecuației  $f'(x) = 0$ . Se construiește șirul lui Rolle:

$$f(a), f(x_1), f(x_2), \dots, f(x_n), f(b) \quad (2.2)$$

Numărul de variații de semn din șirul lui Rolle reprezintă numărul de rădăcini reale ale ecuației  $f(x) = 0$ , iar rădăcinile consecutive ale derivatei pentru care avem variația de semn a șirului reprezintă intervalele în care există o rădăcină reală a ecuației (2.1). Dacă  $f(x)$  este definită pe întreaga axă reală,  $\mathbf{R}$ , șirul lui Rolle conține

---

\*) *Bibliografie:* [3],[6],[7],[11],[21].

și termenii cu valorile funcției la  $+\infty$  și  $-\infty$ :

$$f(-\infty), f(x_1), f(x_2), \dots, f(x_n), f(+\infty). \quad (2.3)$$

0..Pentru aplicarea acestei metode trebuie să determinăm soluțiile ecuației  $f'(x) = 0$ . Pentru ecuații de grad mai mic metoda este avantajoasă, dar pentru ecuații de grad mare rezolvarea ecuației derivatei poate deveni tot atât de dificilă ca și rezolvarea ecuației date  $f(x) = 0$ .

## 2.1.2. METODA ȘIRULUI LUI ȘTURM

Considerăm funcția definită în (2.1) care îndeplinește condițiile de continuitate și derivabilitate pentru  $x \in [a, b]$ .

**Definiția 2.1** Șirul de funcții  $f_0, f_1, f_2, \dots, f_m$  continue pe  $[a, b]$  care satisfac condițiile:

- 1)  $f_0(x) = f(x)$ ;
- 2)  $f_m(x) \neq 0$  pentru  $x \in [a, b]$ ;
- 3) dacă  $f_i(x) = 0$ ,  $1 \leq i \leq m-1$  și  $x \in [a, b]$ , atunci

$$f_{i-1}(x) \cdot f_{i+1}(x) < 0;$$

- 4) dacă  $f_0(x) = 0$  pentru  $x \in (a, b)$ , atunci  $f'_0(x) \cdot f_1(x) > 0$

se numește șir Șturm asociat funcției  $f(x)$ .

Numărul rădăcinilor ecuației  $f(x)$  în intervalul  $[a, b]$  este dat de următoarea teoremă

**Teorema 2.1.** Fie șirul Șturm  $f_0, f_1, f_2, \dots, f_m$ , atașat funcției  $f(x)$  cu condițiile  $f(a) \neq 0$  și  $f(b) \neq 0$ , atunci numărul de rădăcini ale ecuației  $f(x) = 0$  în intervalul  $[a, b]$  este dat de diferența numărului de variații de semn ale șirurilor:

$$f_0(a), f_1(a), \dots, f_m(a) \quad (2.4)$$

$$f_0(b), f_1(b), \dots, f_m(b) \quad (2.5) \text{ În cazul funcției polinom}$$

$P(x)$  care este definită pe  $\mathbf{R}$ , teorema (2.1) devine:

**Teorema 2.2** Fie  $P_0, P_1, P_2, \dots, P_m$  un șir de polinoame construit astfel

$P_0 = P$ ,  $P_1 = P'$ , iar  $P_{i+1}$  este restul împărțirii lui  $P_{i-1}$  la  $P_i$  luat cu semn schimbat, pentru  $2 \leq i \leq m$ . Atunci numărul de rădăcini ale ecuației  $P(x) = 0$  este egal cu diferența dintre numărul de schimbări de semn ale șirurilor:

$$P_0(-\infty), P_1(-\infty), P_2(-\infty), \dots, P_m(-\infty) \quad (2.6)$$

$$P_0(\infty), P_1(\infty), P_2(\infty), \dots, P_m(\infty) \quad (2.7)$$

*Demonstrație.* Trebuie să arătăm că șirul format este un șir Șturm. Condițiile șirului lui Șturm sunt îndeplinite. Prima condiție este îndeplinită din construcția șirului. A doua condiție este îndeplinită deoarece considerăm că polinomul nu are rădăcini multiple, deci  $P(x)$  și  $P'(x)$  nu au rădăcini multiple. Construim termenii șirului folosind algoritmul lui Euclid. Se schimbă doar semnul restului.

$$P_{i-1}(x) = P_i(x) \cdot Q_i(x) - P_{i+1}(x); i = 1, 2, \dots, m-1 \quad (2.8)$$

Aplicând algoritmul, obținem:  $(P_0(x), P_1(x)) = P_m(x) = \text{constantă} \neq 0$ , deoarece  $P_0(x)$  și  $P_1(x)$  sunt prime între ele. Pentru  $P_i(x) = 0$  din relația (2.8) se vede că este îndeplinită

și condiția 3 din definiția șirului lui Sturm. Dacă  $P(x)=0$  pentru  $x \in \mathbf{R}$ , deoarece  $P(x)$  și  $P'(x)$  nu au rădăcini comune, rezultă că :

$$P'(x) = P_0'(x) = P_1(x) \neq 0 \Rightarrow P_0'(x) \cdot P_1(x) = P_1^2(x) > 0,$$

expresie care arată îndeplinirea condiției 4 din definiția șirului lui Sturm. Pe baza teoremei 2.1, teorema este demonstrată.

În cazul când  $P(x)$  are rădăcini multiple, polinomul  $P_m(x)$  va fi cel mai mare divizor al polinoamelor:  $P(x)$  și  $P'(x)$ . Șirul obținut se împarte la  $P_m(x)$  și se obține șirul Sturm căruia îi putem aplica teorema 2.2 .

După relația de recurență (2.8) s-a realizat un algoritm pentru obținerea unui șir Sturm atașat unui polinom de gradul  $n$ . Algoritmul determină resturile cu semn schimbat din algoritmul lui Euclid aplicat polinomului și derivatei lui. Dacă polinomul și derivata lui sunt prime, ultimul rest este o constantă iar în caz contrar este cel mai mare divizor al lor.

### 2.1.2.1. Algoritmul 2.1

```
{Variable
grad:gradul polinomului, întreg);
P1:vectorul coeficienților polinomului;
P2:vectorul coeficienților derivatei polinomului;
Rez:vectorul coeficienților restului;
C1,C2:coeficienții pentru calculul coeficienților restului, reali;
{
    pentru i=grad-1 până la 0 calculează P2[i]=(i+1)P1[i+1];

    { dacă grad=0 atunci Rez(0)=P1(0)-P1(1)
    /* Restul are gradul grad-2 */
    altfel
    {
        calculează C1:=P1[grad]/P2[grad-1];
        calculează C2:=(P1[grad-1]-C1*P2[grad-2])/P2[grad-1];
        pentru i=grad-2 până la 1 calculează
            Rez[i]=P1[i]-C1*P2[i-1]-C2*P2[i];
        calculează Rez[0]=P1[0]-C1*P2[0]-C2*P2[0];
    }
    }
    Tipărește coeficienții restului cu semnul schimbat;
}
}
```

### 2.1.2.2. Implementarea algoritmului 2.1

```

/*Funcția care implementează calculul coeficienților primei derivate
a polinomului.
*/
void DerPoli( int grad,
             double *coef,
             double *rez)
{ int i;
  For (i=grad-1;i>=0;i--) rez[i]=(i+1)*coef[i+1];
}
/* Funcția care implementează termenii șirului lui Șturm pentru
polinom
Funcția întoarce
    0 dacă nu se poate realiza șirul
    1 dacă se realizează șirul
*/
int Divi( int grad,
         double *p1,
         double *p2,
         double *rez)
{
  double const1,const2;
  int i;
  if (grad== 1) {
    rez[0]=p1[0]-p1[1];
    return 1;
  }
  else
  {
    const1=p1[grad]/p2[grad-1];
    const2=(p1[grad-1]-const1*p2[grad-2])/p2[grad-1];
    for (i=grad-2;i>=1;i--)rez[i]=p1[i]-const1*p2[i-1]-const2*p2[i];
    rez[0]=p1[0]-const1*p2[0];
    return 0;
  }
}

```

### 2.1.3. METODA BUDAN-FOURIER

Această metodă se bazează pe teorema lui Budan - Fourier și determină numărul rădăcinilor reale ale ecuației polinomiale cu coeficienți reali .

**Teorema 2.3** Fie polinomul  $P(x) = a_0x^n + a_1x^{n-1} + \dots + a_n$  cu  $a_0 \neq 0$  și șirul derivatelor lui:

$$P(x), P'(x), P''(x), \dots, P^{(n)}(x) = a_0 \cdot n! \quad (2.9)$$

Numărul rădăcinilor ecuației  $P(x) = 0$  în intervalul  $(a, b)$ ,  $a, b \in \mathbf{R}$  este dat de



$$N(a, b) = \underline{N}(a) - \overline{N}(b)$$

sau diferă de acesta cu un număr par

$$N(a, b) = N(a) - \overline{N}(b) - 2m,$$

unde  $\underline{N}(a)$  este numărul inferior de schimbări de semn al șirului derivatelor pentru  $x = a$  și  $\overline{N}(b)$  este numărul superior de schimbări de semn al șirului derivatelor pentru  $x = b$ .

Numărul inferior de schimbări de semn al șirului este numărul de schimbări de semn al subșirului termenilor diferiți de zero, iar numărul superior de schimbări de semn al șirului este numărul de schimbări al șirului unde termenii nuli

$$P_{(b)}^{(g)} = P_{(b)}^{(g+1)} = \dots = P_{(b)}^{(g+k+1)} = 0 \quad \left( P_{(b)}^{(g-1)} \neq 0; P_{(b)}^{(g+k)} \neq 0 \right),$$

se înlocuiesc cu elementele  $P_{(b)}^{(g+j)}$   $j = 0, 1, \dots, k-1$  cărora li se atribuie semnul

$$\text{sign } P_{(b)}^{(g+j)} = (-1)^{k-j} \text{sign } P_{(b)}^{(g+k)}$$

Această metodă are dezavantajul că nu determină precis numărul de rădăcini reale ale ecuației polinomiale.

Dintre toate metodele de separare a rădăcinilor reale ale unei ecuații algebrice cea mai utilizată este metoda șirului lui Șturm, iar pentru ecuații de grad mai mic ca patru se poate utiliza cu mult succes și metoda șirului lui Rolle.

## 2.2. CALCULUL RĂDĂCINILOR REALE ALE ECUAȚIEI ALGEBRICE

Pentru calculul rădăcinilor reale ale unei ecuații algebrice se utilizează mai multe metode numerice dintre care amintim: metoda bisecției (bipartiției), metoda aproximațiilor succesive, metoda lui Newton - Raphson, metoda lui Lobacevski și metoda lui Bairstow.

### 2.2.1. METODA BISECȚIEI (BIPARTIȚIEI)

Fie funcția continuă  $f: [a, b] \rightarrow \mathbf{R}$  și ecuația  $f(x) = 0$  care are soluție unică pe intervalul  $[a, b]$ . Pentru condițiile date funcția satisface inecuația  $f(a) \cdot f(b) \leq 0$ . Se pune problema determinării soluției  $\alpha$  a ecuației  $f(x) = 0$  pe intervalul  $[a, b]$  (fig. 2.1), cu o anumită eroare  $\varepsilon$ .

Metoda constă în a verifica dacă  $a$  sau  $b$  sunt soluții ale ecuației  $f(x) = 0$ . Dacă  $a$  și  $b$  nu sunt soluții, se trece la înjumătățirea intervalului  $[a, b]$  și se determină valoarea  $x_m = (a + b) / 2$ . Se verifică dacă  $x_m$  este soluție a ecuației. Această verificare poate fi făcută prin compararea  $|f(x_m)| < \varepsilon$  sau compararea  $|b_n - a_n| < \varepsilon$ , unde  $\varepsilon$  este eroarea de calcul a soluției ecuației. Dacă  $x_m$  este soluție a ecuației, problema s-a rezolvat; în

caz contrar se evaluează  $f(a), f(x_m)$  și se verifică dacă produsul  $f(a) \cdot f(x_n) < 0$ . Dacă inegalitatea este satisfăcută  $b_1 = x_m$  și  $a = a_1$  și soluția se caută în intervalul  $[a_1, b_1]$ . Dacă inegalitatea nu este satisfăcută,  $a_1 = x_m$  și  $b = b_1$  și soluția se află în intervalul  $[a_1, b_1]$ . Procedul continuă și se obțin două șiruri convergente: unul monoton crescător mărginit superior de  $b$  și altul monoton descrescător mărginit inferior de  $a$ :

$$\begin{aligned} a &< a_1 < a_2 < \dots < a_n < \dots \\ b &> b_1 > b_2 > \dots > b_n > \dots \end{aligned} \quad (2.10)$$

Șirul lungimilor intervalelor care se obțin prin înjumătățire este:

$$b_1 - a_1 = (b - a)/2, \quad b_2 - a_2 = (b - a)/2^2, \quad \dots, \quad b_n - a_n = (b - a)/2^n,$$

un șir descrescător cu limita zero.

$$\lim_{n \rightarrow \infty} \frac{b_n - a_n}{2^n} = 0 \quad (2.11)$$

$$\lim_{n \rightarrow \infty} b_n = \alpha \quad (2.12)$$

$$\lim_{n \rightarrow \infty} a_n = \alpha \quad (2.13)$$

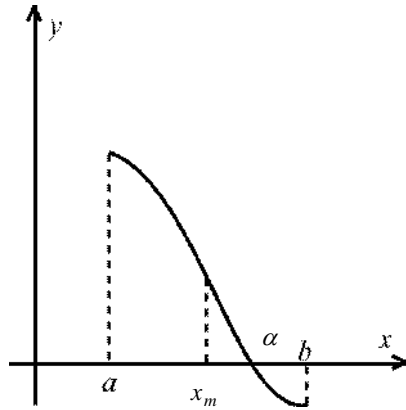


Fig.2.1. Graficul funcției  $y = f(x)$  pe intervalul  $[a, b]$

Algoritmul de calcul se oprește atunci când este îndeplinită condiția  $|b_n - a_n| < \varepsilon$  unde  $\varepsilon$  este eroarea impusă pentru calculul soluției ecuației date. Soluția ecuației este aproximată cu valoarea mijlocului intervalului  $[a_n, b_n]$ . Lungimea intervalului tinde la zero când  $n$  tinde la infinit, limita capetelor intervalelor fiind un punct a cărui abscisă este soluția ecuației.

### 2.2.1.1. Algoritmul 2.2. Bisecția pentru polinoame

{Variabile

$n$ : gradul polinomului, intreg;

$A$ : coeficienții polinomului, vector;

$ls, ld$ : limitele intervalului, real;

$er$ : eroarea de calcul, real;

$rad$ : rădăcina, real;

$x_m$ : jumătatea intervalului  $[l_s, l_d]$ , real;

// se utilizează funcția Valpol din ANEXA 1//

{

dacă  $Valpol(n, A, ls) * Valpol(n, A, ld) > 0$  atunci

{scrie ecuația nu are rădăcină;

```

    stop;
  }

  dacă Valpol(n,A,ls)=0 atunci
    { rad=ls
    stop;
    }
  dacă Valpol(n,A,ld)=0 atunci
    { rad=ld;
    stop;
    }
  xm=(ls+ld)/2;
  cât timp (abs(ld-ls)>er) și Valpol(n,a,xm)<>0 execută
    { xm=(ls+ld)/2;
    dacă Valpol(n,a,xm)*Valpol(n,a,ls)<0 atunci ld=xm
    altfel ls=xm;
    }
  rad=xm;
}

```

### 2.2.1.2. Implementarea algoritmului 2.2

```

/* Funcția întoarce:
0 în caz de eșec
1 în cazul unui rezultat corect */

int BisecțiePolinom( int grad,
                    double Coef[],
                    double ls,
                    double ld,
                    double eroare,
                    double *rădăcina)
{
  double xm;

  if( ValPol(grad,Coef,ls)*ValPol(grad,Coef,ld)>0 ) return 0;
  if( ValPol(grad,Coef,ls)==0){
    *rădăcina=ls;
    return 1;
  }
  if( ValPol(grad,Coef,ld)==0){
    *rădăcina=ld;
    return 1;
  }
  xm=0.5*(ls+ld);

```

```

while ( (fabs(ld-ls)>eroare) && (ValPol(grad,Coef,xm)!=0) )
{
xm=0.5*(ld+ls);
if( ValPol(grad,Coef,ls)*ValPol(grad,Coef,xm)<0 )ld=xm;
else ls=xm;
}
*rădăcina=xm;
return 1;
}

```

### 2.2.1.3. Algoritmul 2.3. Bisecția pentru ecuații transcendente

```

{ Variabile
(F ecuația, funcție reală de variabilă reală)
ls,ld: limitele intervalului, real;
er:eroarea de calcul, real;
xm: mijlocul intervalului, real;
rad:rădăcina, real;
{
dacă F(ls)*F(ld)>0 atunci { soluția nu exista ,
stop }
altfel
dacă F(ls)=0 atunci {soluția este rad=ls;
stop
}
dacă F(ld)=0 atunci {soluția este rad=ld;
stop
}
xm=(ls+ld)/2;
cât timp (abs(ld-ls)>er) și (F(xm)<>0) executa
{ xm=(ls+ld)/2;
dacă F(xm)*F(ls)<0 atunci ld=xm;
altfel ls=xm;
}
soluția este rad=xm;
}

```

### 2.2.1.4. Implementarea algoritmului 2.3

```

/* Funcția întoarce:
0 în caz de eșec
1 în cazul unui rezultat corect
*/
int BisecțieFuncție(double (*f)(double),
double ls,

```

```

double ld,
double err,
double *soluție)
{
double xm;
if ( f(ls)*f(ld)>0) return 0;
if ( f(ls)==0)
{
*soluție=ls;
return 1;
}
if ( f(ld)==0)
{
*soluție=ld;
return 1;
}
xm=0.5*(ls+ld);
while( ( fabs(ld-ls)>err) && ( f(xm)!=0) )
{
xm=0.5*(ls+ld);
if ( f(ls)*f(xm)<0 ) ld=xm;
else ls=xm;
}
*soluție=xm;
return 1 ;
}

```

### 2.2.2. METODA APROXIMAȚIILOR SUCCESIVE

Fie funcția  $f : [a, b] \rightarrow \mathbf{R}$  continuă și derivabilă pe  $(a, b)$  și ecuația  $f(x) = 0$ , care pe intervalul  $(c, d) \subset (a, b)$  are o rădăcină unică  $\alpha$ ,  $f(\alpha) = 0$ . Presupunem că ecuația  $f(x) = 0$ , se scrie sub forma:  $x = \varphi(x)$  (2.14)

Pentru  $x_0$  o aproximație inițială, avem următoarele succesiuni de aproximații:

$$x_1 = \varphi(x_0), x_2 = \varphi(x_1), x_3 = \varphi(x_2), \dots, x_n = \varphi(x_{n-1}) \quad (2.15)$$

Ca urmare formula  $x_n = \varphi(x_{n-1})$  reprezintă o formulă de iterație. Această formulă de iterație este simplă deoarece valoarea calculată depinde numai de valoarea precedentă și este foarte avantajoasă în programarea pe calculator, deoarece nu consumă multă memorie și timpul de calcul este redus.

Dacă valoarea calculată depinde de toate valorile precedente

$$x_n = \varphi(x_0, x_1, x_2, \dots, x_{n-1}) \quad (2.16)$$

atunci consumul de memorie și timpul de calcul sunt mai mari decât în cazul precedent. Dacă funcția  $\varphi$  din formula de iterație nu depinde de rangul de iterație, atunci iterația este de tip staționar, cazul celor două iterații prezentate.

Formula de iterație poate depinde și de rangul de iterație

$$x_n = \varphi_n(x_0, x_1, x_2, \dots, x_{n-1}) \quad (2.17)$$

și în acest caz rezultă cel mai general mod de iterație.

Vom studia formula de iterație  $x_n = \varphi(x_{n-1})$ . Problema care se pune este convergența șirului (2.15), șir care la limită dă soluția ecuației  $x = \varphi(x)$ , deci a ecuației  $f(x) = 0$ .

Convergența șirului este dată de teorema contracției.

**Definiția 2.2** Aplicația  $T: E \rightarrow E$  se numește contracție dacă există un  $\lambda \in \mathbf{R}$  cu proprietatea  $0 < \lambda < 1$  astfel ca :

$$\rho(T_x, T_y) \leq \lambda \rho(x, y), \quad x, y \in E$$

( $E, \rho$ ) reprezentând un spațiu metric complet, iar  $\rho(x, y)$  distanța definită în spațiul  $E$ .

**Teorema 2.4 (Teorema contracției)**

Fie contracția  $T: E \rightarrow E$  și ( $E, \rho$ ) un spațiu metric complet. Atunci :

1) Aplicația  $T$  este continuă

2) Oricare ar fi  $x_0 \in E$  șirul  $x_0, x_1, x_2, \dots, x_n, \dots$  definit prin relația de recurență  $x_{k+1} = Tx_k$  ( $k = 0, 1, 2, \dots, n, \dots$ ) este convergent. Limita șirului este  $\bar{x}$  și este fixă pentru  $T$ , adică  $T\bar{x} = \bar{x}$  și viteza de convergență este:

$$\rho(x_m, \bar{x}) \leq \frac{\lambda^m}{1-\lambda} \rho(x_0, x_1)$$

3) Punctul  $\bar{x}$  este unic

Fie formula de iterație  $x_n = \varphi(x_{n-1})$ . Considerăm că soluția ecuației  $x = \varphi(x)$  este  $a$ , deci  $a = \varphi(a)$ . Scăzând cele două relații se obține:

$$x_n - a = \varphi(x_{n-1}) - \varphi(a) \quad (2.18)$$

Înmulțim partea dreaptă cu  $(x_{n-1} - a) / (x_{n-1} - a)$  și rezultă

$$x_n - a = \frac{\varphi(x_{n-1}) - \varphi(a)}{x_{n-1} - a} (x_{n-1} - a) \quad (2.19)$$

Aplicând teorema lui Lagrange

$$\frac{\varphi(x_{n-1}) - \varphi(a)}{x_{n-1} - a} = \varphi'(\xi) \quad \text{unde } x_{n-1} < \xi < b$$

avem

$$x_n - a = \varphi'(\xi)(x_{n-1} - a)$$

Dacă luăm valoarea maximă a lui  $|\varphi'(x)|$  în  $(a, b)$  pe care o considerăm egală cu  $\lambda$  atunci

$$\begin{aligned} |x_n - a| &\leq \lambda |x_{n-1} - a| \\ |x_{n-1} - a| &< \lambda |x_{n-2} - a| \Rightarrow |x_n - a| < \lambda^2 |x_{n-2} - a| \\ &\dots\dots\dots \\ |x_n - a| &< \lambda^n |x_0 - a| \end{aligned} \quad (2.20)$$

Dacă  $|\varphi'(x)| < \lambda < 1$  pe întreg intervalul, atunci indiferent de alegerea punctului de start  $x_0$ , șirul 2.15 este convergent, deoarece

$$\lim_{n \rightarrow \infty} |x_n - a| < \lim_{n \rightarrow \infty} \lambda^n |x_0 - a| = 0 \Rightarrow \lim_{n \rightarrow \infty} x_n = a,$$

valoare care reprezintă soluția ecuației 2.14.

Dacă  $|\varphi'(x)| > 1$  atunci  $|x_n - a|$  crește odată cu creșterea lui  $n$ , rezultând divergența șirului  $x_n$ . Aceste cazuri de convergență a metodei aproximațiilor succesive pentru

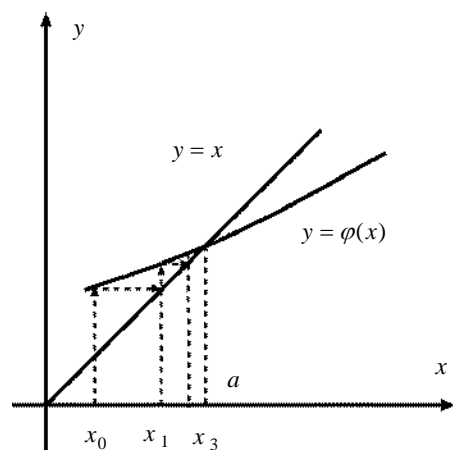


Fig.2.2. Convergența pentru cazul  
 $0 < \varphi'(x) < 1$

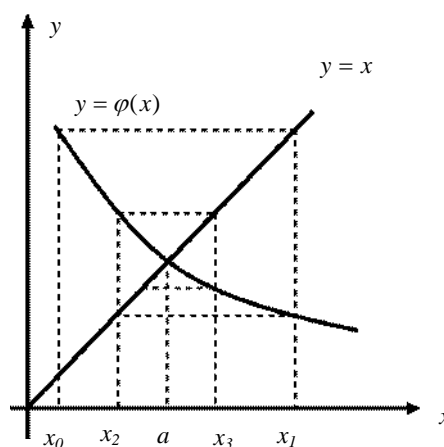


Fig.2.3. Convergența pentru cazul  
 $-1 < \varphi'(x) < 0$

$|\varphi'(x)| < 1$  și de divergență a metodei pentru  $|\varphi'(x)| > 1$  pot fi justificate și prin reprezentări grafice date în figurile: 2.2, 2.3, 2.4, 2.5.

Calculul soluției prin metoda aproximațiilor succesive se face funcție de o eroare impusă care dă criteriul de oprire al programului. Algoritmul se termină când

$$|x_n - x_{n-1}| < \varepsilon \quad (2.21)$$

ceea ce arată că diferența dintre două valori consecutive calculate este mai mică ca  $\varepsilon$

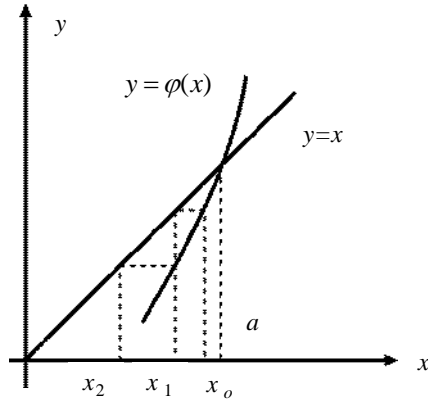


Fig.2.4. Divergența pentru cazul  
 $\phi'(x) > 1$

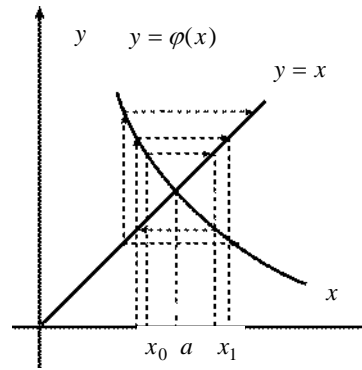


Fig 2.5. Divergența pentru cazul  
 $\phi'(x) < -1$

sau când  $|f(x_n)| < \varepsilon$  (2.22)

și atunci soluția ecuației este aproximată de  $x_n$ , pentru  $\varepsilon$  suficient de mic.

### 2.2.2.1. Algoritmul 2.4. Aproximații succesive

```
{Variabile
(F funcție reală de variabilă reală, ecuația se aduce la  $x=F(x)$ )
x0: punctul de start, real;
pc: punctul de derivare, real;
xn: valoarea curentă a rădăcinii, real;
xn_1: valoarea precedentă a rădăcinii, real;
h: pasul de derivare, real;
der:derivata funcției, real;
ls,ld: limitele intervalului, real;
rad:rădăcina ecuației, real;
{
h=0.0001;
pc=ls;
repetă
    der=(F(pc+h)-F(xc))/h;
    pc=pc+h;
până când (pc>ld) sau der>=1;
dacă der>=1 atunci
    {scrie nu se poate rezolva;
    stop }
xn=x0;
repetă
```



```

    xn_1=xn;
    xn=F(xn_1);
    până când abs(xn-xn_1)<er;
    rad=xn;
    soluția este rad;
}

```

#### 2.2.2.2. Implementarea algoritmului 2.4

```

/* Funcția întoarce:
0 când se găsește rădăcina cu aproximația dorită
1 când nu se îndeplinește condiția de convergență
2 când nu se atinge precizia în numărul de pași impus
*/
int AproxSuc( double (*f)(double),      /* ecuația */
              double ls,                /* limita stângă */
              double ld,                /* limita dreaptă */
              double x0,                /* valoarea de start */
              double err,               /* precizia de aflare a soluției */
              int NMax,                 /* nr. maxim de iterații */
              double *sol               /* soluția */
              )
{
    double xn,xn_1,pct,pas=0.001;
    int iter,sem=1;
    pct=ls;
    do
    {
        if( Derf(f, pct)>=1) sem=0;
        else pct+=pas;
    }
    while( (sem==1) && (pct<=ld) );
    if(sem==0) return 1;
    else
    {
        xn=x0;
        iter=1;
        do
        {
            xn_1=xn;
            xn=f(xn_1);
            iter++;
        }
        while ( ( fabs(xn-xn_1)>err ) && (iter<=NMax) );
        *sol=xn;
    }
}

```

```

if (iter < NMax) return 0;
else return 2;
}

```

### 2.2.3. METODA APROXIMAȚIILOR SUCCESIVE CU VITEZĂ MARE DE CONVERGENȚĂ

Din figura 2.1 se observă că o nouă valoare  $x_{n+1}$  se obține din valoarea calculată  $x_n$  (inițial fiind  $x_0$ ) la care se adaugă o valoare  $\Delta_x$  corespunzător fig. 2.5

$$x_{n+1} = x_n + \Delta_x \quad (2.23)$$

unde  $\Delta_x = f(x_n) - x_n$  (2.24)

Pentru mărirea vitezei de convergență a metodei se pune problema adăugării la  $x_n$  a termenului  $k \Delta_x$  cu  $k > 1$ , astfel ca valoarea  $x_{n+1}$  să fie foarte aproape de soluție și dacă este posibil să fie chiar  $a$ . Deci :

$$x_{n+1} = x_n + k \Delta_x \quad (2.25)$$

Valoarea lui  $k$  se determină cu ajutorul unghiului  $\alpha$  prezentat în fig 2.5 și a segmentului  $\Delta_x$ ,  $k \Delta_x$ ,  $(1-k) \Delta_x$ . Deoarece  $y = x$  (bisectoarea întâi) formează cu axele de coordonate unghiuri de  $45^\circ$ , ABC este un triunghi dreptunghic isoscel, deci  $AB = BC = (1-k) \Delta_x$ . Din triunghiul dreptunghic TBC rezultă:

$$\tan \alpha = BC / TB = (1-k) \Delta_x / (k \Delta_x) = (1-k)/k \quad (2.26) \text{ sau}$$

$$k = 1 / (1 - \tan(\alpha)) \quad (2.27)$$

Tan  $\alpha$  poate fi exprimată cu ajutorul funcției  $\varphi(x)$ :

$$\tan \alpha = \frac{\varphi(a) - \varphi(x_n)}{a - x_n} = \varphi'(\xi) \quad (2.28)$$

unde  $x_n \leq \varphi \leq a$

formulă obținută aplicând teorema lui Lagrange .

Din expresiile (2.27) și (2.28) se obține expresia lui  $k$  sub forma

$$k = \frac{1}{1 - \varphi'(\xi)} \quad (2.29)$$

unde  $\varphi'(\xi)$  poate fi aproximat funcție de valorile cunoscute astfel :

$$\varphi'(\xi) \cong \frac{\varphi(x_n) - x_n}{x_n - x_{n-1}}; \quad x_{n-1} \leq \xi \leq x_n \quad (2.30)$$

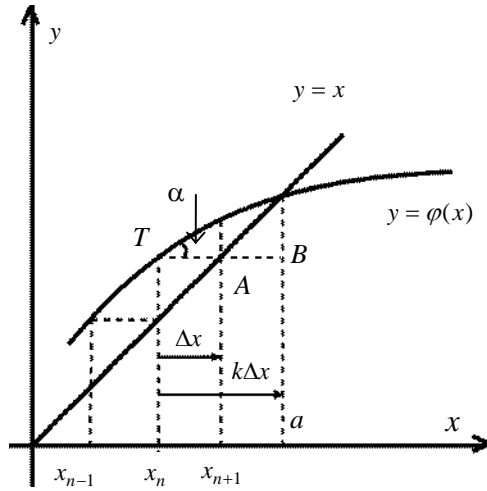


Fig.2.6. Graficul iterației la pasul n

Dacă ținem seama de formulele (2.30), (2.29) și (2.25) putem exprima formula de calcul a lui  $x_{n+1}$  astfel :

$$x_{n+1} = x_n + k(f(x_n) - x_n) \quad (2.31)$$

Influențele lui  $k$  pentru cele patru cazuri ale valorilor derivatei sunt:

1.  $0 < \varphi'(x) < 1$ . Din (2.28) rezultă  $k \in (1, \infty)$ , adică o valoare a lui  $k$  mai mare ca 1 mărește viteza de convergență a metodei prin mărirea pasului;
2.  $-1 < \varphi'(x) < 0$ . Rezultă  $(1/2) < k < 1$ , adică în acest caz pasul trebuie micșorat pentru a obține o mărire a vitezei de convergență a metodei;
3.  $\varphi'(x) > 1$ . Rezultă  $k \in (-\infty, 0)$ , deși metoda în acest caz este divergentă pentru un  $k < 0$  se poate obține o îmbunătățire a metodei;
4.  $\varphi'(x) < 1$ . Rezultă  $k \in (0, 1/2)$ , adică o micșorare a pasului dar fără a ne pronunța asupra convergenței șirului în acest caz.

## 2.2.4. METODA NEWTON-RAPHSON

Aplicând metoda aproximațiilor succesive cu viteză de convergență mărită pentru care considerăm  $\xi = x_n$  rezultă din (2.31):

$$x_{n+1} = x_n + \frac{1}{1 - \varphi'(x_n)} (\varphi(x_n) - x_n) \quad (2.32)$$

Această expresie reprezintă o formulă de iterație pentru determinarea soluției ecuației  $x = \varphi(x)$  și se numește formula Newton - Raphson .

Formula (2.32) mai poate fi exprimată astfel :

$$x_{n+1} = \frac{(\varphi(x_n) - x_n \varphi'(x_n))}{1 - \varphi'(x_n)} \quad (2.33)$$

Formula de iterație (2.33) poate fi reprezentată sub forma :

$$x_{n+1} = g(x_n) \quad \text{unde} \quad g(x) = \frac{\varphi(x) - x \varphi'(x)}{1 - \varphi'(x)} \quad (2.34)$$

Condiția de convergență a șirului, spre soluția ecuației, este dată de relația  $|g'(x)| < 1$ .

Calculăm derivata funcției  $g(x)$

$$g'(x) = \frac{\varphi''(x)[\varphi(x) - x]}{[1 - \varphi'(x)]^2} \quad (2.35)$$

Pentru convergența metodei Newton - Raphson este necesar ca  $|g'(x)| < 1$ , condiție satisfăcută dacă:

1. Soluția de start  $x_0$  este aleasă cât mai aproape de soluția ecuației, pentru ca  $\varphi(x) - x$  să fie cât mai mic ;

2.  $\varphi''(x)$  este cât mai mic ;
3.  $\varphi'(x)$  nu este aproape de 1.

Formula Newton - Raphson (2.32) poate fi scrisă și pentru ecuația implicită  $f(x) = 0$  astfel :

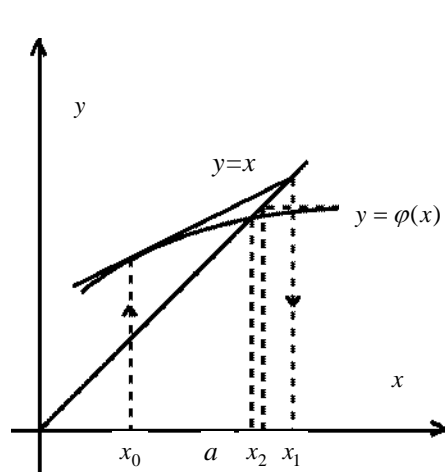


Fig.2.7. Determinarea grafică a soluției prin metoda Newton-Raphson

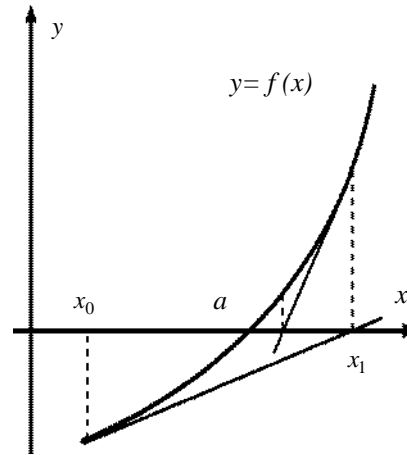


Fig.2.8. Determinarea grafică a soluției prin metoda tangentei.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad (2.36)$$

unde  $f(x) = \varphi(x) - x = 0$ . În acest caz formula (2.35) se scrie sub forma :

$$g'(x) = \frac{f''(x)f(x)}{[f'(x)]^2} \quad (2.37)$$

iar condițiile de convergență se transcriu:

1. Soluția de start să fie cât mai aproape de soluția ecuației,  $f(x)$  să fie cât mai mic;
2.  $f''(x)$  să fie cât mai mic;
3.  $f'(x)$  să fie cât mai depărtat de zero.

Formula (2.36) mai poartă numele și de metoda tangentei. Interpretările geometrice ale metodei Newton-Raphson date prin formulele (2.32) și (2.36) sunt prezentate în figurile 2.6, respectiv 2.7.

Șirul soluțiilor ecuației începe cu soluția de start  $x_0$ . În punctul de abscisă  $x_0$  al curbei  $y = \varphi(x)$  se trasează tangenta la curbă și se determină punctul de intersecție al tangentei cu prima bisectoare. Abscisa  $x_1$  a punctului de intersecție este următoarea valoare a soluției ecuației la primul pas de iterație. Procedul de determinare a soluției ecuației la pasul de iterație curent se face prin determinarea abscisei punctului de intersecție al tangentei la curbă în punctul de abscisă determinat anterior și prima

bisectoare. Șirul de abscise determinat, dacă îndeplinește condiția de convergență, tinde la soluția ecuației  $x = \varphi(x)$ . Pentru metoda tangentei se trasează tangenta la curba  $y = f(x)$  în punctul de abscisă calculat anterior care se intersectează cu axa  $Ox$ . Abscisa punctului de intersecție reprezintă soluția ecuației la etapa curentă de iterație.

#### 2.2.4.1. Algoritmul 2.5. Newton-Raphson pentru ecuații transcendente

```
{Variabile
  (F ecuația, funcție reală de variabilă reală)
  x0: valoarea punctului de start, real;
  nmax: numărul maxim de iterații, întreg;
  i: contor, întreg;
  xn: valoarea curentă a rădăcinii, real;
  xn_1: valoarea precedentă a rădăcinii, real;
  h, er: pasul de derivare, eroarea, reale;
  der: valoarea derivatei în xn_1, real;
  rad: rădăcina ecuației, real;
{
  i=0;
  xn=x0;
  h=0.0001;
  repetă
    xn_1=xn;
    der=(F(xn_1+h)-F(xn_1))/h;
    dacă der=0 atunci
      { scrie ecuația nu se poate rezolva;
        stop;
      }
    xn=xn_1-F(xn_1)/der;
  i=i+1;
  până când (abs(xn-xn_1)<er) sau (i>nmax);
  dacă i>nmax atunci
    {
      scrie soluția nu poate avea precizia dată;
      stop;
    }
  rad=xn;
  scrie soluția este rad;
}
```

#### 2.2.4.2. Implementarea algoritmului 2.5

```

/* Funcția întoarce:
    2 când derivata este nulă
    1 când nu pot afla rădăcina cu precizia dorită
    0 în caz de succes
*/
int NewtonRaphsonF( double (*f)(double),
                    double x0,
                    int niter,
                    double err,
                    double *soluție)
{
    double xn,xn_1,aux;
    int cont=1;
    xn=x0;
    do
    {
        xn_1=xn;
        if( (aux=Derf(f,xn_1))==0) return 2;
        xn=xn_1-f(xn_1)/aux;
        cont++;
    }
    while( ( fabs(xn-xn_1)>err) && (cont<=niter));
    if (cont>=niter) return 1;
    *soluție=xn;
    return 0;
}

```

## 2.2.5. METODA NEWTON-RAPHSON PENTRU POLINOAME

Această metodă mai poartă numele și de metoda Birge - Vieta.

Fie ecuația polinomială  $P(x) = 0$

$$P(x) = a_m x^m + a_{m-1} x^{m-1} + \dots + a_1 x + a_0$$

Aplicăm formula de calcul (2.36)

$$x_{n+1} = x_n - \frac{P(x_n)}{P'(x_n)} \quad (2.38)$$

Pentru calculul valorii unui polinom într-un punct dat  $x_0$ , aplicăm următorul procedeu:

$$P(x) = (x - x_0)(b_m x^{m-1} + b_{m-1} x^{m-2} + \dots + b_2 x + b_1) + b_0, \text{ iar } P(x_0) = b_0 \quad (2.39)$$

Pentru determinarea lui  $b_0$ , identificăm coeficienții polinoamelor din (2.39) și rezultă:

$$a_m = b_m$$

$$a_i = b_i - x_0 b_{i+1}, \quad i = m-1, m-2, \dots, 2, 1, 0$$

Pentru calculul coeficienților  $b_i$  utilizăm formulele :

$$\begin{aligned} b_m &= a_m \\ b_i &= a_i + x_0 b_{i+1} \quad i = m-1, m-2, \dots, 2, 1, 0 \end{aligned}$$

valoarea lui  $b_0$  este  $P(x_0)$ .

Pentru calculul  $P'(x_0)$  utilizăm formula (2.39) :

$$P(x_0) = (x - x_0)Q(x) + b_0 \text{ unde } Q(x) = b_m x^{m-1} + b_{m-1} x^{m-2} + \dots + b_2 x + b_1$$

Prin derivarea expresiei obținute rezultă :

$P'(x) = (x - x_0)Q'(x) + Q(x)$ , iar valoarea derivatei polinomului în  $x_0$  se calculează cu formula:  $P'(x_0) = Q(x_0)$  ceea ce duce la determinarea valorii lui  $Q(x_0)$ , polinom ai cărui coeficienți au fost obținuți anterior.

$$Q(x) = (x - x_0)(c_m x^{m-2} + c_{m-1} x^{m-3} + \dots + c_3 x + c_2) + c_1$$

Aplicând același procedeu ca anterior rezultă:

$$\begin{aligned} c_m &= b_m \\ c_j &= b_j - x_0 c_{j+1} \quad j = m-1, m-2, \dots, 2, 1 \end{aligned}$$

iar valoarea lui  $c_1$  reprezintă valoarea derivatei polinomului în punctul  $x_0$ .

Formula de iterație (2.38) pentru calculul soluției polinomului devine:

$$x_{n+1} = x_n - b_0/c_1 \quad (2.40)$$

unde:

$$\begin{aligned} b_m &= a_m \\ b_i &= a_i + x_0 b_{i+1} \quad i = m-1, \dots, 2, 1, 0 \\ c_m &= b_m \\ c_i &= b_i + x_0 c_{i+1} \quad i = m-1, \dots, 2, 1 \end{aligned} \quad (2.41)$$

### 2.2.5.1. Algoritmul 2.6. Newton-Raphson pentru polinoame

//Se utilizează funcția Valpol si Derpol din ANEXA 1//

{Variabile

$n$ : gradul polinomului, întreg;

$A$ : coeficienții polinomului, vector;

$x0$ : valoarea de start, real;

$er$ : eroarea de calcul, real;

$xn$ : valoarea curentă a rădăcinii, real;

$xn\_1$ : valoarea precedentă a rădăcinii, real;

$nmax$ : numărul maxim de iterații, întreg;

$i$ : contor, întreg;

$rad$ : rădăcina ecuației, real;

{

$i := 0$ ;

$xn := x0$ ;

```

    repetă
    xn_1=xn;
    dacă Derpol(n,A,xn_1)=0 atunci
    {
        ecuația nu se poate rezolva;
        stop;
    }

    xn=xn_1-Valpol(n,A,xn_1)/Derpol(n,A,xn_1);
    i=i+1;
    până când (abs(xn-xn_1)<er) sau (i>nmax);
    dacă i>nmax atunci
    {
        Nu avem precizia cerută;
        stop;
    }
    rad=xn;
    scrie soluția este rad;

}

```

### 2.2.5.2. Implementarea algoritmului 2.6

```

/* Funcția întoarce:
    2 când derivata este nulă
    1 când nu pot afla rădăcina cu precizia dorită
    0 în caz de succes
*/
int NewtonRaphsonP( int grad,
                    double coef[],
                    double x0,
                    int niter,
                    double err,
                    double *soluție)
{
    double xn,xn_1,aux;
    int cont=1;
    xn=x0;
    do
    {
        xn_1=xn;
        if( (aux=ValDerPol(grad,coef,xn_1))==0) return 2;
        xn=xn_1-ValPol(grad,coef,xn_1)/aux;
        cont++;
    }
}

```



```

while( ( fabs(xn-xn_1)>err) && (cont<=niter));
if (cont>=niter) return 1;
*soluție=xn;
return 0;
}

```

### 2.2.6. METODA NEWTON-RAPHSON PENTRU RĂDĂCINI FOARTE APROPIATE

Dacă două rădăcini ale ecuației sunt foarte apropiate atunci este foarte greu să găsim intervalul în care ecuația are o singură rădăcină, sau rădăcinile apropiate. Cum funcția în cele două rădăcini apropiate ia aceeași valoare egală cu zero, conform teoremei lui Rolle există cel puțin un punct  $x \in (a_1, a_2)$  astfel ca derivata funcției să se anuleze (fig 2.8).

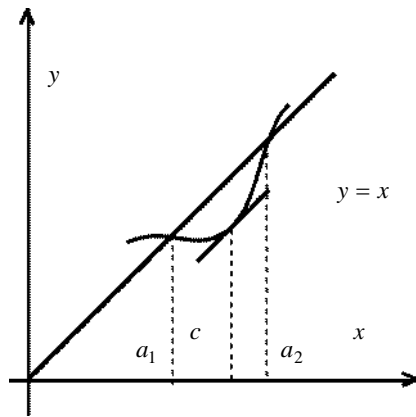


Fig.2.9. Graficul funcției cu două rădăcini foarte apropiate.

$$f(x) = \varphi(x) - x$$

$$f(a_1) = \varphi(a_1) - a_1 = 0$$

$$f(a_2) = \varphi(a_2) - a_2 = 0$$

Rezultă că există un  $x \in (a_1, a_2)$  pentru care  $f'(x) = 0$ . Rezultă că  $f'(x) = \varphi'(x) - 1$  adică  $\varphi'(x) = 1$ . Vom calcula soluția ecuației  $\varphi'(x) = 1$ , aplicând aceeași metodă Newton - Raphson, scriind ecuația derivatei egală cu

unitatea, sub forma:

$$x = x + \varphi'(x) - 1 \quad (2.42)$$

Considerăm rădăcina acestei ecuații  $x = c \in (a_1, a_2)$  pe care o putem lua cu aproximație la mijlocul distanței dintre rădăcinile  $a_1$  și  $a_2$ . Luând distanța dintre rădăcini egală cu  $2d$ , atunci  $c-d$  și  $c+d$  sunt două valori foarte apropiate de rădăcinile ecuației  $a_1$ , respectiv  $a_2$ , pe care le putem lua ca valori de start. Problema care se pune este determinarea valorii lui  $d$ , deoarece rădăcinile ecuației  $a_1, a_2$  nu sunt cunoscute. Pentru aceasta se dezvoltă în serie Taylor funcția  $\varphi(x)$  în jurul punctului  $c$ , soluție a ecuației (2.41)

$$\varphi(x) = \varphi(c) + \frac{(x-c)}{1!} \varphi'(c) + \frac{(x-c)^2}{2!} \varphi''(c) + \dots \quad (2.43)$$

Se iau în considerare primii trei termeni ai dezvoltării. Luând  $x = c + d$ , rezultă:

$$\varphi(c+d) = \varphi(c) + d\varphi'(c) + \frac{d^2}{(2!)}\varphi''(c),$$

sau 
$$c+d = \varphi(c) + d + \frac{(d^2)}{2}\varphi''(c),$$

expresie care se poate scrie sub forma:

$$d = \sqrt{\frac{2(c - \phi(c))}{\phi(c)}} \quad (2.44)$$

Cu ajutorul formulei obținute (2.44) se poate determina valoarea lui  $d$  dacă se cunoaște soluția ecuației (2.42) și se pot calcula valorile de start  $c - d$  și  $c + d$  pentru cele două soluții  $a_1$  respectiv  $a_2$ .

### 2.2.7. METODA LUI BAIRSTOW

Această metodă determină toate rădăcinile atât reale cât și complexe ale unei ecuații polinomiale cu coeficienți reali. Fie ecuația:

$$P_n(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0 = 0 \quad (2.45)$$

care admite soluțiile  $x_1, x_2, x_3, \dots, x_n$  (2.46), care pot fi reale și complexe. Rădăcinile complexe sunt conjugate două câte două. Principiul metodei constă în scrierea polinomului sub forma unui produs de două polinoame unul de gradul doi și celălalt de gradul  $n-2$ . Deci va rezulta o ecuație de gradul doi pe care o vom rezolva ușor aplicând formulele de calcul ale soluțiilor și o ecuație de gradul  $n-2$  pentru care vom proceda la fel pentru descompunere. În acest mod, din aproape în aproape, vom obține toate soluțiile ecuației polinomiale inițiale. Problema principală a metodei este determinarea coeficienților polinomului de gradul doi și a celui de gradul  $n-2$ . Ecuația (2.45) poate fi scrisă astfel:

$$P_n(x) = (x^2 + sx + p)(x^{n-2} + b_{n-1}x^{n-3} + b_{n-2}x^{n-4} + \dots + b_3x + b_2) + Rx + S \quad (2.47)$$

Prin identificarea coeficienților de același grad a ecuațiilor (2.45) și (2.47) se obțin expresiile recursive dintre coeficienții celor două polinoame pentru determinarea coeficienților  $b_i, i = n-1, n-2, \dots, 2$ ,  $R$  și  $S$  funcție de  $s$  și  $p$ . Determinarea valorilor lui  $s$  și  $p$  se face astfel ca  $R$  și  $S$  să se anuleze. În acest caz soluțiile polinomului de gradul doi sunt soluții și pentru ecuația dată.

$$\begin{aligned} a_{n-1} &= b_{n-1} + s \\ a_{n-2} &= b_{n-2} + sb_{n-1} + p \\ a_{n-3} &= b_{n-3} + sb_{n-2} + pb_{n-1} \\ &\text{-----} \\ a_2 &= b_2 + sb_3 + pb_4 \\ a_1 &= R + sb_2 + pb_3 \\ a_0 &= S + pb_2 \end{aligned} \quad (2.48)$$

Din relațiile (2.48) se pot deduce relațiile recursive pentru calculul coeficienților celor două polinoame în care s-a descompus polinomul inițial:

$$\begin{aligned}
b_{n-1} &= a_{n-1} - s \\
b_{n-2} &= a_{n-2} - sb_{n-1} - p \\
b_{n-3} &= a_{n-3} - sb_{n-2} - pb_{n-1} \\
&\text{-----} \\
b_2 &= a_2 - sb_3 - pb_4 \\
R &= a_1 - sb_2 - pb_3 \\
S &= a_0 - pb_2
\end{aligned} \tag{2.49}$$

Pentru uniformizarea formulelor de calcul (2.49) se face substituția formală :

$$\begin{aligned}
R &= b_1 = a_1 - sb_2 - pb_3 \\
S &= b_0 + sb_1
\end{aligned} \tag{2.50}$$

rezultând sistemul neliniar:

$$\begin{aligned}
b_{n-1} &= a_{n-1} - s \\
b_{n-2} &= a_{n-2} - sb_{n-1} - p \\
b_k &= a_k - sb_{k+1} - pb_{k+2} \\
k &= n-3, n-4, \dots, 2, 1, 0
\end{aligned} \tag{2.51}$$

unde  $R(s, p)$  și  $S(s, p)$  au expresiile (2.50).

Determinarea coeficienților  $s$  și  $p$  se face astfel ca restul  $Rx + S$  să fie zero, obținând în acest mod un produs de două polinoame, unul de gradul doi și celălalt de gradul  $n-2$  egal cu zero. Pentru că  $Rx + S = 0$  pentru orice  $x$  rezultă:

$$\begin{aligned}
R(s, p) &= 0 \\
S(s, p) &= 0
\end{aligned} \tag{2.52}$$

Sistemul neliniar (2.52) se rezolvă cu metoda lui Newton sau a matricei funcționale prezentată în capitolul 3. Din sistemul neliniar (2.52) se obține următorul sistem liniar în  $\Delta s_k$  și  $\Delta p_k$  :

$$\begin{aligned}
\frac{\partial R(s_k, p_k)}{\partial s} \Delta s_k + \frac{\partial R(s_k, p_k)}{\partial p} \Delta p_k &= -R(s_k, p_k) \\
\frac{\partial S(s_k, p_k)}{\partial s} \Delta s_k + \frac{\partial S(s_k, p_k)}{\partial p} \Delta p_k &= -S(s_k, p_k)
\end{aligned} \tag{2.53}$$

unde  $\Delta s_k = s_{k+1} - s_k$  iar  $\Delta p_k = p_{k+1} - p_k$ ,  $k$  reprezintă gradul iterației. (2.54)

Se pornește cu o soluție de start  $(s_0, p_0)$  care se înlocuiește în sistemul (2.53) și se obțin iterativ cu ajutorul formulelor (2.54), soluțiile sistemului.

Din expresia lui  $R(s, p)$  și a lui  $S(s, p)$  se deduc derivatele parțiale în raport cu  $s$  și  $p$ .

$$\begin{aligned}
\frac{\partial R(s, p)}{\partial s} &= \frac{\partial b_1}{\partial s} \\
\frac{\partial R(s, p)}{\partial p} &= \frac{\partial b_1}{\partial p} \\
\frac{\partial S(s, p)}{\partial s} &= \frac{\partial b_0}{\partial s} + \frac{\partial b_1}{\partial s} + b_1 \\
\frac{\partial S(s, p)}{\partial p} &= \frac{\partial b_0}{\partial p} + \frac{\partial b_1}{\partial p} \cdot s
\end{aligned} \tag{2.55}$$

Se observă că în ambele derivate parțiale a lui  $R(s, p)$  și  $S(s, p)$  în raport cu  $s$  și  $p$  avem derivata parțială a lui  $b_k$ ,  $k=1, 2, 3, \dots, n-1$  în raport cu  $s$  și  $p$ .

Dacă notăm  $t_k = \frac{\partial b_k}{\partial s}$  și  $q_k = \frac{\partial b_k}{\partial p}$   $k=n-1, n-2, \dots, 2, 1, 0$  se pot calcula valorile expresiilor (2.55) prin derivarea expresiilor (2.51).

$$\begin{aligned}
t_{n-1} &= -1 \\
t_{n-2} &= -s \cdot t_{n-1} - b_{n-1} \\
t_k &= -s \cdot t_{k+1} - p \cdot t_{k+2} - b_{k+1} \\
k &= n-3, n-4, \dots, 1, 0
\end{aligned} \tag{2.56}$$

$$\begin{aligned}
q_{n-1} &= 0 \\
q_{n-2} &= -1 \\
q_k &= -s \cdot q_{k+1} - p \cdot q_{k+2} - b_{k+2} \\
k &= n-3, n-4, \dots, 1, 0.
\end{aligned} \tag{2.57}$$

Coeficienții sistemului (2.53) se calculează mai întâi în punctul de start astfel:

$$\begin{aligned}
R(s_0, p_0) &= b_1(s_0, p_0) \\
S(s_0, p_0) &= b_0(s_0, p_0) + b_1(s_0, p_0)s_0 \\
\frac{\partial S(s_0, p_0)}{\partial s} &= t_1(s_0, p_0) \\
\frac{\partial R(s_0, p_0)}{\partial p} &= q_1(s_0, p_0) \\
\frac{\partial S(s_0, p_0)}{\partial s} &= t_0(s_0, p_0) + t_1(s_0, p_0)s_0 + b_1(s_0, p_0) \\
\frac{\partial S(s_0, p_0)}{\partial p} &= q_0(s_0, p_0) + s_0 q_1(s_0, p_0)
\end{aligned} \tag{2.58}$$

Se rezolvă sistemul pentru soluția de start, iar din expresiile (2.54) se determină soluțiile sistemului pentru iterația întâia  $s_1$  și  $p_1$ . Se continuă procesul de iterație până când  $|s_{k+1} - s_k| < \varepsilon$  și  $|p_{k+1} - p_k| < \varepsilon$  unde  $\varepsilon$  este eroarea impusă. Pentru aceste valori ale soluțiilor sistemului se consideră  $S(s_{k+1}, p_{k+1}) = 0$  și  $R(s_{k+1}, p_{k+1}) = 0$ . În aceste

condiții, prin rezolvarea ecuației de gradul doi, se obțin două soluții reale sau complexe ale ecuației inițiale (2.45). Continuând la fel cu ecuația de gradul  $n-2$  se obține în final un polinom de gradul doi dacă  $n$  este par și un polinom de gradul întâi dacă  $n$  este impar.

### 2.2.7.1. Algoritmul 2.7. Metoda lui Bairstow

*{Variabile*  
*a: vectorul coeficienților ecuației de rezolvat;*  
*b: vectorul coeficienților polinomului obținut prin descompunere;*  
 *$\Delta s, \Delta p$  : variațiile lui  $s$  și  $p$ ;*  
 *$s, p$ : coeficienții polinomului de gradul doi, reali;*  
 *$R, S$ : coeficienții restului, reali;*  
 $\frac{\partial R}{\partial s}, \frac{\partial R}{\partial p}$  : derivatele parțiale ale lui  $R$  în raport cu  $s$  și  $p$ , reale;  
 $\frac{\partial S}{\partial s}, \frac{\partial S}{\partial p}$  : derivatele parțiale ale lui  $S$  în raport cu  $s$  și  $p$ , reale;  
 $s_0, p_0$  : soluțiile de start pentru calculul lui  $s$  și  $p$ , reale;  
 $\varepsilon$  : eroarea de calcul, reală;  
 $n, i, k$  : întregi;  
 {  
   dacă  $a_n \neq 1$  atunci pentru  $k=n-1$  până la 0  $a_k := \frac{a_k}{a_n}$  ;  
    $i := n$ ;  
    $j := -1$ ;  
   repetă  
      $i := i - 2$ ;  
     repetă  
        $j := j + 1$ ;  
        $b_{i+1}(s_j, p_j) = a_{i+1} - s_j$   
        $b_i(s_j, p_j) = a_i - s_j \cdot b_{i+1} - p_j$   
       pentru  $k := i - 1$  până la 0  $b_k(s_j, p_j) = a_k - s_j \cdot b_{k+1} - p_j \cdot b_{k+2}$   
        $t_{i+1} = -1$   
        $t_i = -s_j \cdot t_{i+1} - b_{i+1}$   
       pentru  $k := i - 1$  până la 0  $t_k = -s_j \cdot t_{k+1} - p_j \cdot t_{k+2} - b_{k+1}$

$$q_{i+1} = 0$$

$$q_i = -1$$

$$\text{pentru } k := i - 1 \text{ până la } 0 \quad q_k = -s_j \cdot q_{k+1} - p_j \cdot q_{k+2} - b_{k+2}$$

$$R(s_j, p_j) = b_1(s_j, p_j)$$

$$S(s_j, p_j) = b_0(s_j, p_j) + b_1(s_j, p_j)s_j$$

$$\frac{\partial \mathcal{S}(s_j, p_j)}{\partial s} = t_1(s_j, p_j)$$

$$\frac{\partial \mathcal{R}(s_j, p_j)}{\partial p} = q_1(s_j, p_j)$$

$$\frac{\partial \mathcal{S}(s_j, p_j)}{\partial s} = t_0(s_j, p_j) + t_1(s_j, p_j)s_j + b_1(s_j, p_j)$$

$$\frac{\partial \mathcal{S}(s_j, p_j)}{\partial p} = q_0(s_j, p_j) + s_j q_1(s_j, p_j)$$

rezolvă sistemul

$$\frac{\partial \mathcal{R}(s_j, p_j)}{\partial s} \Delta s_j + \frac{\partial \mathcal{R}(s_j, p_j)}{\partial p} \Delta p_j = -R(s_j, p_j)$$

$$\frac{\partial \mathcal{S}(s_j, p_j)}{\partial s} \Delta s_j + \frac{\partial \mathcal{S}(s_j, p_j)}{\partial p} \Delta p_j = -S(s_j, p_j)$$

$$s_{j+1} = \Delta s_j + s_j;$$

$$p_{j+1} = \Delta p_j + p_j$$

$$\text{până când } (|s_{j+1} - s_j| < \varepsilon) \cap (|p_{j+1} - p_j|) < \varepsilon$$

$$\text{rezolvă ecuația } x^2 + s_{j+1}x + p_{j+1} = 0$$

$$\text{până când } (i-2 < 2)$$

Rezolvă ecuația de gradul 1 sau 2;

Tipărește soluțiile;

}

}

### 2.2.7.2. Implementarea algoritmului 2.7

/\* Funcția care întoarce semnul unui număr real \*/  
int Sign( double x)

```

{
    if (x>0) return 1;
    if (x<0) return -1;
    return 0;
}
/* Funcția Sign */
/* Funcția pentru rezolvarea ecuației de gradul doi */
void ECGR2( double a,
            double b,
            double c, /* coeficienții ecuației */
            double *xr1,
            double *xi1,
            double *xr2,
            double *xi2 /* partea reală și imag. a soluțiilor */
            )
{
    double d; /* determinantul ecuației */
    d=b*b-4*a*c;
    switch ( Sign(d) )
    {
    case 1: {
        *xr1=(-b+sqrt(d))/(2*a);
        *xi1=0;
        *xr2=(-b-sqrt(d))/(2*a);
        *xi2=0;
        break;
    }
    case 0: {
        *xr1=-b/(2*a);
        *xi1=0;
        *xr2=*xr1;
        *xi2=0;
        break;
    }
    case -1: {
        *xr1=-b/(2*a);
        *xi1=sqrt(-d)/(2*a);
        *xr2=*xr1;
        *xi2=-( *xi1);
        break;
    }
    } /* switch */
}; /* ECGR2 */
/* Funcție care implementează efectiv metoda Bairstow pentru
rezolvarea ecuațiilor polinomiale furnizând atât soluțiile reale, cât și
complexe, simple sau multiple. Ea întoarce următoarele coduri:

```

```

1 dacă se obțin soluțiile aproximative ale ecuației
2 dacă nu exista soluție unică (la determinant vezi alg)
3 dacă nu este atinsă precizia dorită
*/
int Bairstow( int grad, /* grad polinom */
              double coef[], /* coeficienții polinomului */
              double szero, /* soluția de start pt s -vezi alg */
              double pzero, /* soluția de start pentru p -vezi alg */
              double eps, /* precizia dorită a soluțiilor */
              int nmax, /* numărul maxim de iterații */
              double alfar[], /* partea reală a vect. soluție */
              double alfai[] /* partea imaginara a vect soluție */
            )
{
  /* Se recomandă a se vedea algoritmul */
  double R; /* coeficientul lui x al restului funcție de r si s */
double S; /* coeficientul liber al restului funcție de r si s */
double RS; /* derivata în raport cu s a lui R */
  double RP; /* derivata în raport cu p a lui R */
  double SS; /* derivata în raport cu s a lui S */
  double SP; /* derivata în raport cu p a lui S */
  double D; /* determinantul de la rez. sist.*/
  double DS; /* delta_s necunoscuta 1 din sistem */
  double DP; /* delta_p necunoscuta 2 din sistem */
  double s0; /* soluția s la pasul anterior */
  double p0; /* soluția p la pasul anterior */
  double s1; /* soluția s la pasul actual */
  double p1; /* soluția p la pasul actual */
  int sem1; /* Semafor de ieșire din bucla mare vezi algoritm */
  int sem2; /* Semafor de ieșire din bucla mică vezi algoritm */
  int k; /* Un contor */
  int j; /* ordinul rădăcinii curente; la un pas aflu rad.j */
          /* si rad. j+1 */
  int m; /* ordinul polinomului la un moment dat */
  int i; /* numărul de iterații curent */
  double b[NrMax]; /* coeficienții polinomului de ordin m-2 */
  double t[NrMax]; /* vectorul derivatelor coefic. b în raport cu s */
  double q[NrMax]; /* vectorul derivatelor coefic. B în raport cu p */
  /* realizăm normarea polinomului */
  for(k=grad-1;k>=0;k--)coef[k]/=coef[grad];
  coef[grad]=1;
  /* end normare */
  j=1;
  sem1=0;
  do
  {

```



```

m=grad
switch (Sign(m-2))
{
  case -1: { /* a rămas ec. de gr 1 */
    alfar[j]=-coef[m-1];
    alfai[j]=0;
    sem1=1;
    break;
  };
  case 0: { /* a rămas ecuație de gr 2 */
    ECGR2(1.0,coef[m-1],coef[m-2],
          &alfar[j],&alfai[j],
          &alfar[j+1],&alfai[j+1]);
    sem1=1;
    break;
  };
  case 1: { /* ecuație de grad mai mare ca doi */
    s0=rzero;
    p0=szero;
    i=1;
    sem2=0;
    do
    {
      b[m-2]=1;
      b[m-3]=coef[m-1]-r0;
      t[m-2]=0;
      t[m-3]=-1;
      q[m-2]=q[m-3]=0;
      if(m>3)for(k=m-4;k>=0;k--)
      {
        b[k]=coef[k+2]-b[k+1]*s0-b[k+2]*p0;
        t[k]=-b[k+1]-t[k+1]*s0-t[k+2]*p0;
        q[k]=-q[k+1]*s0-q[k+2]*p0-b[k+2];
      }
      R=coef[1]-s0*b[0]-p0*b[1];
      S=coef[0]-p0*b[0];
      RS=-b[0]-s0*t[0]-p0*t[1];
      RP=-q[0]*s0-b[1]-p0*q[1];
      SS=-p0*t[0];
      SP=-b[0]-p0*q[0];
      D=RR*SS-SR*RS;
      if(D!=0)
      {
        DS=(-R*SS+S*RS)/D;
        DP=(-S*RR+R*SR)/D;
        s1=s0+DS;

```

```

    p1=p0+DP;
    if((fabs(DS)<=eps)&&(fabs(DP)<=eps) )
    {
        ECGR2(1,r1,s1,&alfar[j],&alfai[j],
        &alfar[j+1],&alfai[j+1]);
        j+=2;
        sem2=1;
        for(k=0;k<=m-2;k++)coef[k]=b[k];
    }
    else
    {
        i++;
        if(i>nmax) return 3;
        else
        {
            s0=s1;
            p0=p1;
        }
    }
}
else return 2;
} while (sem2!=1);
break;
}
}; /* Case */
} while (sem1!=1);
return 1;
} /* end Bairstow */

```

## 2.3. APLICAȚII

1. Se consideră un amplificator integrat, având în buclă deschisă o amplificare de forma:

$$H(s) = \frac{338.6260295}{s^3 + 0.5284s^2 + 0.05120153s + 0.00096807437}$$

Să se determine polii funcției de amplificare.

Pentru rezolvare, mai întâi determinăm numărul de rădăcini reale ale polinomului de la numitorul fracției aplicând șirul lui Șturm. Cu ajutorul funcției ce implementează șirul lui Șturm se obține următorul șir:

$$f_1(s) = s^3 + 0.5284s^2 + 0.05120153s + 0.00096807437;$$

$$f_2(s) = 3s^2 + 1.0568s + 0.05120153$$

$$f_3(s) = 0.027912s + 0.002038$$

$$f_4(s) = 0.009969$$

Se determină numărul variațiilor de semn ale șirului lui Șturm la  $-\infty$  și  $\infty$ . Pentru aceasta se realizează tabelul 2.1

Tabelul 2.1

$s$	$f_1$	$f_2$	$f_3$	$f_4$	$n$
$-\infty$	-	+	-	+	3
$\infty$	+	+	+	+	0

Diferența dintre numărul de variații la  $-\infty$  și  $\infty$  dă informații asupra numărului de rădăcini reale ale ecuației date. Pentru determinarea intervalelor în care se găsesc soluțiile ecuației date, facem șirul derivatelor.

$$f(s) = s^3 + 0.5284s^2 + 0.05120153s + 0.00096807437;$$

$$f'(s) = 3s^2 + 1.0568s + 0.05120153$$

$$f''(s) = 6s + 1.0568$$

$$f'''(s) = 6$$

Se determină soluțiile ecuațiilor din șirul derivatelor și se ține seama că între două rădăcini consecutive ale derivatei unei funcții poate exista cel mult o rădăcină a funcției. Soluțiile se determină cu eroarea  $\varepsilon = 10^{-5}$  suficientă pentru determinarea intervalelor în care se găsesc soluțiile. Pentru derivata de ordinul doi se consideră intervalul  $(-5, +5)$  și se determină soluția  $s_1'' = 0.17613$  cu ajutorul metodei biseecției. Pentru derivata întâi se determină o soluție în intervalul  $(-5, 0.17613)$  și cealaltă în intervalul  $(0.17613, 5)$ . Se obțin soluțiile:

$$s_1' = -0.29426; \quad s_2' = -0.05800$$

Acum se pot determina intervalele în care se găsesc polii funcției de amplificarea:

$$(-5, 0.29426); (-0.29426, -0.05800) \text{ și } (-0.05800, 5)$$

Polii sunt determinați cu metodele:

$$\text{Biseecție:} \quad s_1 = -0.409; \quad s_2 = -0.0943; \quad s_3 = -0.0251$$

Aproximații succesive:  $s_1 = -0.409$ ;  $s_2 = -0.0943$ ;  $s_3 = -0.0251$

Newton-Raphson:  $s_1 = -0.409$ ;  $s_2 = -0.0943$ ;  $s_3 = -0.0251$

2. Se consideră funcția de transfer a unui filtru analogic de tip Cebîșev dată sub forma:

$$H(s) = \frac{0.04381}{s^4 + 0.6192s^3 + 0.61401692s^2 + 0.20379268s + 0.0491588}$$

Să se determine polii filtrului.

Filtrul admite poli complecși și pentru determinarea lor aplicăm metoda lui Bairstow. Se obțin următorii poli ca soluții ale polinomului de la numitorul funcției de transfer:

$$s_{12} = -0.090700 \pm 0.639041i \quad s_{34} = -0.218900 \pm 0.264732i$$

3. Se dă ecuația

$$e^x - 1/x = 0$$

care are o soluție în intervalul (0.1, 1). Să se calculeze soluția ecuației aplicând metoda biseției pentru ecuații transcendente, metoda Newton-Raphson și metoda aproximațiilor succesive. Eroarea de calcul se consideră  $\varepsilon = 0.0000000001$  pentru toate metodele. Să se compare rezultatele.

# 3

## REZOLVAREA NUMERICĂ A SISTEMELOR DE ECUAȚII\*

Fie funcția  $f: X \rightarrow Y$ ,  $X \subset \mathbf{R}^n$ ,  $Y \subset \mathbf{R}^n$  și  $f(x) = 0$  un sistem de ecuații. După gradul necunoscutelor care intră în aceste ecuații, sistemele sunt liniare, dacă termenii ecuațiilor sunt de gradul întâi și neliniare dacă există ecuații ce conțin termeni de grad mai mare ca unu.

### 3.1. REZOLVAREA NUMERICĂ A SISTEMELOR LINIARE

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

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

sau matriceal  
unde:

$$AX=B \quad (3.2)$$

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ - & - & - & - \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad (3.3)$$

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

Metodele de rezolvare a sistemelor liniare le putem împărți în două categorii: metode directe care presupun un număr finit de operații și metode indirecte care

---

\*) *Bibliografie* [3], [4], [5], [8], [9], [15], [17]





### 3.1.1.2.1. Algoritmul 3.2. Sistem superior triunghiular

```

{ Variabile
  A: matricea necunoscutelor sistemului;
  B: vectorul termenilor liberi;
  X: vectorul soluțiilor;
  n,i,j:întregi;
{
    calculează  $x_n = \frac{b_n}{a_{nn}}$  ;

    pentru i=n-1,..., 1 calculează  $x_i = \frac{b_i - \sum_{j=i+1}^n a_{ij} x_j}{a_{ii}}$  ;
    pentru i= 1 ,..., n tipărește  $x_i$ 
}

```

### 3.1.1.2.2. Implementarea algoritmului 3.2

```

/* Funcția întoarce determinantul matricei principale */

double TRIUNGHISUP (int or_mat,
                    double MAT[][NrMax],
                    double TL[],
                    double XX[])

{
    int i;          /* indice de linie */
    int j;          /* indice de coloană */
    double d=1;     /* determinant */

    for (i=1;i<=or_mat;i++) d*=MAT[i][i];
    if ( d==0 ) return 0;

    /*retrosubstituția */

    for (i=or_mat;i>=1;i--)
    {
        XX[i]=TL[i];
        for(j=or_mat;j>=i+1;j--) XX[i]=XX[i]- MAT[i][j]*XX[j];
        XX[i]=XX[i]/MAT[i][i];
    }

    return d;
}

```



}

### 3.1.1.3. Metoda lui Gauss de eliminare

Această metodă constă în realizarea unui sistem triunghiular prin eliminarea termenilor de sub diagonală principală, sistemul (3.1). Pentru aceasta se analizează toți termenii  $a_{k1}$ ,  $k=1, \dots, n$ , și ecuația care are valoarea maximă a acestui coeficient este adusă pe primul loc. Dacă toți  $a_{k1}=0$  pentru  $k=1, \dots, n$  sistemul este incompatibil. Prima ecuație a sistemului după reordonare se înmulțește pe rând cu factorul :

$$m_{k1} = \frac{a_{k1}}{a_{11}}, \quad k=2, \dots, n \quad (3.10)$$

și se scade din ecuația de pe poziția  $k$ , obținând pe coloana 1 toți termenii zero în afară de  $a_{11}$ . Sistemul devine :

$$\begin{aligned} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + a_{13}^{(1)}x_3 + \dots + a_{1n}^{(1)}x_n &= b_1^{(1)} \\ a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 + \dots + a_{2n}^{(2)}x_n &= b_2^{(2)} \\ a_{32}^{(2)}x_2 + a_{33}^{(2)}x_3 + \dots + a_{3n}^{(2)}x_n &= b_3^{(2)} \\ &\vdots \\ a_{n2}^{(2)}x_2 + a_{n3}^{(2)}x_3 + \dots + a_{nn}^{(2)}x_n &= b_n^{(2)} \end{aligned} \quad (3.11)$$

Procedeeul se continuă cu aducerea pe locul lui  $a_{22}$  a celui mai mare termen din coloana doi și se înmulțește linia a doua cu :

$$m_{k2} = \frac{a_{k2}}{a_{22}}, \quad k=3, 4, \dots, n \quad (3.12)$$

și se scade din ecuația de pe poziția  $k$ . Pentru coloana  $i$  după aducerea celui mai mare coeficient în locul lui  $a_{ii}$  se înmulțește această coloană cu :

$$m_{ki} = \frac{a_{ki}}{a_{ii}}, \quad k=i+1, \dots, n \quad (3.13)$$

și se scade din linia de pe poziția  $k$ .

Pentru  $i = n-1$  se obține sistemul :

$$\left\{ \begin{aligned} a_{11}^{(1)}x_1 + a_{12}^{(1)}x_2 + a_{13}^{(1)}x_3 + \dots + a_{1n}^{(1)}x_n &= b_1^{(1)} \\ a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 + \dots + a_{2n}^{(2)}x_n &= b_2^{(2)} \\ &\vdots \\ a_{nn}^{(n)}x_n &= b_n^{(n)} \end{aligned} \right. \quad (3.14)$$

Soluțiile se obțin printr-o retrosubstituție, pornind de la ultima ecuație

$$x_n = \frac{b_n^{(n)}}{a_{nn}^{(n)}} \quad x_i = \left( b_i^{(i)} - \sum_{j=i+1}^n a_{ij}^{(i)} x_j \right) \frac{1}{a_{ii}^{(i)}} \quad (3.15)$$

*Observație.* Metoda lui Gauss de eliminare se poate aplica și fără a aduce pe locul  $a_{ii}$ ,  $i=1,2,\dots,(n-1)$  celui mai mare termen, condiția este ca  $a_{ii} \neq 0$ , dar în acest caz precizia de calcul scade. Metoda este prezentată pentru cazul celei mai bune precizii de determinare a soluțiilor.

### 3.1.1.3.1. Algoritm 3.3. Metoda lui Gauss de eliminare

```
{Variabile
A:matricea sistemului;
B:vectorul termenilor liberi;
X:vectorul soluțiilor ;
n,i :ordinul sistemului, indicele liniei, întregi;
k : indice suplimentar al liniei, întreg;
j : indice coloană, întreg;
m: multiplicator, real;
d: determinantul sistemului, real;
max: elementul maxim de pe coloana, real;
lp: indicele liniei cu element maxim, întreg;
{
i=1; det=1;
repetă
    max= abs(A[i,i]);
    lp=1;
    pentru k=i+1 ... n execută
        dacă max<abs(A[k,i]) atunci
            { max<A[k,i];
              lp=k;}
    dacă max=0 atunci
        { GAUSS=0;
          exit; }
    dacă lp<>i atunci
        { pentru j=i,...,n execută
            sc=A[i,j];A[i,j]=A[lp,j];A[lp,j]=sc;
            sc=B[i];B[i]=B[lp];B[lp]=sc;
            d= -d;
        }
    k=i+1;
    repetă
        m=A[k,i]/A[i,i];
        pentru j =i,...,n execută A[k,j]=A[k,j]-m*A[i,j];
        B[k]=B[k]-m*B[i];
        k=k+1;
    până când k>n;
    i=i+1;
```

```

    până când  $i \geq n$ ;
    pentru  $i=1, \dots, n$  execută     $d := d * A[i, i]$ ;
    dacă  $d=0$  atunci
        {      GAUSS=0;
          exit;  }
     $X[n] = B[n] / A[n, n]$ ;
    pentru  $i=n-1, \dots, 1$  execută
    {  $X[i] = B[i]$ ;
      pentru  $j=n, \dots, i+1$  execută     $X[i] = X[i] - A[i, j] * X[j]$ ;
       $X[i] = X[i] / A[i, i]$ ; }
    GAUSS=d;
}

```

### 3.1.1.3.2. Implementarea algoritmului 3.3

```

#define SCHIMB(a,b,temp) (temp)=(a);(a)=(b);(b)=(temp);
/* Funcția implementează metoda eliminării a lui Gauss (pivotare
parțială). Funcția întoarce determinantul matricei principale */
double GAUSS(int or_mat,
             double MAT[][NrMax],
             double TL[],
             double XX[])
{
    int i;    /* indice de linie */
    int j;    /* indice de coloană */
    int k;    /* indice suplimentar de coloană */
    double m; /* multiplicator */
    double d; /* determinant */
    double sc;
    double max;
    int lp;   /* indicele liniei în care se găsește pivotul */
    i=1;
    d=1;

    do
    {
        max=fabs( MAT[i][i]);lp=i;
        for(k=i+1;k<=or_mat;k++)
            if( max< fabs(MAT[k][i]) )
            {
                max=MAT[k][i];
                lp=k;
            }
        if (max==0) return 0;
        if(lp!=i)

```

```

        {
            for(j=i;j<=or_mat;j++)
                SCHIMB(MAT[i][j],MAT[lp][j],sc);
            SCHIMB(TL[i],TL[lp],sc);
            d*=-1;
        }
        k=i+1;
    do
        {
            m=MAT[k][i]/MAT[i][i];
            for (j=i;j<=or_mat;j++)
                MAT[k][j]=MAT[k][j]-m*MAT[i][j];
            TL[k]=TL[k]-m*TL[i];
            k++;
        }

        while (k<=or_mat);
        i++;
    }while (i<or_mat);
    for (i=1;i<=or_mat;i++) d*=MAT[i][i];
    if ( d==0 ) return 0;

    /*retrosubstituția */

    for (i=or_mat;i>=1;i--)
    {
        XX[i]=TL[i];
        for(j=or_mat;j>=i+1;j--) XX[i]=XX[i]-MAT[i][j]*XX[j];
        XX[i]=XX[i]/MAT[i][i];
    }

    return d;
}

```

#### 3.1.1.4. Metoda lui Crout

Rezolvarea sistemului  $AX=B$  (3.16) constă în descompunerea matricelor  $A=LU$  și  $B=LD$  (3.17),  $L$  fiind o matrice inferior triunghiulară iar  $U$  o matrice superior triunghiulară cu elementele diagonalei principale egale cu unitatea. După înlocuirea relațiilor (3.17) în ecuația (3.16) prin simplificare se obține ecuația  $UX=D$  (3.18) care este ușor de rezolvat. Din sistemul matriceal format de ecuațiile (3.17) se calculează elementele matricelor  $U$  și  $L$ .

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & 0 & \dots & 0 \\ - & - & - & - & - & - \\ l_{n1} & l_{n2} & l_{n3} & l_{n4} & \dots & 1 \end{pmatrix} \begin{pmatrix} 1 & u_{12} & u_{13} & \dots & \dots & u_{1n} \\ 0 & 1 & u_{23} & \dots & \dots & u_{2n} \\ 0 & 0 & 1 & \dots & \dots & u_{3n} \\ - & - & - & - & - & - \\ 0 & 0 & 0 & \dots & \dots & 1 \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ - & - & - & - \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad (3.16)$$

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 & \dots & 0 \\ l_{21} & l_{22} & 0 & 0 & \dots & 0 \\ l_{31} & l_{32} & l_{33} & 0 & \dots & 0 \\ - & - & - & - & - & - \\ l_{n1} & l_{n2} & l_{n3} & l_{n4} & \dots & 1 \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ - \\ d_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ - \\ b_n \end{pmatrix} \quad (3.17)$$

Din ecuațiile (3.19) și (3.20) prin identificarea elementelor corespunzătoare rezultă următoarele formule de calcul ale matricelor  $U$  și  $L$ .

$$l_{i1} = a_{i1}; \quad i = 1, 2, \dots, n; \quad u_{1j} = \frac{a_{1j}}{a_{11}}; \quad j = 1, 2, \dots, n; \quad d_1 = \frac{b_1}{a_{11}}; \quad a_{11} \neq 0; \quad (3.18)$$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}; \quad i \geq j; \quad u_{ij} = \frac{1}{l_{ii}} \left( a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \right); \quad l_{ii} \neq 0; \quad i < j; \quad (3.19)$$

$$d_i = \frac{1}{l_{ii}} \left( b_i - \sum_{k=1}^{i-1} l_{ik} d_k \right) \quad (3.20)$$

Soluțiile sistemului se calculează printr-o retrosubstituție cu ajutorul următoarelor formule:

$$x_n = d_n; \quad x_i = \frac{d_i - \sum_{j=i+1}^n a_{ij} x_j}{u_{ii}}; \quad i = n-1, n-2, \dots, 2, 1; \quad (3.21)$$

#### 3.1.1.4.1. Algoritm 3.4. Metoda lui Crout

{Variabile

*MAT*:matricea coeficienților necunoscutelor sistemului;

*TL*:vectorul termenilor liberi;

*X*:vectorul soluțiilor sistemului;

*U*:matricea superior triunghiulară cu diagonala principală unitară;

*L*:matricea inferior triunghiulară ;

*n*:ordinul sistemului, întreg;

*i, j*:indice de linie respectiv de coloană, întreg;

*k*:indice suplimentar, întreg;

{

dacă  $a_{11} = 1$  atunci scrie -sistem nerezolvabil

altfel

{ pentru  $i=1, \dots, n$

```

    pentru j=1 ,..., n
    { calculează
       $l_{i1} = a_{i1}$  ;
       $u_{1j} = \frac{a_{1j}}{a_{11}}$  ;
       $d_1 = \frac{b_1}{a_{11}}$  ;  $a_{11} \neq 0$  ;
    }
    dacă  $i \geq j$  atunci
      { calculează
         $l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj}$  ;
         $u_{ij} = \frac{1}{l_{ii}} \left( a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \right)$  ;
         $d_i = \frac{1}{l_{ii}} \left( b_i - \sum_{k=1}^{i-1} l_{ik} d_k \right)$ 
      }
    dacă  $i < j$  atunci
      { calculează
         $u_{ij} = \frac{1}{l_{ii}} \left( a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} \right)$  ;
      }
    }
    calculează
    {  $x_n = d_n$  ;
      pentru  $i=n-1 ,..., 1$ 
        calculează  $x_i = \frac{d_i - \sum_{j=i+1}^n a_{ij} x_j}{u_{ii}}$  ;
      }
    Tipărește soluțiile
  }

```

#### 3.1.1.4.2. Implementarea algoritmului 3.4

```

/* Funcția întoarce
0 dacă nu se poate face descompunerea matricei
1 dacă descompunerea matricei are loc */
int Crout( int or_mat,
double MAT[][NrMax],
double X[],
double TL[],

```

```

double L[][NrMax],
double U[][NrMax])
{
    int i,j,k;
    double s;
    static double Y[NrMax];
    /* Pentru fiecare coloană - i - */
    for(i=1;i<=or_mat;i++)
    {
        /* Se calculează elementele matricei L */
        /* Pentru fiecare linie - j - */
        for(j=i;j<=or_mat;j++)
        {
            s=0;
            for(k=1;k<=i-1;k++)s+=L[j][k]*U[k][i];
            L[j][i]=MAT[j][i]-s;
        }
        for(k=1;k<=i-1;k++)L[k][i]=0;

        /* Se calculează elementele matricei R */
        /* Pentru fiecare coloană - j - */
        for(j=i+1;j<=or_mat;j++)
        {
            s=0;
            for(k=1;k<=i-1;k++)s+=L[i][k]*U[k][j];
            if ( L[i][i]==0 )return 0; else U[i][j]=(MAT[i][j]-s)/L[i][i];
        }

        for(j=1;j<=i-1;j++)U[i][j]=0;
        U[i][i]=1;
    }
    if( TRIUNGHIINF(or_mat,L,TL,Y)==0 )return 2;
    If( TRIUNGHIISUP(or_mat,U,Y,X)==0 )return 2;
    return 1;}

```

### 3.1.1.5. Metoda lui Cholesky

Metoda lui Cholesky se referă la matricele simetrice ce au proprietatea că  $A^T = A$  și în acest caz  $l_{ij} = u_{ji}l_{jj}$ ;  $i = 1, 2, \dots, n-1$ ;  $j = 1, 2, \dots, n-1$ ;  $i \neq j$ ;

#### 3.1.1.5.1. Algoritmul 3.5. Metoda lui Cholesky

Algoritmul pentru metoda lui Cholesky este identic cu algoritmul Crout, dar se ține cont că  $a_{ij} = a_{ji}$ ;  $i, j = 1, \dots, n$

### 3.1.1.5.2. Implementarea algoritmului 3.5.

```

/* Funcția întoarce
0 dacă nu se poate face descompunerea matricei
1 dacă descompunerea matricei are loc
2 dacă matricea nu e simetrică
3 dacă sistemul nu se poate rezolva
*/
int Choleski( int or_mat,
              double MAT[][NrMax],
              double L[][NrMax],
              double X[],
              double TL[])
{
    int i,j,k;
    double s;
    static double U[NrMax][NrMax], Y[NrMax];
    /* Verificarea simetriei */
    for(i=1; i<=or_mat; i++)
        for(j=i+1; j<=or_mat; j++)
            if( MAT[i][j]!=MAT[j][i] )return 2;
    /* Pentru fiecare linie */
    for(i=1; i<=or_mat; i++)
    {
        /* Calculăm elementele de pe linie cu j<i */
        for(j=1; j<=i-1; j++)
        {
            s=0;
            for(k=1; k<=j-1; k++)s+=L[i][k]*L[j][k];
            if( L[j][j]==0 ) return 0; else L[i][j]=(MAT[i][j]-s)/L[j][j];
        }
        /* Aflăm elementul diagonal */
        s=0;
        for(k=1; k<=i-1; k++)s+=L[i][k]*L[i][k];
        if( (MAT[i][i]-s)<0)return 0; else L[i][i]=sqrt(MAT[i][i]-s);

        /* Umplu cu zero spatiile din matricea L pentru j>i */
        for(k=i+1; k<=or_mat; k++)L[i][k]=0;
    }
    for(i=1; i<=or_mat; i++)
        for(j=1; j<=or_mat; j++)
            U[i][j]=L[j][i];

```



```

if( TRIUNGHINF(or_mat,L,TL,Y)==0 )return 3;
if( TRIUNGHISUP(or_mat,U,Y,X)==0 )return 3;
return 1;
}

```

### 3.1.1.6. Metoda Gauss-Jordan sau matriceală formală

Această metodă se deosebește de metoda eliminării a lui Gauss prin faptul că matricea sistemului este adusă la o matrice diagonală prin transformări elementare aplicate ecuației matriceale (3.2) .

$$E_n E_{n-1} E_{n-2} \dots E_2 E_1 A X = E_n E_{n-1} E_{n-2} \dots E_2 E_1 B \quad (3.22)$$

Transformările elementare se fac astfel ca

$$E_n E_{n-1} \dots E_2 E_1 A = E \quad (\text{matricea unitate}) \quad (3.23)$$

Ecuația (3.22) devine

$$X = C \quad (3.24)$$

unde

$$C = E_n E_{n-1} \dots E_1 B \quad (3.25)$$

Prin identificarea celor două matrice se obțin soluțiile sistemului (3.2) .

Pentru obținerea matricei  $E$  (3.23) se fac următoarele transformări asupra coeficienților matricei  $A$  și  $B$  :

$$a_{ij}^{(1)} = a_{uj}$$

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \frac{a_{i,k-1}^{(k-1)}}{a_{k-1,k-1}^{(k-1)}} a_{k-1,j}^{(k-1)} \quad \text{pentru } i \neq k-1, j \geq k-1$$

$$a_{k-1,j}^{(k)} = a_{k-1,j}^{(k-1)} \quad \text{pentru } j \geq k-1$$

$$a_{i,j}^{(k)} = a_{ij}^{(k-1)} \quad \text{pentru } j < k-1 \quad (3.26)$$

$$b_i^{(1)} = b_i$$

$$b_i^{(k)} = b_i^{(k-1)} - \frac{a_{i,k-1}^{(k-1)}}{a_{k-1,k-1}^{(k-1)}} b_{k-1}^{(k-1)} \quad \text{pentru } i \neq k-1$$

$$b_{k-1}^{(k)} = b_{k-1}^{(k-1)} \quad \text{pentru } k = 2, 3, \dots, n$$

**3.1.1.6.1. Algoritm 3.6. Metoda Gauss-Jordan***{Variabile**A : matricea sistemului;**B : vectorul termenilor liberi;**X: vectorul soluțiilor;**n : numărul necunoscutelor (ordinul matricei), întreg;**lin : indicele liniei, întreg;**k : indice suplimentar al liniei, întreg;**j : indice coloană, întreg;**er1 : indică dacă e false că algoritmul poate continua ( s-a găsit pivot );**ok : indică dacă e true succesul interschimbării liniilor, boolean;**det : variabila în care se calculează determinantul sistemului, real ;**{**lin = 1;**er1 = false;**det = 1;**repetă**dacă A[lin,lin] = 0 atunci**{ ok = false;**k = lin+1;**repetă**dacă A[k,lin] ≠ 0 atunci {**INTERSCHIMBA LINIA lin cu LINIA k ;**ok = true;**}**altfel k = k+1;**dacă k = n+1 atunci er1 = true ;**până când (er1 = true) sau (ok = true)**}**dacă er1 = false atunci {**k = lin+1;**repetă**calculează  $m = \frac{A[k,lin]}{A[lin,lin]}$  ;**pentru j = lin ... n calculează A[k,j] = A[k,j] - m • A[lin,j]**B[k] = B[k] - m • B[lin];**k = k+1;**până când k = n+1;**}**lin = lin+1;**până când (lin = n+1) sau (er1 = true );*

```

    pentru lin = 1...n calculează  $det = det \cdot A[lin.lin]$ ;
    dacă  $det \neq 0$  atunci
        {  $B[n] = \frac{B[n]}{A[n,n]}$  ;
          pentru lin = n-1
            {
              pentru j = n... lin+1 calculează  $B[lin] = B[lin] -$ 
                 $A[lin,j]B[j]$ 
                 $B[lin] = \frac{B[lin]}{A[lin,lin]}$  }
              pentru lin = 1...n {
                identifică  $X[lin] = B[lin]$ ;
                tipărește  $X[lin]$ ;
              }
            }
        altfel tipărește "Sistemul este incompatibil sau compatibil nedeterminat "
    }

```

### 3.1.1.6.2. Implementarea algoritmului 3.6

```

#define SCHIMB(a,b,temp) (temp)=(a);(a)=(b);(b)=(temp);
/* Funcția implementează metoda Gauss-Jordan pentru
   rezolvarea sistemelor liniare.
   Funcția întoarce determinantul matricei principale */

double GAUSS_JORDAN(int or_mat,
                    double MAT[][NrMax],
                    double TL[],
                    double XX[])
{
    int i;    /* indice de linie */
    int j;    /* indice de coloană */
    int k;    /* indice suplimentar de coloană */
    double m; /* multiplicator */
    double d; /* determinant */
    double sc;
    double max;
    int lp;   /* indicele liniei în care se găsește pivotul */
    i=1;
    d=1;
    do
    {
        max=fabs( MAT[i][i]);lp=i;
        for(k=i+1;k<=or_mat;k++)

```

```

        if( max< fabs(MAT[k][i]) )
        {
            max=MAT[k][i];
            lp=k;
        }
    if (max==0) return 0;
    if(lp!=i)
    {
        for(j=i;j<=or_mat;j++){
            SCHIMB(MAT[i][j],MAT[lp][j],sc);}
            SCHIMB(TL[i],TL[lp],sc);
            d*=-1;
        }
    k=1;
    do
    {
        if(k!=i)
        {
            m=MAT[k][i]/MAT[i][i];
            for (j=i;j<=or_mat;j++)MAT[k][j]=MAT[k][j]-
m*MAT[i][j];
            TL[k]=TL[k]-m*TL[i];
        }
        k++;
    }
    while (k<=or_mat);
    i++;
}while (i<=or_mat) ;
for (i=1;i<=or_mat;i++) d*=MAT[i][i];
if ( d==0 )return 0;
for(i=1;i<=or_mat;i++)XX[i]=TL[i]/MAT[i][i];
return d;
}

```

### 3.1.1.7. Metoda factorizării QR

Această metodă face posibilă rezolvarea sistemelor  $AX=B$  pentru cazul când  $A \in \mathbf{R}^{m \times n}$  și  $B \in \mathbf{R}^m$ ,  $m > n$  caz în care nu au soluție în general. Soluționarea se face cu ajutorul metodei lui Householder ce are la bază următoarea teoremă:

**Teorema 3.1.** Pentru orice matrice  $A \in \mathbf{R}^{m \times n}$  există o matrice ortogonală  $H \in \mathbf{R}^{m \times n}$  astfel încât  $HA=R$  unde  $R \in \mathbf{R}^{m \times n}$  și este o matrice superior triunghiulară. Matricea  $H$  se construiește astfel:

$$H = \prod_{p=1}^n H_p \quad (3.27)$$

Matricea  $H_p$  poartă numele de reflector Householder și se determină cu formula:

$$H_p = I_m - 2 \cdot \frac{v_p \cdot v_p^T}{v_p^T \cdot v_p} \quad (3.28)$$

unde  $v$  este vectorul Householder de forma:  $v_p = (0, 0, \dots, v_{pp}, \dots, v_{np})$  (3.29)

iar  $I_m$  este matricea unitate de ordinul  $m$ . Reflectori Householder sunt simetrici

$$H_p^T = H_p \quad (3.30)$$

Produsul  $HA$  se realizează iterativ și cu fiecare pas matricea  $A$  devine parțial superior diagonală până la ordinul pasului. De exemplu, la pasul  $p$  matricea de ordinul  $p$  cuprinsă în matricea  $A$  devine superior diagonală. Reflectori Householder se calculează cu formula (3.28) care detaliată arată astfel:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & h_{pp} & \dots & h_{pn} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & h_{np} & \dots & h_{nn} \end{pmatrix} \text{ iar } h_{qr} = 1 - \frac{2}{\sum_{i=p}^n v_{ip}^2} v_{qp} \cdot v_{rp} \quad (3.31)$$

Notăm vectorul  $\alpha_i = (a_{1i}, a_{2i}, \dots, a_{ii}, \dots, a_{ni})^T$  (3.32)

$$H_p \cdot A_p = H_p \cdot [\alpha_1 \ \alpha_2 \ \dots \ \alpha_p \ \dots \ \alpha_n] = [H_p \alpha_1 \ H_p \alpha_2 \ \dots \ H_p \alpha_p \ \dots \ H_p \alpha_n] \quad (3.33)$$

Ținând cont că printr-o transformare ortogonală, suma pătratelor elementelor unei linii este invariantă putem scrie următoarea egalitate:

$$\sum_{i=1}^m a_{ip}^2 = \sum_{i=1}^{p-1} a_{ip}^2 + a_{ip}^2 \text{ din care rezultă } a_{ip} = \pm \sqrt{\sum_{i=p}^m a_{ip}^2} = \sqrt{s_p^2} \quad (3.34)$$

Un element al matricei din expresia (3.33) poate fi scris astfel:

$$H_p \alpha_p = \left( I_m - 2 \cdot \frac{v_p \cdot v_p^T \cdot \alpha_p}{v_p^T \cdot v_p} \right) \quad (3.35)$$

iar un element al expresiei (3.35) se poate exprima sub forma:

$$(H_p \cdot \alpha_p)_i = a_{ip} - \frac{2}{\sum_{i=p}^m v_{ip}^2} \sum_{i=p}^m a_{ip} \cdot v_{ip} = \begin{cases} a_{ip} & \text{dacă } i < p \\ 0 & \text{dacă } i > p \end{cases} \quad (3.36)$$

Se alege  $v_{ip} = a_{ip}$  dacă  $i > p$  (3.37)

În acest caz rezultă din expresia (3.36)  $v_{pp} = a_{pp} \pm s_p$  unde  $s_p$  se determină din relația (3.34) astfel ca  $v_{pp}$  să aibă valoarea maximă în valoare absolută, condiție îndeplinită dacă  $a_{pp}$  și  $s_p$  au același semn. În acest caz

$$s_p = \text{sign}(a_{pp}) \sqrt{\sum_{i=p}^m a_{ip}^2} \quad (3.38)$$

Componenta diagonală se calculează înlocuind în expresia (3.36) indicele  $i$  cu  $p$  și rezultă:

$$\dot{a}_{pp} = -s_p \quad (3.39)$$

Elementele de sub diagonală se fac zero iar cele situate în dreapta coloanei  $p$  se calculează astfel:

$$H_p \cdot \alpha_j = \alpha_j - \frac{2}{\sum_{i=p}^m v_{ip}^2} \cdot (v_p^T \cdot \alpha_j) \cdot v_p \quad (j > p) \quad (3.40)$$

sau pentru un element al matricei se poate scrie următoarea formulă de calcul :

$$((H_p \cdot \alpha_j)_j) = \begin{cases} a_{ij} & \text{dacă } i < p \\ a_{ij} - \frac{2}{\sum_{i=p}^m v_{ip}^2} \cdot \sum_{i=p}^m v_{ip} \cdot a_{ij} \cdot v_{ip} & \text{dacă } i \geq p \quad (j > p) \end{cases} \quad (3.41)$$

Matricea superior triunghiulară este dată de  $R = A_{n-1}$  iar matricea inferior triunghiulară este dată de  $Q = (H_n \cdot H_{n-1} \dots H_1 \cdot I_m)$ .

### 3.1.1.7.1. Algoritmul 3.7. Metoda factorizării QR

*{Variabile*

*A:matricea sistemului;*

*B:vectorul termenilor liberi;*

*X:vectorul soluțiilor;*

*Q:matricea ortogonală de ordinul mxm;*

*R:matricea superior triunghiulară de ordinul mxn;*

*k,i,n,m:indici, întregi;*

*{*

*Construiește matricea Q ca matrice unitate de ordinul m;*

*pentru k=1, ..., m*

*{calculează*

*pentru i=k, ..., m*

*{calculează  $s = a_{ik}^2$ ;*

*dacă  $a_{kk} < 0$  atunci  $s = -s$ ;*

*pentru i=1, ..., k-1  $v_i = 0$ ;*

*$v_k = a_{kk} + s$*

*$a_{kk} = -s$ ;*

*}*

*pentru i= k+1, ..., n*

*calculează  $v_i = a_{ik}$ ;*

*$a_{ik} = 0$ ;*

*$p = s * v_k$ ;*

*}*

*calculează  $A_{k+1} = Q_k * A_k$ ;*

```

         $b_{k+1} = Q_k * b_k ;$ 
         $Q^{(K)} = Q_k * q^{(\alpha-1)} ;$ 
    }
    rezolvă sistenu  $A_m * x = b_m ;$ 
    calculează  $R = A^T ;$ 
    calculează  $Q = (Q^{(m)})^T ;$ 
    Scrie soluțiile;
}

```

### 3.1.1.7.2. Implementarea algoritmului 3.7

```

/*Funcția care impementează metoda QR*/
void QR( int m,          /* nr de ecuații */
        int n,          /* nr de necunoscute m>n */
        double A[][NrMax], /*matricea sistemului */
        double x[],      /*vectorul necunoscutelor */
        double b[],      /*termenii liberi */
        double Q[][NrMax], /*matrice ortog. de ordin mxm */
        double R[][NrMax]) /*matrice sup. triun. de ordin mxn */
{
    int i,j,k;
    double s,sumap;
    static double v[NrMax];
    double p,t;
    for(i=1;i<=m;i++)
    for(j=1;j<=m;j++)
    {
        if(i==j)Q[i][j]=1;
        else Q[i][j]=0;
    }
    for(k=1;k<=n;k++)

    s=0;
    for(i=k;i<=m;i++)s+=pow(A[i][k],2);
    s=sqrt(s);
    if(A[k][k]<0) s=-s;
    for(i=1;i<=k-1;i++)v[i]=0;
    v[k]=A[k][k]+s;
    for(i=k+1;i<=m;i++)v[i]=A[i][k];

    /* Se transformă matricea A */

    A[k][k]=-s;
    for(i=k+1;i<=m;i++)A[i][k]=0;
}

```

```

p=s*v[k];
for(j=k+1;j<=n;j++)
{
t=0;
for(i=k;i<=m;i++)t+=v[i]*A[i][j]/p;
for(i=k;i<=m;i++)A[i][j]=A[i][j]-t*v[i];
}
/* Se transformă termenul liber */
t=0;
for(i=k;i<=m;i++) t+=v[i]*b[i]/p;
for(i=k;i<=m;i++) b[i]=b[i]-t*v[i];

/* Se transformă matricea Q */
for(j=1;j<=m;j++)
{
t=0;
for(i=k;i<=m;i++)t+=v[i]*Q[i][j]/p;
for(i=k;i<=m;i++)Q[i][j]=Q[i][j]-t*v[i];
}
}

/* Se rezolvă sistemul triunghiular */
for(i=n;i>=1;i--)
{
s=0;
for(j=i+1;j<=n;j++)s+=A[i][j]*x[j];
x[i]=(b[i]-s)/A[i][i];
}

/* Se calculează matricea R (identică cu A) */
for(i=1;i<=m;i++)
for(j=1;j<=n;j++)R[i][j]=A[i][j];

/* Se calculează matricea ortogonală Q */

for(i=1;i<=m;i++)
for(j=1;j<=m;j++)

{
s=Q[i][j];
Q[i][j]=Q[j][i];
Q[j][i]=s;
}
}

```



### 3.1.1.8. Metoda de rezolvare a sistemelor tridiagonale

Un tip special de sisteme liniare îl reprezintă sistemele tridiagonale care au diagonală principală și diagonalele vecine diferite de zero (3.30 ).

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & \dots & 0 & 0 & 0 \\ - & - & - & - & - & - & - & - \\ 0 & 0 & 0 & 0 & \dots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ - \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ - \\ d_{n-1} \\ d_n \end{pmatrix} \quad (3.42)$$

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

$$LU=A \quad (3.43)$$

Ecuția (3.43) poate fi scrisă detaliat astfel :

$$\begin{pmatrix} l_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ p_2 & l_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & p_3 & l_3 & \dots & 0 & 0 & 0 \\ - & - & - & - & - & - & - \\ 0 & 0 & 0 & \dots & p_{n-1} & l_{n-1} & 0 \\ 0 & 0 & 0 & \dots & 0 & p_n & l_n \end{pmatrix} \begin{pmatrix} 1 & u_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & u_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & u_3 & \dots & 0 & 0 \\ - & - & - & - & - & - & - \\ 0 & 0 & 0 & 0 & \dots & 1 & u_{n-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 \end{pmatrix} = \begin{pmatrix} b_1 & c_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_3 & b_3 & c_3 & \dots & 0 & 0 & 0 \\ - & - & - & - & - & - & - & - \\ 0 & 0 & 0 & 0 & \dots & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & 0 & \dots & 0 & a_n & b_n \end{pmatrix} \quad (3.44)$$

Făcând înmulțirea matricelor  $L, U$  și identificând cu matricea  $A$  se obțin relațiile de recurență pentru calculul elementelor matricelor  $L$  și  $U$ :

$$\begin{aligned} p_i &= a_i & i=2,3,\dots,n \\ u_i &= \frac{c_i}{l_i} & i=1,2,3,\dots,n \\ l_1 &= b_1 \\ l_i &= b_i - a_i u_{i-1} & i=2,3,\dots,n \end{aligned} \quad (3.45)$$

Condiția de existență a relațiilor este  $l_i \neq 0 \quad i = 1,2,\dots,n$

Sistemul (3.42) poate fi scris sub forma

$$LUX=D \quad (3.46)$$

sau sub forma a două sisteme

$$\begin{cases} L \cdot Y = D \\ U \cdot X = Y \end{cases} \quad (3.47)$$

Cunoaștem pe  $L$  și  $D$ , deci se calculează mai întâi  $Y$

$$\begin{pmatrix} l_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ a_2 & l_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & a_3 & l_3 & \dots & 0 & 0 & 0 \\ - & - & - & - & - & - & - \\ 0 & 0 & 0 & \dots & a_{n-1} & l_{n-1} & 0 \\ 0 & 0 & 0 & \dots & 0 & a_n & l_n \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ - \\ y_{n-1} \\ y_n \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ - \\ d_{n-1} \\ d_n \end{pmatrix} \quad (3.48)$$

Prin identificarea în (3.48) se deduc următoarele relații recursive de calcul :

$$\begin{cases} y_1 = \frac{d_1}{l_1} \\ y_i = \frac{(d_i - a_i y_{i-1})}{t_i}, \quad i=2,3,\dots,n \end{cases} \quad (3.49)$$

Din sistemul  $UX=Y$  scris detaliat :

$$\begin{pmatrix} 1 & u_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & 1 & u_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & 1 & u_3 & \dots & 0 & 0 & 0 \\ - & - & - & - & - & - & - & - \\ 0 & 0 & 0 & 0 & \dots & 1 & u_{n-1} & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & 1 & u_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ - \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ - \\ y_{n-1} \\ y_n \end{pmatrix} \quad (3.50)$$

rezultă relațiile recursive de calcul ale rădăcinilor. Prin retrosubstituție se obțin :

$$\begin{cases} x_n = y_n \\ x_i = y_i - s_i x_{i+1}, \quad i=n-1,\dots,1 \end{cases} \quad (3.51)$$

care reprezintă rădăcinile sistemului tridiagonal (3.42)

### 3.1.1.8.1. Algoritm 3.8. Sisteme tridiagonale

{Variabile

*a*: vectorul elementelor subdiagonale din matricea sistemului(vezi 3.42);

*b*: vectorul elementelor diagonale din matricea sistemului(vezi 3.42);

*c*: vectorul elementelor supradiagonale din matricea sistemului (vezi 3.42);

*l*: vectorul elementelor diagonale din matricea *L* (vezi 3.44);

*p*: vectorul elementelor subdiagonale din matricea *L* (vezi 3.44);

*u*: vectorul elementelor supradiagonale din matricea *U* (vezi 3.44);

*y* vectorul *UX*;

*x* vectorul soluțiilor;

*i*: indice, întreg;

*eroare*: semafor, boolean;

```

n: ordinul matricei, întreg;
{
    pentru i = 2,...,3 calculează  $p_i = a_i$  ;
    calculează  $l_1 = b_1$  ;
    dacă  $l_1 = 0$  atunci scrie "Sistemul nu poate fi rezolvat prin metoda
    aceasta " STOP;
    altfel calculează  $u_1 = \frac{c_1}{l_1}$  ;
    i=1 ; eroare = fals ;
    repetă
        i = i+1 ;
        calculează  $l_i = b_i - a_i s_{i-1}$  ;
        dacă  $l_i = 0$  atunci eroare = adevărată
            altfel calculează  $u_i = \frac{c_i}{l_i}$ 
    până când i = n-1 sau eroare = adevărată
    dacă eroare = adevărată atunci
        scrie "Sistemul nu poate fi rezolvat prin metoda
        aceasta " STOP
    altfel {
        calculează  $l_n = b_n - a_n u_{n-1}$ 
        calculează  $y_1 = \frac{d_1}{l_1}$ 
        pentru i = 1,2,3, ... , n calculează  $y_i = \frac{d_i - a_i y_{i-1}}{l_i}$ 

        calculează  $x_n = y_n$ 

        pentru i= n-1 .... 1 calculează  $x_i = y_i - s_i x_{i+1}$  ;
        tiparește soluțiile  $x_i$  i=1,2,...,n
    }
}

```

### 3.1.1.8.2. Implementarea algoritmului 3.8

```

/* Funcția care verifică dacă sistemul este sau nu tridiagonal
Funcția întoarce :
0 dacă sistemul nu este tridiagonal;
1 dacă sistemul este tridiagonal;
*/
int IsTridi(int or_mat,
double MAT[][NrMax] )

```

```

{
int i,j;

for(i=1;i<=or_mat;i++)
for(j=1;j<=or_mat;j++)
if( (abs(i-j)>1) && (MAT[i][j]!=0))return 0;
return 1;
}
/* Funcția care implementează metoda de rezolvare a sistemelor
liniare tridiagonale
Funcția întoarce :
1 dacă se găsesc soluțiile
0 în caz de eșec
*/

int SolveTridi(int or_mat,
               double a[],
               double b[],
               double c[],
               double TL[],
               double sol[])

{
int i;
static double p[NrMax],t[NrMax],s[NrMax],teta[NrMax];
for(i=2;i<=or_mat;i++)p[i]=a[i];
t[1]=b[1];if( t[1]==0 )return 0 ;
s[1]=c[1]/t[1];
teta[1]=TL[1]/t[1];
for(i=2;i<=or_mat-1;i++)
{
t[i]=b[i]-a[i]*s[i-1];
if(t[i]==0) return 0;
s[i]=c[i]/t[i];
teta[i]=( TL[i]-a[i]*teta[i-1])/t[i];
}
t[or_mat]=b[or_mat]-a[or_mat]*s[or_mat-1];
if( t[or_mat]==0 ) return 0;
teta[or_mat]=(TL[or_mat]-a[or_mat]*teta[or_mat-1])/t[or_mat];
sol[or_mat]=teta[or_mat];
for(i=or_mat-1;i>=1;i--)sol[i]=teta[i]-s[i]*sol[i+1];
return 1;
}

```

### 3.1.2. METODE INDIRECTE



Aceasta implică  $\lim_{k \rightarrow \infty} C^{k-1} = 0$ ;  $X = (I - C)^{-1}D$ ;  $X = CX + D$  unde  $X$  sunt toate soluțiile ecuației date (3.54).

Condiția ca toate valorile proprii ale unei matrici să fie subunitare este :

$$g_i = \sum_{\substack{j=1 \\ j \neq i}}^n \left| \frac{a_{ij}}{a_{ii}} \right| < 1 \text{ pentru } i = 1, 2, \dots, n \quad (3.57)$$

Această condiție trebuie îndeplinită de sisteme pentru a putea fi rezolvate prin metoda lui Jacobi.

### 3.1.2.1.1. Algoritm 3.9. Metoda Jacobi

```

{ Variabile
A: matricea sistemului;
B: matricea termenilor liberi;
S: matricea rezultată prin eliminarea elementului diagonal;
p: variabila care ține produsul elementelor de pe diagonală; real;
n, er: ordinul sistemului, er=1, A, er=0, F: întregi;
i, k: indici, întregi;
α: ține suma, real (3.57);
X: vectorul soluțiilor;
{
p=1;
pentru k = 1...n calculează p = p * ak,k;
dacă p = 0 atunci scrie "Sistemul nu se poate rezolva cu această
metodă " STOP.
i = 1;
er = 1
repetă
    calculează  $\alpha_i = \sum_{\substack{j=1 \\ j \neq i}}^n \left| \frac{a_{ij}}{a_{ii}} \right|$ ;
    dacă  $\alpha_i \geq 1$  atunci er = 0;
    i = i + 1;
până când (i = n + 1) sau (er = 1);
dacă er = 1 atunci {
    scrie "Sistemul nu se poate rezolva cu această metodă ";
    STOP;
}
altfel { k = 0;
    repetă
        i = 1;
    repetă

```

```

        calculează  $x_i^{(k)} = \frac{b_i}{a_{i,i}}$ 
        pentru  $j = 1 \dots n$ 
            calculează  $x_i^{(k)} = x_i^{(k)} - \frac{a_{ij}}{a_{i,i}} x_j^{(k-1)}$ ;
         $i = i + 1$ ;
    până când  $i = n + 1$ ;
     $k = k + 1$ ;
    până când  $|x_i^{(k)} - x_i^{(k-1)}| \leq \varepsilon \quad i = 1 \dots n$ 
    scrie soluțiile  $x_i = x_i^{(k)} \quad i = 1 \dots n$ 
    STOP
}}
}

```

### 3.1.2.1.2. Implementarea algoritmului 3.9

```

/* Funcția care verifică dacă o matrice este dominant diagonală
Funcția întoarce
0 în caz de eșec
1 în caz de succes */
int DD(int or_mat,
    double MAT[][NrMax])
{
    int i,j;
    double sum;

    for(i=1;i<=or_mat;i++)
    {
        sum=0;
        for(j=1;j<=or_mat;j++) sum+=fabs(MAT[i][j]);
        if ( 2*fabs(MAT[i][i])<=sum )return 0; }

    return 1;
}

/*Funcția care implementează metoda iterativă Jacobi
pentru rezolvarea sistemelor liniare.

Funcția întoarce
0 în caz de reușită;
1 dacă matricea nu e dominant diagonală
*/

int JACOBI(int or_mat,

```

```

double MAT[][NrMax],
double TL[],
double x0[],
double err)

{
  int i,j,k,sem;
  static double x1[NrMax];

  if( DD(or_mat,MAT)==0) return 1;

  do
  {
    sem=1;
    for(i=1;i<=or_mat;i++)
    {
      x1[i]=TL[i]/MAT[i][i];
      for(j=1;j<=or_mat;j++)if(j!=i)
      x1[i]-
      =(MAT[i][j]*x0[j])/MAT[i][i];
    }
    for(i=1;i<=or_mat;i++)
    {
      if( fabs(x1[i]-x0[i])>err)sem=0;
      x0[i]=x1[i];
    }
  }while(sem==0);
  return 0;
}

```

### 3.1.2.2. Metoda Gauss-Seidel

Această metodă diferă de metoda lui Jacobi prin faptul că la fiecare iterație se utilizează valorile calculate la pasul anterior pentru variabilele ale căror valori nu sunt cunoscute la pasul curent și valorile de la pasul curent pentru variabilele calculate. Astfel formula de calcul (3.53) devine :

$$x_i^{(k)} = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} x_j^{(k)} - \sum_{j=i+1}^n \frac{a_{ij}}{a_{ii}} x_j^{(k-1)} \quad (3.58)$$

$$i = 1, 2, \dots, n$$

$$k = 1, 2, \dots, n, \dots$$

În rest, metoda se identifică cu metoda lui Jacobi.

Pentru același grad de precizie, viteza de convergență a metodei lui Gauss-Seidel este de două ori mai mare decât viteza de convergență a metodei lui Iacobi .



**3.1.2.2.1. Algoritm 3.10. Metoda Gauss-Seidel**

```

{ Variabile
  A: matricea sistemului;
  B: matricea termenilor liberi;
  S: matricea rezultată prin eliminarea elementului diagonal;
  p: variabila care ține produsul elementelor de pe diagonală; real;
  n,er:ordinul sistemului,er=1,A, er=0, F:întregi;
  i,k: indici, întregi;
  α: ține suma, real (3.57);
  X: vectorul soluțiilor;

{
  p=1;
  pentru k = 1...n calculează p= p • ak,k ;
  dacă p= 0 atunci scrie "Sistemul nu se poate rezolva cu această
  metodă "
                                STOP.

    i =1;
    er =1
    repetă
      calculează  $\alpha_i = \sum_{\substack{j=1 \\ j \neq i}}^n \left| \frac{a_{ij}}{a_{ii}} \right|$ ;

      dacă  $\alpha_i \geq 1$  atunci er = 0 ;
      i = i+1;
    până când (i = n+1) sau (er = 1);
    dacă er = 1 atunci {
      scrie "Sistemul nu se poate rezolva cu această metodă ";
                                STOP;
    }

    altfel {
      k = 0;
      repetă

        i =1;
        repetă
          calculează  $x_i^{(k)} = \frac{b_i}{a_{i,i}}$ 

          pentru j= 1... n
          {
            dacă j < i atunci
              calculează  $x_i^{(k)} = x_i^{(k)} - \frac{a_{i,j}}{a_{i,i}} x_j^{(k)}$  ;

```

```

        dacă  $j > i$  atunci
            calculează  $x_i^{(k)} = x_i^{(k)} - \frac{a_{i,j}}{a_{i,i}} x_j^{(k-1)}$ ;
        }
         $i = i + 1$ ;
        până când  $i = n + 1$ ;
         $k = k + 1$ ;
        până când  $|x_i^{(k)} - x_i^{(k-1)}| \leq \varepsilon$        $i = 1 \dots n$ 
        scrie soluțiile  $x_i = x_i^{(k)}$        $i = 1 \dots n$ 
        STOP
    }
}

```

### 3.1.2.2.2. Implementarea algoritmului 3.10

*/\* Funcția care verifică dacă o matrice este dominant diagonală*

*Funcția întoarce*

*0 în caz de eșec*

*1 în caz de succes \*/*

```

int DD(int or_mat,
       double MAT[][NrMax])
{
    int i,j;
    double sum;

    for(i=1;i<=or_mat;i++)
    {
        sum=0;
        for(j=1;j<=or_mat;j++) sum+=fabs(MAT[i][j]);
        if ( 2*fabs(MAT[i][i])<=sum )return 0;
    }
    return 1;
}

```

*/\* Funcția implementează metoda iterativă Gauss-Seidel  
pentru rezolvarea sistemelor liniare*

*Funcția întoarce*

*0 în caz de reușită;*

*1 dacă matricea nu e dominant diagonală*

```

*/

int GAUSS_SEIDEL(int or_mat,
                  double MAT[][NrMax],
                  double TL[],
                  double x0[],
                  double err)
{
    int i,j,k,sem;
    double x1[NrMax],sum;

    for(j=1;j<=or_mat;j++)
        if( DD(or_mat,MAT)==0) return 1;
    do
    {
        sem=1;
        for(i=1;i<=or_mat;i++)
        {
            x1[i]=TL[i]/MAT[i][i];
            {
                if (j<i) x1[i]-(MAT[i][j]*x1[j])/MAT[i][i];
                if (j>i) x1[i]-(MAT[i][j]*x0[j])/MAT[i][i];
            }
        }
        for(i=1;i<=or_mat;i++)
        {
            if( fabs(x1[i]-x0[i])>err)sem=0;
            x0[i]=x1[i];
        }
    }
    while(sem==0);
    return 0;
}

```

### 3.2. REZOLVAREA NUMERICĂ A SISTEMELOR NELINIARE

În acest paragraf se studiază rezolvarea numerică a sistemelor de ecuații neliniare.

Fie  $f(x) = 0 \quad f: X \rightarrow Y, X, Y \subset \mathbb{R}^n$  (3.59)

un sistem dat sub formă vectorială. Vectorul  $x$  are  $n$  componente. Funcția  $f$  are următoarele componente  $f_1, f_2, f_3 \dots f_n$ ,  $n$  funcții definite pe domeniul  $X \subset \mathbb{R}^n$ . Sistemul scris cu ajutorul componentelor vectorilor se prezintă astfel:

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \dots\dots\dots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases} \quad (3.60)$$

Dacă cel puțin una din funcțiile  $f_1, f_2, f_3 \dots f_n$  sunt neliniare în variabilele  $x_1, x_2, \dots, x_n$  sistemul de ecuații (3.59) sau (3.60) se numește sistem de ecuații neliniare.

### 3.2.1. METODA LUI NEWTON DE REZOLVARE A ECUAȚIILOR NELINIARE

Presupunem că în vecinătatea  $V \subset X$  există o soluție unică  $(\overline{x_1}, \overline{x_2}, \dots, \overline{x_n})$  a sistemului  $f_i(\overline{x_1}, \overline{x_2}, \dots, \overline{x_n}) = 0$  pentru  $i = 1, 2, \dots, n$  (3.61)

Dacă  $f_i(x_1, x_2, \dots, x_n)$  și  $\frac{\partial f_i}{\partial x_j}$ ;  $i, j = 1, 2, \dots, n$  sunt continue pe  $V$  și iacobianul

$$F(x) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \dots\dots\dots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad (3.62)$$

are determinantul  $\det(F(x)) \neq 0$  pentru  $(\overline{x_1}, \overline{x_2}, \dots, \overline{x_n})$  atunci funcția  $f: X \rightarrow Y$  este o transformare regulată într-o vecinătate a soluției și are o inversă care are aceeași proprietate.

În acest caz putem aplica metoda Newton pentru sisteme ca o funcție de iterație ce poate fi scrisă

$$x_{k+1} = x_k - [F(x_k)]^{-1} \cdot f(x_k), \quad k = 1, 2, \dots \quad (3.63)$$

Prelucrând această ecuație se poate aduce la forma :

$$\sum_{j=1}^n \frac{\partial f_i(x_k)}{\partial x_j} (x_{jk+1} - x_{jk}) = -f_i(x_k), \quad i = 1, 2, \dots, n, \quad k = 0, 1, 2, \dots \quad (3.64)$$

Această ecuație reprezintă dezvoltarea în serie Taylor a funcției  $f(x_1, x_2, \dots, x_n)$  în vecinătatea punctului  $(x_k, x_{2k}, \dots, x_{nk})$  utilizând termenii până la derivata parțială de

ordinul doi, exclusiv. Dacă notăm  $x_{j,k+1} - x_{j,k} = \Delta_{j,k}$   $j = 1, 2, \dots, n$  sistemul (3.64) se poate scrie matriceal astfel :

$$\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \bigg|_{x=x_k} \begin{pmatrix} \Delta X_{1K} \\ \Delta X_{2K} \\ \vdots \\ \Delta X_{nK} \end{pmatrix} = \begin{pmatrix} -f_1(x) \\ -f_2(x) \\ \vdots \\ -f_n(x) \end{pmatrix} \bigg|_{x=x_k} \quad (3.65)$$

Sistemul 3.65 este un sistem linear în necunoscutele  $\Delta X_{1K}, \Delta X_{2K}, \dots, \Delta X_{nK}$ . Din aceste soluții se determină soluția sistemului la pasul  $k+1$ ,

$$x_{j,k+1} = x_{j,k} + \Delta X_{j,k}, \quad j = 1, 2, \dots, n \quad (3.66)$$

Procesul iterativ se continuă până când

$$|\Delta_{jk}| < \varepsilon \quad j = 1, 2, \dots, n \quad \text{unde } \varepsilon > 0 \text{ este eroarea impusă de operator.}$$

La baza convergenței șirului  $\Delta_{jk}$   $j = 1, 2, \dots, n$  stă următoarea teoremă :

**Teorema 1.** (Teorema lui Kantorovici)

Dacă în sfera închisă

$$u(x_0, \gamma) = \{x \in V, \|x - x_0\| \leq \gamma\} \quad (3.67)$$

sunt satisfăcute condițiile :

a) Matricea funcțională  $F(x) = \left( \frac{\partial f}{\partial x_j} \right)$  pentru  $x = x_0$  are o inversă  $\Gamma_0 = (F(x_0))^{-1}$ ,

cu proprietatea ca  $\|\Gamma_0\| \leq A_0$  ;

b)  $\|\Gamma_0 f(x_0)\| = \|x_k - x_0\| \leq B_0$ ;

c)  $\sum_{p=1}^n \left| \frac{\partial^2 f_i(x)}{\partial x_j \cdot \partial x_p} \right| \leq C$  ,  $i, j = 1, 2, \dots, n$  și  $x \in u(x_0, g)$  ;

d) Constantele  $A_0, B_0$ , și  $C_0$  satisfac inegalitatea

$$h_0 = 2nA_0 B_0 C \leq 1$$

Atunci pentru punctul de start  $x_0$ , șirul iterativ :

$$x_{k+1} = x_k - (F(x_k))^{-1} f(x_k) \quad k = 0, 1, 2, \dots$$

este convergent către soluția  $\alpha$ . Viteza de convergență este dată de relația

$$\|x_k - \alpha\| \leq \left(\frac{1}{2}\right)^{k-1} h_0^{2^{p-1}} B_0.$$

Demonstrația teoremei nu o prezentăm, ea putând fi găsită în literatura de specialitate.

### 3.2.1.1. Algoritmul 3.11 pentru sisteme neliniare. Metoda lui Newton

```

{ Variabile
  X : valorile soluțiilor iterative , vector ;
  ΔX : valorile de modificare a soluțiilor iterative , vector ;
  X0 : soluția de start a sistemului , vector ;
  det F : determinanatul matricei F(x) ;
  {
    k = 0;
    Dacă det F = det F( xk ) = 0 atunci scrie “Alege altă soluție de start
    “
    STOP
    altfel
    { repetă
      k = k+1;
      calculează F ( xk-1 ) ;

    { rezolvă sistemul liniar

      
$$\sum_{j=1}^n \frac{\partial f_i(x_{k-1})}{\partial x_j} [x_{j,k} - x_{j,k-1}] = -f_i(x_{k-1}) \quad i = 1, 2, \dots, n$$

      calculează
      
$$x_{j,k} = \Delta x_{j,k-1} + x_{j,k-1}, \quad j = 1, 2, \dots, n$$

    }
    până când
       $|x_{j,k} - x_{j,k-1}| \leq \varepsilon, \quad j = 1, 2, \dots, n \quad \text{sau} \quad F(x_{k-1}) = 0$ 
    }
    Soluțiile sistemului sunt
  }
}

```

### 3.2.1.2. Implementarea algoritmului 3.11

```

/* Funcția care implementează metoda lui NEWTON de rezolvare a
sistemelor neliniare cu două ecuații și două necunoscute.
Funcția întoarce
0 dacă determinantul este 0 sau nu s-a atins precizia
1 dacă se obțin soluțiile
*/

int RezSistNelin( double(* ec1)(double x,double y),
/* prima ecuație din sistem */

```

```

double(* ec2)(double x,double y),
/* a doua ecuație din sistem */
double(* dec1x)(double x,double y),
/*derivata primei ecuații în raport cu prima variabilă
double(* dec1y)(double x,double y),
/*derivata primei ecuații în raport cu a doua variabilă */
double(* dec2x)(double x,double y),
/*derivata ecuației a doua în raport cu prima variabilă */
double(* dec2y)(double x,double y),
/*derivata ecuației a doua în raport cu a doua variabilă */
double eroare,
/* eroarea de calcul a soluției */
int NITER, /* numărul maxim de iterații */
double *startx, /* x start */
double *starty) /* y start */
{

int i ;
double deltax,deltay,det,detx,dety,xn_1,xn,yn_1,yn;

i=0;
xn=*startx;
yn=*starty;
do
{
xn_1=xn;
yn_1=yn;
det=dec1x(xn_1,yn_1)*dec2y(xn_1,yn_1)-
dec2x(xn_1,yn_1)*dec1y(xn_1,yn_1);
detx=-
ec1(xn_1,yn_1)*dec2y(xn_1,yn_1)+ec2(xn_1,yn_1)*dec1y(xn_1,yn_1);
dety=-
dec1x(xn_1,yn_1)*ec2(xn_1,yn_1)+dec2x(xn_1,yn_1)*ec1(xn_1,yn_1);
if (det==0) return 0;
deltax=detx/det;
deltay=dety/det;
xn=xn_1+deltax;
yn=yn_1+deltay;
}
while(((fabs(xn-xn_1)>eroare) || (fabs(yn-yn_1)>eroare) &&
i<=NITER));
if(i>=NITER) return 0;
*startx=xn;
*starty=yn;
return 1;
}

```

### 3.3. APLICAȚII

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

$$R_1 = 2.4k\Omega, R_2 = 3.6k\Omega, R_3 = 1.5k\Omega, R_4 = 1k\Omega,$$

$$R_5 = 1.8k\Omega, R_6 = 2.7k\Omega, R_7 = 1.2k\Omega, R_8 = 3k\Omega,$$

$$E_1 = 10V, E_2 = 12V, E_3 = 15V$$

Se cer curenții prin fiecare latură a circuitului.

Aplicând teoremele lui Kirchhoff pentru  $N-1$  noduri deci 3 și pentru  $O=L-N+1$  ochiuri fundamentale deci  $O=4$ , unde  $N$  reprezintă numărul total de noduri ale circuitului, iar  $L$  numărul total de laturi ale circuitului, rezultă următorul sistem liniar de șapte ecuații cu șapte necunoscute. Necunoscutele sunt curenții din cele șapte laturi ale circuitului, matricea necunoscutelor sistemului este formată din valorile rezistoarelor din circuit, iar termenii liberi ai sistemului sunt valorile surselor de alimentare.

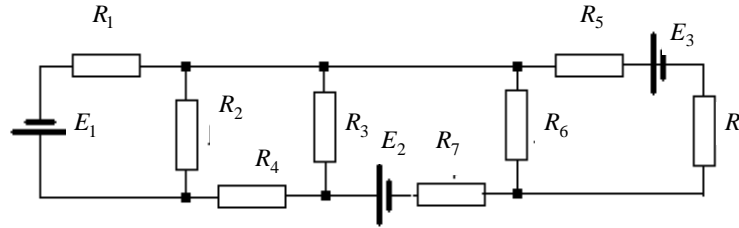


Fig.3.1. Circuit electric cu surse de cc și rezistoare

$$\begin{cases} R_1 \cdot I_1 - R_2 \cdot I_2 + 0 \cdot I_3 + 0 \cdot I_4 + 0 \cdot I_5 + 0 \cdot I_6 + 0 \cdot I_7 = -E_1 \\ 0 \cdot I_1 + R_2 \cdot I_2 - R_3 \cdot I_3 - R_4 I_4 + 0 \cdot I_5 + 0 \cdot I_6 + 0 \cdot I_7 = 0 \\ 0 \cdot I_1 + 0 \cdot I_2 + R_3 \cdot I_3 + 0 \cdot I_4 + 0 \cdot I_5 - R_6 \cdot I_6 - R_7 \cdot I_7 = E_2 \\ 0 \cdot I_1 + 0 \cdot I_2 + 0 \cdot I_3 + 0 \cdot I_4 + R_5 \cdot I_5 + R_6 \cdot I_6 + 0 \cdot I_7 = -E_3 \\ I_1 + I_2 + I_3 + 0 \cdot I_4 - I_5 + I_6 + 0 \cdot I_7 = 0 \\ I_1 + I_2 + 0 \cdot I_3 + I_4 + 0 \cdot I_5 + 0 \cdot I_6 + 0 \cdot I_7 = 0 \\ 0 \cdot I_1 + 0 \cdot I_2 + 0 \cdot I_3 + 0 \cdot I_4 + I_5 - I_6 + I_7 = 0 \end{cases}$$

Introducând valorile numerice ale rezistențelor și ale surselor se obține sistemul:

$$\begin{cases} 2.4 \cdot I_1 - 3.6 \cdot I_2 + 0 \cdot I_3 + 0 \cdot I_4 + 0 \cdot I_5 + 0 \cdot I_6 + 0 \cdot I_7 = -10 \\ 0 \cdot I_1 + 3.6 \cdot I_2 - 1.5 \cdot I_3 - I_4 + 0 \cdot I_5 + 0 \cdot I_6 + 0 \cdot I_7 = 0 \\ 0 \cdot I_1 + 0 \cdot I_2 + 1.5 \cdot I_3 + 0 \cdot I_4 + 0 \cdot I_5 - 2.7 \cdot I_6 - 1.2 \cdot I_7 = 12 \\ 0 \cdot I_1 + 0 \cdot I_2 + 0 \cdot I_3 + 0 \cdot I_4 + 4.8 \cdot I_5 + 2.7 \cdot I_6 + 0 \cdot I_7 = -15 \\ I_1 + I_2 + I_3 + 0 \cdot I_4 - I_5 + I_6 + 0 \cdot I_7 = 0 \\ I_1 + I_2 + 0 \cdot I_3 + I_4 + 0 \cdot I_5 + 0 \cdot I_6 + 0 \cdot I_7 = 0 \\ 0 \cdot I_1 + 0 \cdot I_2 + 0 \cdot I_3 + 0 \cdot I_4 + I_5 - I_6 + I_7 = 0 \end{cases}$$



Determinantul matricei sistemului este  $d=683.834400$

Soluțiile calculate cu ajutorul metodei eliminării lui Gauss sunt:

$$i_1 = 1.685145 ; i_2 = 1.654348 ; i_3 = 3.949904 ; i_4 = 0.030797 ; i_5 = 3.410878 ; i_6 = -0.508228 ;$$

$$i_7 = 3.919107$$

Semnele minus arată că sensul curenților luați pentru scrierea ecuațiilor lui Kirchhoff este contrar celor din realitate. Sistemul nu îndeplinește condiția pentru a fi rezolvat cu metodele iterative .

2. Se consideră sistemul :

$$4.4x_1 - 0.4x_2 + x_3 + 0.7x_4 = 3.2$$

$$0.23x_1 + 5.4x_2 - 1.5x_3 + 0.1x_4 = 2.7$$

$$1.3x_1 + 0.34x_2 + 6.2x_3 - 0.3x_4 = 1.4$$

$$0.3x_1 + 1.4x_2 - 0.34x_3 + 2.8x_4 = 2.1$$

Soluțiile sistemului obținute cu metoda eliminării lui Gauss sunt:

$$x_1 = 0.683916 ; x_2 = 9.484220 ; x_3 = 0.077529 ; x_4 = 0.444124 ; \quad d=393.832664 ;$$

Prin metoda lui Jacobi, pentru care sistemul îndeplinește condiția de rezolvare, se obțin următoarele soluții:  $x_1 = 0.683916 ; x_2 = 9.484220 ; x_3 = 0.077529 ; x_4 = 0.444124 ;$

luând soluțiile de start:  $x_1 = 1 ; x_2 = 1 ; x_3 = 1 ; x_4 = 1 .$

# 4

## DERIVAREA NUMERICĂ\*

În multe probleme tehnice este necesar să se cunoască derivata într-un punct a unui semnal eșantionat. Rezultatele multor experimentări electronice sunt date sub formă de funcții tabelate:  $u = f(i)$ ,  $i = g(u)$  etc. Calculul derivatelor acestor funcții într-un anumit punct ne poate da informații asupra unui parametru al circuitului cum ar fi rezistența dinamică, inversul ei etc. Pentru determinarea derivatei funcțiilor tabelate suntem nevoiți să utilizăm metodele numerice.

Metodele numerice de calcul al derivatei într-un punct dat, se pot aplica și funcțiilor a căror expresie analitică este cunoscută, obținând în acest mod tangenta la curbă în punctul de calcul.

### 4.1. DERIVATA NUMERICĂ PRIN DOUĂ PUNCTE

Prin definiție derivata funcției  $f(x)$  în punctul  $x_0$ , dacă există, este :

$$f'(x_0) = \lim_{\Delta x} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \quad (4.1)$$

Pentru o valoare mică a lui  $\Delta x$  calculul expresiei :

$$D^+ = \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \quad (4.2)$$

poate aproxima destul de bine derivata funcției în punctul  $x_0$ ,  $f'(x_0)$ . Valoarea lui  $\Delta x$  poate fi și negativă, derivata funcției  $f'(x_0)$  putând fi aproximată și cu expresia:

$$D^- = \frac{f(x_0 - \Delta x) - f(x_0)}{-\Delta x} \quad (4.3)$$

În figura 4.1 se observă că la limită pentru  $\Delta x \rightarrow 0$  ambele corzi ale curbei, PQ și QR, devin tangente la curbă în  $x_0$ , QT. Dacă se consideră coarda PR la curbă, se observă că ea aproximează mult mai bine derivata funcției în punctul  $x_0$ .

$$f'(x_0) \approx \frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x} = \frac{D^+ + D^-}{2} \quad (4.4)$$

Dacă notăm  $2\Delta x = h$ , se obține derivata numerică în  $x_0$ :

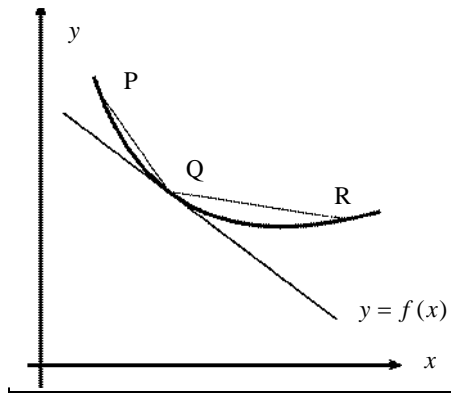
---

\*)Bibliografie: [6], [7], [15]

$$f'(x_0) \approx \frac{f\left(x_0 + \frac{h}{2}\right) - f\left(x_0 - \frac{h}{2}\right)}{h} \quad (4.5)$$

Deoarece în formula de calcul a derivatei numerice intervin numai două puncte  $x_0 - h/2$  și  $x_0 + h/2$ , numim această formulă derivata prin două puncte a funcției  $f(x)$ .

Pentru calculul derivatelor de ordin superior se aplică formula (4.5) pentru funcția  $f'(x)$ . Pentru o mai bună înțelegere se face un calcul complet al derivatei de ordinul doi și trei a metodei de calcul a derivatelor de ordin superior.



$$\begin{aligned} f''(x_0) &\approx \frac{f'\left(x_0 + \frac{h}{2}\right) - f'\left(x_0 - \frac{h}{2}\right)}{h} \\ &= \frac{f(x_0 + h) - f(x_0) - f(x_0) + f(x_0 - h)}{h^2} \\ &= \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{h^2} \quad (4.6) \\ f'''(x_0) &\approx \frac{f''\left(x_0 + \frac{h}{2}\right) - f''\left(x_0 - \frac{h}{2}\right)}{h} \end{aligned}$$

$$\begin{aligned} &= \frac{f\left(x_0 + \frac{3}{2}h\right) - 2f\left(x_0 + \frac{h}{2}\right) + f\left(x_0 - \frac{h}{2}\right) - f\left(x_0 + \frac{h}{2}\right) + 2f\left(x_0 - \frac{h}{2}\right) - f\left(x_0 - \frac{3}{2}h\right)}{h^3} \\ &= \frac{f\left(x_0 + \frac{3}{2}h\right) - f\left(x_0 + \frac{h}{2}\right) - f\left(x_0 - \frac{h}{2}\right) + f\left(x_0 - \frac{3}{2}h\right)}{h^3} \quad (4.7) \end{aligned}$$

În acest mod se poate obține orice derivată de ordin superior.

#### 4.1.1. EROAREA DE TRUNCHIERE A DERIVATEI PRIN DOUĂ PUNCTE

Dezvoltăm în serie Taylor funcția  $f(x)$  în jurul punctului  $x_0$ , reținând primii trei termeni :

$$f(x) = f(x_0) + \frac{x-x_0}{1!} f'(x_0) + \frac{(x-x_0)^2}{2!} f''(x_0) + \frac{(x-x_0)^3}{3!} f'''(\xi) \quad (4.8)$$

unde  $\xi \in [x, x_0]$ . Substituim  $x = x_0 + \frac{h}{2}$  în dezvoltarea lui  $f(x)$ , (4.8).

$$f\left(x_0 + \frac{h}{2}\right) = f(x_0) + \frac{h}{2} f'(x_0) + \frac{h^2}{8} f''(x_0) + \frac{h^3}{48} f'''(\xi_1) \quad (4.9)$$

unde  $\xi_1 \in \left[x_0, x_0 + \frac{h}{2}\right]$ .

Substituim  $x = x_0 - \frac{h}{2}$  în dezvoltarea lui  $f(x)$ , (4.8)

$$f\left(x_0 - \frac{h}{2}\right) = f(x_0) - \frac{h}{2} f'(x_0) + \frac{h^2}{8} f''(x_0) - \frac{h^3}{48} f'''(\xi_2) \quad (4.10)$$

unde  $\xi_2 \in \left[x_0 - \frac{h}{2}, x_0\right]$ . Scăzând membru cu membru relațiile (4.9) și (4.10) rezultă:

$$\begin{aligned} f'(x_0) &= \frac{f\left(x_0 + \frac{h}{2}\right) - f\left(x_0 - \frac{h}{2}\right)}{h} - \frac{h^2}{24} \cdot \frac{f'''(\xi_1) + f'''(\xi_2)}{2} = \\ &= \frac{f\left(x_0 + \frac{h}{2}\right) - f\left(x_0 - \frac{h}{2}\right)}{h} - \frac{h^2}{24} \cdot f'''(\xi) \end{aligned} \quad (4.11)$$

unde  $\xi \in \left[x_0 - \frac{h}{2}, x_0 + \frac{h}{2}\right]$ ,

rezultă că eroarea de trunchiere a derivatei prin două puncte este :

$$e_T = \frac{h^2}{24} \cdot f'''(\xi) \quad (4.12)$$

Dacă  $|f'''(\xi)| < M$  pentru orice  $x \in \left[x_0 - \frac{h}{2}, x_0 + \frac{h}{2}\right]$  atunci  $e_T < \frac{h^2}{24} \cdot M$

#### 4.1.2. ALGORITMUL 4.1. DERIVATA PRIN DOUĂ PUNCTE

{Variabile

$x_0$  : punctul în care se calculează derivata , real;

$h$  : pasul, real ;

$der$  : variabila derivatei, real;

{

calculează  $der = \frac{f(x_0 + 0.5h) - f(x_0 - 0.5h)}{h}$ ;

}

Derivata este  $der$  ;

}

### 4.1.3. IMPLEMENTAREA ALGORITMULUI 4.1

```

/* Funcția care implementează derivata prin două puncte.
Funcția întoarce valoarea derivatei
*/
double Derivata2P( double (*f) (double),
                  double x0,
                  double h)
{
    return((f(x0+0.5*h)-f(x0-0.5*h))/h);
}

```

## 4.2. DERIVATA NUMERICĂ PRIN TREI PUNCTE

Deoarece în calculul acestei derivate numerice intervin trei puncte o denumim derivata numerică prin trei puncte .

Se pune problema determinării derivatei numerice a funcției  $f(x)$  în punctul  $x_0$ , sub forma:

$$f'(x_0) = af(x_0 - h_1) + bf(x_0) + cf(x_0 + h_2) \quad (4.13)$$

unde  $x_0, h_1, h_2$  sunt date cunoscute, iar  $a, b, c$  necunoscute. Pentru determinarea acestor necunoscute vom considera funcția  $f(x)$ , o constantă, liniară și pătratică .

Pentru  $f(x) = c \Rightarrow f'(x) = 0, f'(x_0) = 0$ , iar ecuația (4.13) devine:

$$0 = a + b + c, \quad (4.14)$$

pentru  $f(x) = (x - x_0) \Rightarrow f'(x) = 1, f'(x_0) = 1$  și

$$1 = -h_1a + 0b + h_2c, \quad (4.15)$$

pentru  $f(x) = (x - x_0)^2 \Rightarrow f'(x) = 2(x - x_0), f'(x_0) = 0$  și

$$0 = h_1^2a + 0b + h_2^2c \quad (4.16)$$

Deoarece  $h_1, h_2 > 0$  și determinantul sistemului  $\Delta = h_1 h_2 (h_1 + h_2) \neq 0$ , sistemul este unic determinat. Rezolvând sistemul, se obțin valorile necunoscutelor :

$$a = \frac{-h_2}{h_1(h_1 + h_2)}, \quad b = \frac{h_2^2 - h_1^2}{h_1 h_2 (h_1 + h_2)}, \quad c = \frac{h_1}{h_2(h_1 + h_2)} \quad (4.17)$$

Formula derivatei numerice a funcției  $f(x)$  prin trei puncte este :

$$f'(x_0) \approx \frac{h_1^2 f(x_0 + h_2) + (h_2^2 - h_1^2) f(x_0) - h_2^2 f(x_0 - h_1)}{h_1 h_2 (h_1 + h_2)} \quad (4.18)$$

care dă o eroare de trunchiere nulă pentru funcțiile constante, liniare și pătratice. Pentru  $h_1 = h_2 = h/2$  se obține formula derivatei numerice prin două puncte.

### 4.2.1. EROAREA DE TRUNCHIERE A DERIVATEI PRIN TREI

### PUNCTE

Pentru funcțiile de grad mai mare ca doi, eroarea de trunchiere o determinăm din dezvoltarea în serie Taylor a funcției  $f(x)$  :

$$f(x) = f(x_0) + \frac{(x-x_0)}{1!} f'(x_0) + \frac{(x-x_0)^2}{2!} f''(x_0) + \frac{(x-x_0)^3}{3!} f'''(\xi) \quad (4.19)$$

unde  $\xi \in [x, x_0]$ .

Substituim  $x = x_0 - h_1$  și rezultă:

$$f(x_0 - h_1) = f(x_0) - \frac{h_1}{1!} f'(x_0) + \frac{h_1^2}{2!} f''(x_0) - \frac{h_1^3}{3!} f'''(\xi_1) \quad (4.20)$$

unde  $\xi_1 \in [x_0 - h_1, x_0]$ .

$$f(x_0 + h_2) = f(x_0) + \frac{h_2}{1!} f'(x_0) + \frac{h_2^2}{2!} f''(x_0) + \frac{h_2^3}{3!} f'''(\xi_2) \quad (4.21)$$

unde  $\xi_2 \in [x_0, x_0 + h_2]$ .

Se înmulțește egalitatea (4.20) cu  $-h_2^2$  și egalitatea (4.21) cu  $h_1^2$  și se adună. Rezultă :

$$f'(x_0) \approx \frac{h_1^2 f(x_0 + h_2) + (h_2^2 - h_1^2) f(x_0) - h_2^2 f(x_0 - h_1)}{h_1 h_2 (h_1 + h_2)} - \frac{h_1 h_2}{6 (h_1 + h_2)} [h_2 f'''(\xi_1) + h_1 f'''(\xi_2)] \quad (4.22)$$

Eroarea de trunchiere este dată de formula :

$$e_T = \frac{h_1 h_2}{6 (h_1 + h_2)} [h_2 f'''(\xi_1) + h_1 f'''(\xi_2)] \quad (4.23)$$

Dacă  $|f'''(x)| < M$  pentru orice  $x \in [x_0 - h, x_0 + h_2]$  atunci

$$e_T \leq \frac{h_1 h_2}{6} M \quad (4.24)$$

#### 4.2.2. ALGORITMUL 4.2. DERIVATA PRIN TREI PUNCTE

{Variabile

$x_0$  : punctul în care se calculează derivata , real ;

$h_1$  : distanța punctului din stânga față de  $x_0$ , real ;

$h_2$  : distanța punctului din dreapta față de  $x_0$ , real ;

der : variabila derivatei , real;

{

$$\text{calculează} \quad der = \frac{h_1^2 f(x_0 + h_2) + (h_2^2 - h_1^2) f(x_0) - h_2^2 f(x_0 - h_1)}{h_1 h_2 (h_1 + h_2)} ;$$

}

### 4.2.3. IMPLEMENTAREA ALGORITMULUI 4.2

```

/* Funcția care implementează derivata prin trei puncte.
   Funcția întoarce valoarea derivatei      }
*/
double Derivata3P( double (*f) (double),
                  double x0,
                  double h1,
                  double h2)
{
return(h1*h1*f(x0+h2)+(h2*h2-h1*h1)*f(x0)-h2*h2*f(x0-
h1))/(h1*h2*(h1+h2));
}

```

### 4.3. DERIVATA NUMERICĂ PRIN CINCI PUNCTE

Pentru formula acestei derivate se utilizează cinci puncte, două câte două egal distanțate de punctul central, în care se calculează derivata :

$$x_0 - 2h, x_0 - h, x_0, x_0 + h, x_0 + 2h \quad (4.25)$$

Formula derivatei numerice prin cinci puncte se caută sub forma :

$$f'(x_0) = af(x_0 - 2h) + bf(x_0 - h) + cf(x_0) + df(x_0 + h) + ef(x_0 + 2h) \quad (4.26)$$

Pentru determinarea necunoscutelor  $a, b, c, d, e$ , particularizăm funcția  $f(x)$  cu funcție: constantă, liniară, pătratică, de gradul trei și de gradul patru .

Pentru  $f(x) = c \Rightarrow f'(x) = 0, f'(x_0) = 0$  și ecuația (4.26) devine:

$$0 = a + b + c + d + e \quad (4.27)$$

Pentru  $f(x) = x - x_0 \Rightarrow f'(x) = 1, f'(x_0) = 1$  și

$$1 = -2ha - hb + 0c + hd + 2he \quad (4.28)$$

Pentru  $f(x) = (x - x_0)^2 \Rightarrow f'(x) = 2(x - x_0), f'(x_0) = 0$  și

$$0 = 4h^2a + h^2b + 0c + h^2d + 4h^2e \quad (4.29)$$

Pentru  $f(x) = (x - x_0)^3 \Rightarrow f'(x) = 3(x - x_0)^2, f'(x_0) = 0$  și

$$0 = -8h^3a - h^3b + 0c + h^3d + 8h^3e \quad (4.30)$$

Pentru  $f(x) = (x - x_0)^4 \Rightarrow f'(x) = 4(x - x_0), f'(x_0) = 0$  și

$$0 = 16h^4a + h^4b + 0c + h^4d + 16h^4e \quad (4.31)$$

Determinantul sistemului format de ecuațiile (4.27), (4.28), (4.29), (4.30), (4.31) este  $\Delta = 144h^{10} \neq 0$  pentru că  $h > 0$ . Ca urmare sistemul dat este unic determinat și are soluțiile :

$$a = 12h^9, b = -96h^9, c = 0, d = 96h^9, e = -12h^9 \quad (4.32)$$

Formula derivatei numerice prin cinci puncte a funcției  $f(x)$  este :

$$f'(x_0) = \frac{1}{12h} [f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)] \quad (4.33)$$

Această formulă de derivare numerică are eroarea de trunchiere zero până la funcții de gradul patru. Pentru funcții de grad mai mare ca patru, eroarea de trunchiere se determină analog ca la derivatele precedente, obținându-se valoarea :

$$e_T = \frac{24}{5} h^4 f^{IV}(\xi) \quad (4.34)$$

unde  $x \in [x_0 - 2h, x_0 + 2h]$

Dacă numărul de puncte crește pentru calculul unei derivate, atunci eroarea de trunchiere se micșorează.

#### 4.3.1. ALGORITMUL 4.3. DERIVATA PRIN CINCI PUNCTE

```
{Variabile
x0 : punctul în care se calculează derivata, real ;
h : pasul, real ;
der : variabila derivatei, real;
{
  calculează der =  $\frac{1}{12h} [f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)]$ ;
}
Derivata este der ;
}
```

#### 4.3.2. IMPLEMENTAREA ALGORITMULUI 4.3

```
/* Funcția care implementează derivata prin cinci puncte.
Funcția întoarce valoarea derivatei
*/
double Derivata5P( double (*f) (double),
                  double x0,
                  double h)
{
  return(f(x0-2*h)-8*f(x0-h)+8*f(x0+h)-f(x0+2*h))/(12*h);
}
```

### 4.4. CALCULUL DERIVATEI FUNCȚIILOR TABELATE

Se știe că numai funcțiile continue au derivată. În practică întâlnim des cazuri când o funcție continuă este dată sub formă de eșantioane, fără a cunoaște expresia analitică a funcției. În acest caz, se poate calcula derivata funcției date sub formă de tabel cu ajutorul programului prezentat în paragraful 4.4.1. Deoarece eșantioanele pot fi luate la pași diferiți, se utilizează metoda derivării în trei puncte. Pentru ca derivata să fie



cât mai exactă este necesar ca pasul de eșantionare să fie cât mai mic. În caz contrar, derivata se va calcula cu eroare mare.

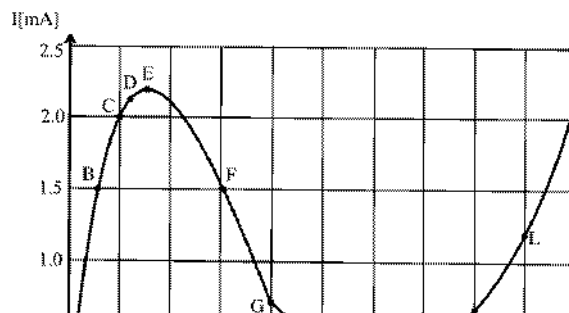
#### 4.4.1. PROGRAM PENTRU CALCULUL DERIVATEI UNEI FUNCȚII EȘANTIONATE

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
{
double Derfcnr( double h1,
                 double h2,
                 double f1,
                 double f0,
                 double f2)
{
return (h1*h1*f2+(h2*h2-h1*h1)*f0-h2*h2*f1)/(h1*h2*(h1+h2));
}
void main(void)
{
double pas1,pas2,fc1,fc0,fc2;
clrscr( );
printf( "Dați pasul1" ); scanf( "%lf",&pas1);
printf( "Dați pasul2" ); scanf( "%lf",&pas2);
printf( "dați funcția în x0-h1" );scanf( "%lf",&fc1);
printf( "Dați funcția în x0" );scanf( "%lf",&f0);
printf( "Dați funcția în x0+h2" );scanf( "%lf",&fc2);
printf( "derivata este %lf\n",Derfcnr(pas1,pas2,fc1,fc0,fc2));
getche( );
/
```

#### 4.5. APLICAȚII

1. Se dă caracteristica  $I-U$  a unei diode tunel în figura 4.2. Se cere rezistența dinamică a diodei în punctele A,B,C,D,E,F,G,H,I,J,K,L.

Pentru calculul rezistenței dinamice a diodei în punctele cerute se utilizează programul dat în paragraful 4.4.1. Acest program îndeplinește condițiile problemei deoarece nu se cunoaște expresia analitică a caracteristicii diodei tunel și se cere rezistența dinamică în diferite puncte ale caracteristicii. Din grafic se determină, la diferite distanțe  $h_1$  și  $h_2$  de punctul dat, valorile funcției. Aceste valori sunt date în tabelul 4.1



Tabelul 4.1.

	$h_1$	$h_2$	$f_1$	$f_0$	$f_2$	Derivata
A	0.2	0.3	0.7	0.1	0.2	0.223333
B	0.2	0.3	2.3	27	40	29.33333
C	0.2	0.2	40	50	77	92.53333
D	0.01	0.01	60	65	68	400.3333
E				70		$\infty$
F	0.2	0.3	130	150	170	86.66667
G	0.2	0.3	190	200	240	83.33333
H	0.25	0.2	220	250	295	38.33333
I	0.1	0.1	250	275	300	500.8922
J				300		$\infty$
K	0.2	0.3	380	400	430	100.4356
L	0.2	0.3	440	450	470	56.34012

2. Se consideră funcția

$$f(x) = 2 \cdot x^2 \cdot \exp(x) + \sin(3 \cdot x - 1.2)$$

și se cere derivata funcției în punctul  $x = 1$ .

Derivata s-a calculat prin toate metodele și s-au obținut următoarele rezultate:

-derivata prin două puncte  $h = 10^{-6}$ ;  $f'(1) = 15.628085$

-derivata prin trei puncte  $h_1 = 10^{-6}$ ;  $h_2 = 1.5 \cdot 10^{-6}$ ;  $f'(1) = 15.628085$

-derivata prin cinci puncte  $h = 10^{-6}$ ;  $f'(1) = 15.628085$ .

# 5

## INTEGRAREA NUMERICĂ\*

Metodele numerice de integrare se clasifică după tipul funcției de integrat și valoarea limitelor de integrare.

I. Prima grupă de metode se referă la funcțiile continue și cu limite finite de integrare. Aceste metode se împart la rândul lor în două subgrupe în funcție de modul de divizare a intervalului de integrare:

a) Metode ce împart intervalul de integrare în subintervale de aceeași lungime, numărul subintervalelor fiind impus de operator. Dintre aceste metode amintim: metoda dreptunghiului, metoda trapezului, metoda lui Simpson și metoda lui Richardson;

b) Metode ce împart intervalul de integrare în așa fel încât eroarea de calcul să fie minimă. Dintre aceste metode amintim și studiem metoda cuadraturii a lui Gauss.

II. A doua grupă de metode se referă la integralele improprii, adică la integrarea funcțiilor cu discontinuități de speța întâi și a doua pe intervale de integrare finite sau integrarea funcțiilor continue pe intervale de integrare infinite.

III. A treia grupă de metode numerice de integrare se ocupă cu integrarea dublă a funcțiilor de două variabile. Amintim în acest sens formulele de cubatură a trapezului și a lui Simpson.

### 5.1. INTEGRAREA FUNCȚIILOR DE O SINGURĂ VARIABILĂ. METODE DE DIVIZAREA CONSTANTĂ

Aceste metode împart intervalul de integrare într-un număr  $n$  de subintervale de lungime egală. Numărul  $n$  are influență asupra preciziei rezultatului integralei astfel: cu cât  $n$  este mai mare, cu atât precizia rezultatului este mai mare, deci cele două mărimi sunt direct proporționale. Numărul  $n$  este ales de proiectant.

#### 5.1.1. METODA DREPTUNGIULUI

Această metodă are erori de calcul mari pentru funcții diferite de o constantă. În cazul când se dorește o evaluare grosieră a unei integrale, se poate aplica această metodă, iar dacă numărul subintervalelor pentru intervalul de integrare crește, eroarea

---

\*)Bibliografie: [6],[7],[15],[22]

de calcul scade. Această creștere a numărului de subintervale se face în detrimentul

timpului de calcul. Se consideră :

$$I = \int_a^b f(x) dx \quad (5.1)$$

unde  $f(x)$  este o funcție continuă pe  $[a, b]$  și  $a, b$  sunt finite.

$I$ - reprezintă aria hașurată punctat din fig. 5.1.

Calculul numeric al acestei integrale se realizează prin divizarea intervalului  $[a, b]$  în  $n$  subintervale de lungime egală cu

$$\Delta x_i = \frac{b-a}{n} = x_{i+1} - x_i = h, i = 0, 1, \dots, n-1. \quad (5.2)$$

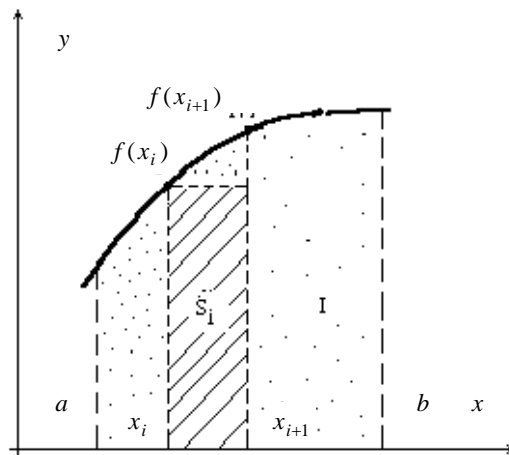
Se calculează aproximativ aria fiecărui dreptunghi

$$s_i = \frac{b-a}{n} \cdot f(x_i) \quad (5.3)$$

și se însumează

$$\sum_{i=0}^{n-1} s_i = h \cdot \sum_{i=0}^{n-1} f(x_i) \quad (5.4)$$

unde  $x_i = a + h \cdot i$



Formula (5.4) reprezintă *formula de integrare a dreptunghiului*.

Pentru funcția constantă metoda dreptunghiului are eroarea de calcul nulă deoarece aria formată de funcție, axa  $Ox$  și verticalele în capetele intervalului este egală cu integrala din funcție pe intervalul dat. Această metodă nu are aplicații în practică datorită erorilor mari, pe care le introduce.

Fig. 5.1. Reprezentarea grafică a integralei  $I$

#### 5.1.1.1. Algoritmul 5.1. Metoda dreptunghiului

{ Variabile

$ls$ : limita stângă a intervalului de integrare, reală;

$ld$ : limita dreaptă a intervalului de integrare, reală;

$n$ : numărul de subintervale, întreg;

$h$ : valoarea lungimii unui subinterval, reală;

$sum$ : valoarea integralei, reală;

{

```

sum=0;
calculează  $h = \frac{ld - ls}{n}$  ;
pentru  $i=0, \dots, n-1$  calculează  $sum = sum + h * f(i)$ 
tipărește valoarea integralei sum;
}

```

### 5.1.1.2. Implementarea algoritmului. Metoda dreptunghiului

```

/* Funcția care implementează metoda de integrare a dreptunghiului.
Funcția întoarce valoarea integralei.
*/
double DreptunghiF(double (*f)(double),
double ls,
double ld,
int nrpas)
{
    int i;
    double suma=0,h;
    h=(ld-ls)/nrpas;
    for(i=0;i<=nrpas-1;i++)suma+=h*f(ls+i*h);
    return suma;
}

```

### 5.1.2. METODA TRAPEZULUI

Fie  $I = \int_a^b f(x)dx$  unde  $f(x)$  este continuă pe  $[a, b]$  și  $a, b$  sunt finite.

Funcția este reprezentată grafic în figura 5.2, iar integrala  $I$  reprezintă aria hașurată și punctată.

Intervalul  $[a, b]$  se împarte în  $n$  subintervale de lungime egală

$$\Delta x_i = x_{i+1} - x_i = \frac{b-a}{n} = h, \quad i=0,1,\dots,n-1, \quad x_0=a \text{ și } x_n=b$$

Aria  $I_i$  este aproximată cu aria trapezului  $(x_i, x_{i+1}, f(x_i), f(x_{i+1}))$

$$I_i \approx \frac{f(x_i) + f(x_{i+1})}{2} h \quad (5.5)$$

$$I \approx \frac{h}{2} \sum_{i=0}^{n-1} [f(x_i) + f(x_{i+1})] = \frac{h}{2} [f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)] \quad (5.6)$$

Această expresie prezintă formula de integrare numerică prin metoda trapezului și are eroare de trunchiere nulă pentru funcții până la gradul întâi inclusiv.

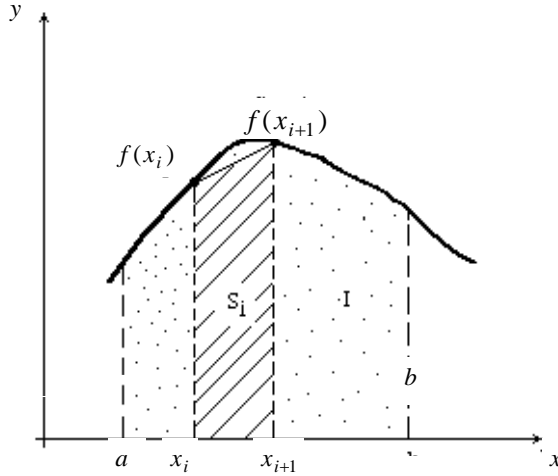


Fig. 5.2. Reprezentarea grafică a metodei de integrare a trapezului

Metoda de integrare a trapezului este superioară din punct de vedere al erorilor de trunchiere față de metoda dreptunghiului, dar ca timp de calcul este comparabilă cu aceasta, pentru un același număr de pași de integrare.

Simplitatea ei o face utilizabilă în numeroase cazuri, precizia ei depinzând de numărul de subintervale alese. Cu cât acest număr este mai mare cu atât precizia este mai bună, dar timpul de calcul al integralei crește. Această integrală stă la baza metodei lui Richardson.

### 5.1.2.1. Eroarea de trunchiere pentru metoda trapezului

Vom calcula eroarea de trunchiere pentru  $I_i$  (5.5). Pentru aceasta vom dezvoltă funcția  $f(x)$  în jurul punctelor  $x_i$  și  $x_{i+1}$ :

$$f(x) = f(x_i) + \frac{(x-x_i)}{1!} \cdot f'(x_i) + \frac{(x-x_i)^2}{2!} \cdot f''(x_i) + \dots \quad (5.7)$$

$$f(x) = f(x_{i+1}) + \frac{(x-x_{i+1})}{1!} \cdot f'(x_{i+1}) + \frac{(x-x_{i+1})^2}{2!} \cdot f''(x_{i+1}) + \dots \quad (5.8)$$

Cu ajutorul celor două dezvoltări (5.7) și (5.8) vom construi o nouă funcție, media acestor funcții care aproximează cel mai bine funcția în intervalul  $(x_i, x_{i+1})$ .

Considerând  $x_{i+1} = x_i + h$  putem scrie noua funcție astfel:

$$\begin{aligned} f(x) = & \frac{f(x_{i+1}) + f(x_i)}{2} + (x-x_i) \cdot \frac{f'(x_i) + f'(x_{i+1})}{2} - h \cdot f'(x_{i+1}) + \\ & + (x-x_i)^2 \cdot \frac{f''(x_i) + f''(x_{i+1})}{4} - h \cdot (x-x_i) \frac{f''(x_{i+1})}{2} + \frac{h^2}{4} \cdot f''(x_{i+1}) + \dots \end{aligned} \quad (5.9)$$

Prin integrarea acestei funcții (5.9) de la  $x_i$  la  $x_{i+1}$  se obține următorul rezultat:

$$\begin{aligned} \int_{x_i}^{x_{i+1}} f(x) dx = & \frac{f(x_{i+1}) + f(x_i)}{2} \cdot h + \frac{f'(x_{i+1}) + f'(x_i)}{4} \cdot h^2 - \frac{f'(x_{i+1})}{2} \cdot h^2 + \\ & + \frac{f''(x_{i+1}) + f''(x_i)}{12} \cdot h^3 - \frac{f''(x_{i+1})}{4} \cdot h^3 + \frac{f''(x_{i+1})}{4} \cdot h^3 + \dots = \end{aligned}$$

$$= \frac{f(x_{i+1}) + f(x_i)}{2} \cdot h - \frac{f'(x_{i+1}) - f'(x_i)}{4} \cdot h^2 - \frac{f''(x_{i+1}) + f''(x_i)}{12} \cdot h^3 + \dots \quad (5.10)$$

Observăm că eroarea de trunchiere este

$$e_{T_i} = -\frac{f(x_{i+1}) - f(x_i)}{4} \cdot h^2 - \frac{f''(x_{i+1}) + f''(x_i)}{12} \cdot h^3 + \dots \quad (5.11)$$

Considerăm că eroarea de trunchiere este de forma:

$$e_{T_i} \approx k \cdot h^2 \cdot [f'(x_{i+1}) - f'(x_i)] \quad (5.12)$$

unde  $k$  se determină astfel ca formulele (5.11) și (5.12) să fie egale. Considerăm o funcție pentru care avem eroare de trunchiere prin metoda trapezului. Aceasta este funcția pătratică  $f(x) = x^2$ . Ținând cont că  $x_{i+1} = x_i + h$  rezultă :

$$\int_{x_i}^{x_{i+1}} x^2 dx = \frac{x^3}{3} \Big|_{x_i}^{x_{i+1}} = \frac{x_{i+1}^3 - x_i^3}{3} = x_i^2 \cdot h + h^2 \cdot x_i + \frac{h^3}{3} \quad (5.13)$$

Aplicând metoda trapezului aceleași funcții avem:

$$\int_{x_i}^{x_{i+1}} x^2 dx = \frac{(x_{i+1}^2 + x_i^2)}{2} \cdot h + e_{T_i} = x_i^2 \cdot h + x_i \cdot h^2 + \frac{h^2}{2} + e_{T_i} \quad (5.14)$$

$$\text{Din relațiile (5.13) și (5.14) rezultă } e_{T_i} = -h^3 / 6 \quad (5.15)$$

Aplicând formula (5.12) pentru  $f(x) = x^2$  rezultă

$$e_{T_i} = k \cdot h^2 \cdot (2x_{i+1} - 2x_i) = 2 \cdot k \cdot h^3 \quad (5.16)$$

$$\text{Din formulele (5.16) și (5.15) rezultă } k = -\frac{1}{12}$$

Eroarea de trunchiere pentru trapezul  $(x_i, x_{i+1}, f(x_i), f(x_{i+1}))$  este

$$e_{T_i} \approx -\frac{1}{12} \cdot h^2 \cdot [f'(x_{i+1}) - f'(x_i)] \quad (5.17)$$

iar pentru întreaga integrală pe intervalul  $[a, b]$  avem cu aproximație eroarea de trunchiere:

$$e_T \approx -(1/2) \cdot h^2 \cdot [f'(b) - f'(a)] \quad (5.18)$$

Această eroare reprezintă aproximativ suma arilor cuprinse între curbă și coarda dusă prin punctele  $(x_i, f(x_i)), (x_{i+1}, f(x_{i+1}))$ ,  $i = 0, 1, \dots, n-1$

### 5.1.2.2. Eroarea de rotunjire pentru metoda trapezului

Formula de calcul a integralei numerice prin metoda trapezului este dată în expresia (5.6). Construim graful de procedură a formulei de calcul considerând că  $f(x_i)$  au erorile relative  $\varepsilon_i$ ,  $i = 0, 1, \dots, n$ , nodurile în care se realizează operațiile  $r_i$ ,  $i = 1, 2, \dots, n+3$ .

$$E_I = (((\dots((\varepsilon_1 \cdot \frac{f(x_1)}{f(x_1) + f(x_2)} + \varepsilon_2 \cdot \frac{f(x_2)}{f(x_1) + f(x_2)} + r_1) \cdot \frac{f(x_1) + f(x_2)}{f(x_1) + f(x_2) + f(x_3)} +$$

$$\begin{aligned}
& + \varepsilon_3 \cdot \frac{f(x_3)}{f(x_1) + f(x_2) + f(x_3)} + r_2) \dots + r_{n-3}) \frac{f(x_1) + f(x_2) + \dots + f(x_{n-2})}{f(x_1) + f(x_2) + \dots + f(x_{n-2}) + f(x_{n-1})} + \\
& + \varepsilon_{n-1} \frac{f(x_{n-1})}{f(x_1) + f(x_2) + \dots + f(x_{n-1})} + r_{n-2} + r_{n-1}) \frac{2f(x_1) + 2f(x_2) + \dots + 2f(x_{n-1})}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \\
& + (\varepsilon_0 \frac{f(x_0)}{f(x_0) + f(x_1)} + \varepsilon_n \frac{f(x_n)}{f(x_0) + f(x_n)} + r_n) \frac{f(x_0) + f(x_n)}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \\
& + r_{n+1} + \varepsilon_h + r_{n+2} = r_{n+1} + \varepsilon_h + r_{n+2} + r_n \frac{f(x_0) + f(x_n)}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \\
& + \varepsilon_0 \frac{f(x_0)}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \varepsilon_n \frac{f(x_n)}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \\
& + (r_{n-2} + r_{n-1}) \frac{2f(x_1) + 2f(x_2) + \dots + 2f(x_{n-1})}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \varepsilon_{n-1} \frac{2f(x_{n-1})}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \\
& + r_{n-3} \frac{2[f(x_1) + f(x_2) + \dots + f(x_{n-2})]}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \varepsilon_{n-2} \frac{2f(x_{n-2})}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \\
& + r_{n-4} \frac{2[f(x_1) + f(x_2) + \dots + f(x_{n-3})]}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \dots + \varepsilon_2 \frac{2f(x_2)}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} + \\
& + \varepsilon_1 \frac{2f(x_1)}{f(x_0) + 2f(x_1) + \dots + 2f(x_{n-1}) + f(x_n)} \tag{5.19}
\end{aligned}$$

Ținând cont că eroarea relativă este  $\varepsilon_I = \frac{e_I}{I}$ , unde  $e_I$  este eroarea absolută, putem calcula eroarea absolută:

$$\begin{aligned}
e_I = & (r_{n+1} + \varepsilon_h + \varepsilon_{n+2}) \cdot I + r_n \cdot \frac{h \cdot [f(x_0) + f(x_n)]}{2} + \varepsilon_0 \cdot \frac{h \cdot f(x_0)}{2} + \\
& + \varepsilon_{n-1} \cdot h \cdot f(x_{n-1}) + (r_{n-1} + r_{n-2}) \cdot h \cdot [f(x_1) + f(x_2) + \dots + f(x_{n-1})] + \\
& + \varepsilon_{n-2} \cdot h \cdot f(x_{n-2}) + r_{n-3} \cdot h \cdot [f(x_1) + f(x_2) + \dots + f(x_{n-2})] + \\
& + \varepsilon_{n-3} \cdot h \cdot f(x_{n-3}) + r_{n-4} \cdot h \cdot [f(x_1) + f(x_2) + \dots + f(x_{n-3})] + \dots + \\
& + \varepsilon_2 \cdot h \cdot f(x_2) + \varepsilon_1 \cdot h \cdot f(x_1) + r_1 \cdot h \cdot [f(x_1) + f(x_2)] \tag{5.20}
\end{aligned}$$



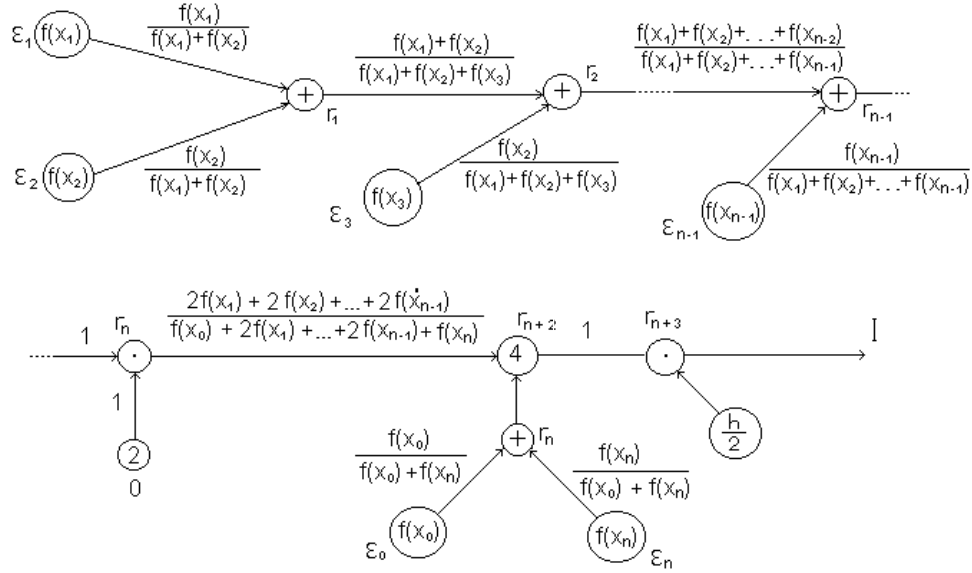


Fig.5.3. Graful de procedură pentru metoda trapezului

Dacă considerăm că toate erorile relative ce intervin în calcul se obțin tot prin rotunjire și sunt mai mici ca  $5 \cdot 10^{-t}$  unde  $t$  este mantisa calculatorului și  $|f(x_i)| < \alpha$  pentru  $i = 0, 1, 2, \dots, n$ , atunci rezultă următoarea margine a erorii absolute:

$$\begin{aligned}
 e_I &\leq 5 \cdot 10^{-t} \cdot [3 \cdot I + h \cdot (\alpha + \frac{\alpha}{2} + \alpha + 2 \cdot (n-1) \cdot \alpha + \alpha + (n-2) \cdot \alpha + \\
 &\quad \alpha + (n-3) \cdot \alpha + \alpha + (n-4) \cdot \alpha + \dots + \alpha + 2 \cdot \alpha + \alpha)] = \\
 &= 5 \cdot 10^{-t} \cdot [3nh\alpha + 5 \cdot h \cdot \alpha / 2 + h \cdot \alpha(n^2 + 3 \cdot n - 8) / 2] = \\
 &= 5 \cdot 10^{-t} \cdot h \cdot \alpha(n^2 + 9 \cdot n - 3) / 2 \\
 e_I &\leq 5 \cdot 10^{-t} \cdot h \cdot \alpha(n^2 + 9 \cdot n - 3) / 2
 \end{aligned} \tag{5.21}$$

Se observă că eroarea de rotunjire depinde proporțional de valoarea lui  $n$  ( numărul de puncte ale diviziunii ). Când  $n$  crește, eroarea de rotunjire se mărește datorită creșterii numărului de operații de calcul.

### 5.1.2.3. Algoritmul 5.2. Metoda trapezului

{ Variabile

ls: limita stângă a intervalului de integrare, reală;

ld: limita dreaptă a intervalului de integrare, reală;

*n*: numărul de subintervale, întreg;  
*h*: valoarea lungimii unui subinterval, reală;  
*sum*: valoarea integralei, reală;

```
{
    calculează  $h = \frac{ld - ls}{n}$  ;
    calculează  $sum = \frac{f(ls) + f(ld)}{2} \cdot h$  ;
    pentru i=1 până la n-1 calculează sum = sum + h * f (ls+i*h);
    tipărește valoarea integralei sum;
}
```

#### 5.1.2.4. Implementarea algoritmului 5.2

```
/* Funcția care implementează metoda de integrare a trapezului.
   Funcția întoarce valoarea integralei
*/
double TrapezF(double (*f)(double),
               double ls,
               double ld,
               int nrpas)
{
    int i;
    double suma, h;
    h = (ld - ls) / nrpas;
    suma = 0.5 * h * (f(ls) + f(ld));
    for(i = 1; i <= nrpas - 1; i++) suma += h * f(ls + i * h);
    return suma;
}
```

#### 5.1.3. METODA LUI RICHARDSON

Această metodă dă o precizie mai bună de calcul a integralei numerice decât metoda trapezului și s-a obținut prin modificarea metodei trapezului.

Se pleacă de la eroarea de trunchiere a metodei trapezului (5.18),  $e_T = Ch^2$  pentru diviziunea  $h = (b-a)/n$ .

Pentru o altă diviziune  $k = (b-a)/m$  se obține eroarea de trunchiere

$$e_T = Ck^2 \quad (5.22)$$

Ca urmare

$$\begin{aligned} I &= I_h + Ch^2 \\ I &= I_k + Ck^2 \end{aligned} \quad (5.23)$$

Prin scădere se calculează  $C = \frac{I_h - I_k}{k^2 - h^2}$  și înlocuind în formula integralei  $I$  rezultă:

$$I = I_h + \frac{I_h - I_k}{\left(\frac{k}{h}\right)^2 - 1} \quad (5.24)$$

expresie ce poartă denumirea de formula lui Richardson și are o precizie mai mare decât metoda trapezului.

### 5.1.3.1. Algoritmul 5.3. Metoda lui Richardson

*{ Variabile*

*ls: limita stângă a intervalului de integrare, reală;*

*ld: limita dreaptă a intervalului de integrare, reală;*

*n: numărul de subintervale, întreg;*

*m: numărul de subintervale, întreg;*

*h: valoarea lungimii unui subinterval cu diviziunea n, reală;*

*k: valoarea lungimii unui subinterval cu diviziunea m, reală;*

*sumh: valoarea integralei cu diviziunea h, reală;*

*sumk: valoarea integralei cu diviziunea k, reală;*

*sum: valoarea integralei, reală;*

*{ calculează  $h = \frac{ld - ls}{n}$  ;*

*calculează  $k = \frac{ld - ls}{m}$  ;*

*calculează  $sumh = \frac{f(ls) + f(ld)}{2} \cdot h$  ;*

*calculează  $sumk = \frac{f(ls) + f(ld)}{2} \cdot k$  ;*

*pentru  $i=1$  până la  $n-1$  calculează  $sumh = sumh$*

*+  $h \cdot f(ls + i \cdot h)$ ;*

*pentru  $i=1$  până la  $m-1$  calculează  $sumk = sumk + k \cdot$*

*$f(ls + i \cdot k)$ ;*

*calculează  $sum = sumh + (sumh - sumk) / ((k/h) \cdot (k/h) - 1)$ ;*

*tipărește valoarea integralei sum;*

*}*

### 5.1.3.2. Implementarea algoritmului 5.3

*/\* Funcția care implementează metoda de integrare a lui Richardson.  
Funcția întoarce valoarea integralei*

```

*/
double RichardsonF(double (*f)(double),
                    double ls,
                    double ld,
                    int nrpash,
                    int nrpask)
{
    int i;
    double suma,sumah,sumak,h,k;
    h=(ld-ls)/nrpash;
    k=(ld-ls)/nrpask;
    sumah=0.5*h*(f(ls)+f(ld));
    sumak=0.5*k*(f(ls)+f(ld));
    for(i=1;i<=nrpash-1;i++)sumah+=h*f(ls+i*h);
    for(i=1;i<=nrpask-1;i++)sumak+=k*f(ls+i*k);
    suma=sumah+(sumah-sumak)/((k*k)/(h*h)-1));
    return suma;
}

```

#### 5.1.4. METODA LUI SIMPSON

Metoda lui Simpson utilizează tot procedeul împărțirii intervalului de integrare în subintervale egale, dar aproximarea este aici făcută cu aria de sub o parabolă, pentru două intervale adiacente. Parabola trece prin trei puncte consecutive ale diviziunii. Formula de calcul a metodei lui Simpson se poate deduce mult mai ușor utilizând formula lui Richardson. Această formulă se utilizează pentru două diviziuni între care avem relațiile:

$$k = 2h, \quad k = \frac{b-a}{m}, \quad h = \frac{b-a}{n} \quad (5.25)$$

Scriem formula metodei trapezului pentru fiecare diviziune în parte:

$$I_h = \frac{h}{2} [f(x_0) + 2f(x_1) + 2f(x_2) + 2f(x_3) + \dots + 2f(x_{n-1}) + f(x_n)] \quad (5.26)$$

$$I_k = h [f(x_0) + 2f(x_2) + 2f(x_4) + 2f(x_6) + \dots + 2f(x_{n-2}) + f(x_n)] \quad (5.27)$$

Aplicăm formula lui Richardson (5.23):

$$\begin{aligned}
 I &= h \left[ \frac{f(x_0)}{2} + f(x_1) + f(x_2) + f(x_3) + f(x_4) + \dots + f(x_{n-1}) + \frac{f(x_n)}{2} \right] + \\
 &+ h \left[ \frac{f(x_0)}{6} + \frac{1}{3}f(x_1) + \frac{1}{3}f(x_2) + \frac{1}{3}f(x_3) + \frac{1}{3}f(x_4) + \dots + \frac{1}{3}f(x_{n-1}) + \frac{f(x_n)}{6} \right] +
 \end{aligned}$$

$$+ h \left[ -\frac{1}{3} f(x_0) - \frac{2}{3} f(x_2) - \frac{2}{3} f(x_4) - \dots - \frac{2}{3} f(x_{n-2}) - \frac{1}{3} f(x_n) \right]$$

$$I = \frac{h}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \dots + 4f(x_{n-1}) + f(x_n)] \quad (5.28)$$

Expresia reprezintă formula de calcul numeric al integralei pentru metoda lui Simpson.

#### 5.1.4.1. Algoritmul 5.4. Metoda lui Simpson

```
{ Variabile
ls: limita stângă a intervalului de integrare, reală;
ld: limita dreaptă a intervalului de integrare, reală;
n: numărul de subintervale, întreg;
h: valoarea lungimii unui subinterval, reală;
suma: valoarea integralei, reală;
{ Variabile
ls: limita stângă a intervalului de integrare, reală;
ld: limita dreaptă a intervalului de integrare, reală;
n: numărul de subintervale, întreg;
h: valoarea lungimii unui subinterval, reală;
suma: valoarea integralei, reală;
{
    calculează  $h = \frac{ld - ls}{n}$  ;
    calculează  $sum = h \cdot \frac{f(l_s) + f(l_d)}{3}$  ;
    pentru i=1 până la n-1
        dacă i = par atunci calculează  $sum = sum + (2/3) * h * f(ls + i * h)$ ;
        altfel calculează  $sum = sum + (4/3) * h * f(ls + i * h)$ ;
    tipărește valoarea integralei sum;
}
```

#### 5.1.4.2. Implementarea algoritmului 5.4

```
/* Funcția care implementează metoda de integrare a lui Simpson.
   Funcția întoarce valoarea integralei. */
double SimpsonF(double (*f)(double),
                double ls,
                double ld,
```

```

        int nrpas)
    {
        int i;
        double suma,h;
        h=(ld-ls)/nrpas;
        suma=h*(f(ls)+f(ld))/3.0;
        for(i=0;i<=nrpas-1;i++)suma+=2*(1+i%2)*h*f(ls+i*h)/3.0;
        return suma }

```

## 5.2. INTEGRAREA FUNCȚIILOR DE O SINGURĂ VARIABILĂ CU METODE CU DIVIZAREA VARIABILĂ

Dintre aceste metode se prezintă metoda cuadraturii lui Gauss. Această metodă determină punctele de divizare ale intervalului de integrare astfel ca eroarea de calcul a integralei să fie minimă.

### 5.2.1. METODA CUADRATURII GAUSSIENE CU DOUĂ PUNCTE

Această metodă reduce orice interval de integrare  $[a,b]$  la intervalul  $[-1,1]$  cu ajutorul formulei de substituție:

$$y = \frac{2x - (b+a)}{b-a} \quad (5.29)$$

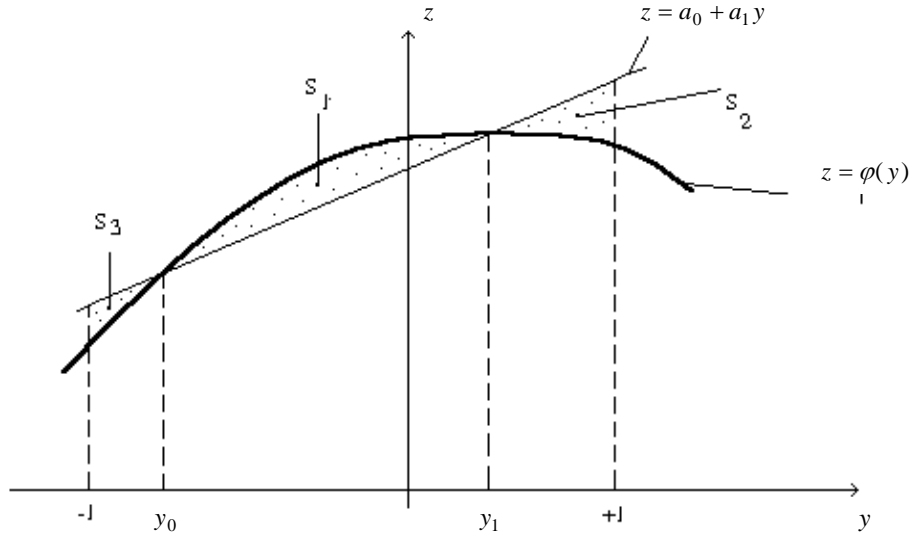
Pentru  $x = a$  rezultă  $y = -1$ , iar pentru  $x = b$  rezultă  $y = 1$ . Substituția este dată de formula:

$$x = \frac{1}{2}(b-a)y + \frac{1}{2}(b+a) \quad (5.30)$$

și 
$$dx = \frac{1}{2}(b-a)dy \quad (5.31)$$

Ca urmare, integrala  $I = \int_a^b f(x)dx$  se transformă în integrala:

$$I = \int_{-1}^{+1} f\left[\frac{1}{2}(b-a)y + \frac{1}{2}(b+a)\right] \frac{1}{2}(b-a)dy = \int_{-1}^{+1} \varphi(y)dy \quad (5.32)$$

Fig.5.4. Reprezentarea grafică a funcției  $z = \varphi(y)$  și a punctelor de divizare

Formula de calcul a integralei o demonstrăm pentru cazul a două puncte în intervalul de integrare, puncte ce le determinăm astfel ca integrala  $I = \int_{-1}^{+1} \varphi(y) dy$  să dea

eroarea zero până la un polinom de gradul trei inclusiv. Alegerea punctelor de divizare a intervalului  $[-1, 1]$  se face astfel ca între ariile  $S_1$ ,  $S_2$ ,  $S_3$  să avem relația

$$S_1 = S_2 + S_3 \quad (5.33)$$

până la o funcție  $\varphi(y)$  de gradul trei.

Integrala prin metoda cuadraturii o calculăm cu formula:  $I = k_0 \varphi(y_0) + k_1 \varphi(y_1)$

unde  $y_0$  și  $y_1$  sunt punctele de divizare a intervalului, iar  $k_0$ ,  $k_1$  sunt niște ponderi, toate necunoscute pe care le vom determina.

Calculăm valoarea exactă a integralei  $I = \int_{-1}^{+1} \varphi(y) dy$  ținând cont de formula (5.33)

$$I = \int_{-1}^{+1} \varphi(y) dy = \int_{-1}^{+1} (a_0 + a_1 y) dy \quad (5.34)$$

unde  $z = a_0 + a_1 \cdot y$  reprezintă ecuația dreptei care trece prin punctele  $(y_0, \varphi(y_0))$  și  $(y_1, \varphi(y_1))$ .

Considerăm funcția  $\varphi(y)$  de gradul trei, pentru care integrala se calculează cu eroarea zero:

$$\varphi(y) = b_0 + b_1 y + b_2 y^2 + b_3 y^3 \quad (5.35)$$

Această funcție o putem scrie și sub forma următoare:

$$\varphi(y) = a_0 + a_1 y + (y - y_0)(y - y_1)(\alpha_0 + \alpha_1 y) \quad (5.36)$$

punând în evidență trecerea curbei  $\varphi(u)$  prin punctele  $(y_0, \varphi(y_0))$  și  $(y_1, \varphi(y_1))$  ale dreptei  $z = a_0 + a_1 y$ .

Egalitatea (5.34) se scrie sub forma:

$$\int_{-1}^{+1} (a_0 + a_1 y) dy = \int_{-1}^{+1} [a_0 + a_1 y + (y - y_0)(y - y_1)(\alpha_0 + \alpha_1 y)] dy \quad (5.37)$$

și, pentru ca această egalitate să fie satisfăcută pentru orice  $\alpha_0$  și  $\alpha_1$ , trebuie ca

$$\int_{-1}^{+1} (y - y_0)(y - y_1) dy = 0 \quad (5.38)$$

$$\text{și} \quad \int_{-1}^{+1} y(y - y_0)(y - y_1) dy = 0 \quad (5.39)$$

Din aceste două ecuații se obține sistemul:

$$\begin{cases} \int_{-1}^{+1} [y^2 - (y_0 + y_1)y + y_0 y_1] dy = 0 \\ \int_{-1}^{+1} [y^3 - (y_0 + y_1)y^2 + y_0 y_1 y] dy = 0 \end{cases} \quad (5.40)$$

sau

$$\begin{cases} y_0 y_1 + \frac{1}{3} = 0 \\ y_0 + y_1 = 0 \end{cases} \quad (5.41)$$

$$\text{cu soluțiile} \quad y_0 = -\frac{1}{\sqrt{3}} \text{ și } y_1 = \frac{1}{\sqrt{3}} \quad (5.42)$$

Pentru calculul ponderilor  $k_0, k_1$  utilizăm egalitatea

$$I = \int_{-1}^{+1} \varphi(y) dy = \int_{-1}^{+1} (a_0 + a_1 y) dy = k_0 \varphi\left(-\frac{1}{\sqrt{3}}\right) + k_1 \varphi\left(\frac{1}{\sqrt{3}}\right) \quad (5.43)$$

$$\int_{-1}^{+1} (a_0 + a_1 y) dy = \left( a_0 y + \frac{a_1}{2} y^2 \right) \Big|_{-1}^{+1} = 2a_0 \quad (5.44)$$

Înlocuind în (5.43) rezultă:

$$k_0 \left( a_0 - \frac{1}{\sqrt{3}} a_1 \right) + k_1 \left( a_0 + \frac{1}{\sqrt{3}} a_1 \right) = 2a_0 \quad (5.45)$$

Prin identificare se obține sistemul:

$$\begin{cases} k_0 + k_1 = 2 \\ k_0 - k_1 = 0 \end{cases} \quad (5.46)$$

$$\text{cu soluțiile:} \quad k_0 = 1, k_1 = 0 \quad (5.47)$$

Formula de calcul a integralei prin metoda cuadraturii gaussiene când utilizăm două puncte de divizare este:

$$I = \int_a^b f(x) dx = \int_{-1}^{+1} \varphi(y) dy = \varphi\left(-\frac{1}{\sqrt{3}}\right) + \varphi\left(\frac{1}{\sqrt{3}}\right) \quad (5.48)$$

$$\text{unde:} \quad \varphi(y) = \frac{1}{2} (b-a) f(y) \quad (5.49)$$



### 5.2.1.1. Eroarea de trunchiere a formulei cuadraturii gaussiene prin două puncte

Integrala dintr-un polinom până la gradul trei are eroarea de trunchiere nulă. Pentru polinoame de grad mai mare ca trei este:

$$e_T = k\psi^{(IV)}(\xi) \quad , \quad -1 < \xi < 1 \quad (5.50)$$

Pentru determinarea lui  $k$  luăm  $\varphi(y) = y^4$

$$\int_{-1}^{+1} \varphi(y) dy = \int_{-1}^{+1} y^4 dy = \left. \frac{y^5}{5} \right|_{-1}^{+1} = \frac{2}{5} \quad (5.51)$$

$$\int_{-1}^{+1} \varphi(y) dy = \int_{-1}^{+1} y^4 dy = \varphi\left(-\frac{1}{\sqrt{3}}\right) + \varphi\left(-\frac{1}{\sqrt{3}}\right) + e_T = \frac{2}{9} + e_T \quad (5.52)$$

Din egalitățile (5.51) și (5.52) rezultă:

$$e_T = \frac{8}{45}$$

Aplicând formula (5.55) pentru  $\varphi(y) = y^4$  unde  $\varphi^{(IV)}(y) = 24$  avem:

$$\frac{8}{45} = k \cdot 24 \quad \text{de unde rezultă} \quad k = \frac{1}{135}$$

Eroarea de trunchiere pentru formula cuadraturii gaussiene prin două puncte este:

$$e_T = \frac{1}{135} \varphi^{(IV)}(\xi) \quad , \quad -1 < \xi < 1 \quad (5.53)$$

### 5.2.2. Metoda cuadraturii gaussiene cu mai multe puncte de divizare

În acest caz

$$\int_a^b f(x) dx = \int_{-1}^{+1} \varphi(u) du = \sum_{i=0}^{n-1} k_i \varphi(y_i) \quad (5.54)$$

utilizând  $n$  puncte de divizare și  $n$  ponderi.

Valorile punctelor de divizare în intervalul  $[-1,1]$  sunt rădăcinile polinoamelor lui Legendre care sunt definite prin relația de recurență:

$$\begin{aligned} P_0(y) &= 1, \quad P_1(y) = y \\ P_n(y) &= \frac{1}{n} [(2n-1)yP_{n-1}(y) - (n-1)P_{n-2}(y)] \end{aligned} \quad (5.55)$$

iar ponderile sunt date de formula:

$$k_i = \frac{2}{(1 - y_i^2) \left[ P_n'(y_i) \right]^2} \quad (5.56)$$

Pentru polinoamele lui Legendre până la gradul 16 sunt date rădăcinile și ponderile în funcția de implementare a metodei.

În cazul a  $n$  puncte de divizare a intervalului de integrare  $[-1,1]$  eroarea de trunchiere este zero pentru toate integralele polinoamelor de grad mai mic decât  $2n-1$ , inclusiv  $2n-1$ .

#### 5.2.2.1. Eroarea de trunchiere pentru formula cuadraturii gaussiene cu mai multe puncte de divizare

Considerăm un polinom de gradul  $2n$  pentru care integrala gaussiană are eroarea de trunchiere:

$$e_T = k \varphi^{(2n)}(\xi), \quad -1 < \xi < 1 \quad (5.57)$$

Calculăm integrala pentru  $\varphi(y) = y^{2n}$

$$\int_{-1}^{+1} \varphi(y) dy = \int_{-1}^{+1} y^{2n} dy = \left. \frac{y^{2n+1}}{(2n+1)} \right|_{-1}^{+1} = \frac{2}{2n+1} \quad (5.58)$$

$$\int_{-1}^{+1} \varphi(y) dy = \int_{-1}^{+1} y^{2n} dy = \sum_{i=0}^{n-1} k_i y_i^{2n} + e_T \quad (5.59)$$

Din egalarea relațiilor (5.58) și (5.59) rezultă:

$$e_T = \frac{2}{2n+1} - \sum_{i=0}^{n-1} k_i y_i^{2n} \quad (5.60)$$

Aplicând formula (5.57), unde  $\varphi^{(2n)}(\xi) = (2n)!$  și egalând-o cu (5.60) rezultă:

$$k = \frac{1}{(2n)!} \left( \frac{2}{2n+1} - \sum_{i=0}^{n-1} k_i u_i^{2n} \right) \quad (5.61)$$

iar eroarea de trunchiere

$$e_T = \frac{\varphi^{(2n)}(\xi)}{(2n)!} \left( \frac{2}{2n+1} - \sum_{i=0}^{n-1} k_i u_i^{2n} \right) \quad (5.62)$$

#### 5.2.2.2. Algoritmul 5.5. Metoda cuadraturii a lui Gauss

{ Variabile

ls: limita stângă a intervalului de integrare, reală;

ld: limita dreaptă a intervalului de integrare, reală;

n: gradul polinomului lui Legendre;

A[n, i]: matricea ponderilor, reali;

```

    U[n, i]:matricea soluțiilor polinoamelor lui Legendre, reale;
    sum: valoarea integralei, reală;
{ Construieste matricea A[n, i];
  Construieste matricea Y[n, i];
  suma=0;
  pentru i=1 până la n calculează
    sum=sum+A[n,i]*(1/2)*(ld-ls)*f((1/2)*(ld-
      ls)*Y[n,i]+(1/2)*(ls+ld));
  tipărește valoarea integralei sum;
}
}

```

### 5.2.2.3. Implementarea algoritmului 5.5

```

/* Funcția care implementează metoda cuadraturii lui Gauss
   pentru integrarea funcțiilor reale de variabilă reală
*/
double CuadraturaGauss(double (*f)(double),
                      double ls,
                      double ld,
                      int ord_pol)
{
    static double A[17][17]=

    {

/* n=0 */
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

/* n=1 */
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

/* n=2 */
{1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

/* n=3 */
{ 5.0/9.0,8.0/9.0,5.0/9.0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

/* n=4 */
{0.347854854137454      0.652145145862564
 0.652145145862564      0.347854854137454
 0                      0
 0                      0
 0                      0
 0                      0

```

0	0
0	0}
/* n=5 */	
{0.236926885056189,	0.478628670499366
0.568888888888889,	0.478628670499366,
0.236926885056189	0,
0,	0,
0,	0,
0,	0,
0,	0,
0,	0}
/* n=6 */	
{ 0.17132449237917,	0.360761573048139,
0.467913934572691,	0.467913934572691,
0.360761573048139,	0.17132449237917,
0,	0,
0,	0,
0,	0,
0,	0,
0,	0}
/* n=7 */	
{0.12948496616887,	0.279705391489277,
0.381830050505119,	0.417959183673469,
0.381830050505119,	0.279705391489277,
0.12948496616887,	0,
0,	0,
0,	0,
0,	0,
0,	0}
/* n=8 */	
{0.101228536290376,	0.222381034453374,
0.313706645877887,	0.362684783378362,
0.362684783378362,	0.313706645877887,
0.222381034453374,	0.101228536290376,
0,	0,
0,	0,
0,	0,
0,	0},
/* n=9 */	
{ 0.08127438361574,	0.180648160694857
0.260610696402935	0.312347077040003,
0.33023935500126	0.312347077040003,
0.260610696402935,	0.180648160694857,
0.08127438361574,	0,
0,	0,

0,	0,
0,	0},
/* n=10 */	
{0.066671344308688,	0.14945134915058
0.219086362515982,	0.269266719309996,
0.295524224714753,	0.295524224714753,
0.269266719309996,	0.219086362515982,
0.14945134915058,	0.066671344308688,
0,	0,
0,	0,
0,	0},
/* n=11 */	
{0.055668567116,	0.125580369465,
0.186290210928,	0.233193764592,
0.26280454451,	0.272925086778,
0.26280454451,	0.233193764592,
0.186290210928,	0.125580369465,
0.055668567116,	0,
0,	0,
0,	0},
/* n=12 */	
{0.047175336387,	0.106939325995,
0.160078328542,	0.203167426723,
0.233492536538,	0.249147045813,
0.249147045813,	0.233492536538,
0.203167426723,	0.160078328542,
0.106939325995,	0.047175336387,
0,	0,
0,	0},
/* n=13 */	
{0.040484004765,	0.092121499838,
0.13887351022,	0.178145980762,
0.207816047537,	0.226283180263
0.232551553231,	0.226283180263,
0.207816047537	0.178145980762,
0.13887351022,	0.092121499838
0.040484004765,	0,
0,	0},
0,	
/* n=14 */	
{0.035119460332,	0.08015808716,
0.121518570688,	0.157203167158,
0.185538397478,	0.205198463721,
0.215263853463,	0.215263853463,
0.205198463721,	0.185538397478,

```
static double U[17][17]=
```

{	
/* n=0 */	
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},	
/* n=1 */	
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},	
/* n=2 */	
{0.5773502692,	-0.5773502692,
0,	0,
0,	0,
0,	0,
0,	0,
0,	0,
0,	0,
0,	0},
/* n=3 */	
{ 0.7745966692,	0,
-0.7745966692	0,
0,	0,
0,	0,
0,	0,
0,	0,
0,	0,

0,	0},
/* n=4 */	
{0.861136311594053,	0.339981043584856,
-0.339981043584856,	-0.861136311594053,
0,	0,
0,	0,
0,	0,
0,	0,
0,	0,
0,	0}
/* n=5 */	
{0.906179845938664,	0.538469310105683,
0,	-0.538469310105683,
-0.906179845938664,	0,
0,	0,
0,	0,
0,	0,
0,	0,
0,	0},
/* n=6 */	
{0.932469514203152,	0.661209386466265,
0.238619186083197,	-0.238619186083197,
-0.661209386466265,	-0.932469514203152,
0,	0,
0,	0,
0,	0,
0,	0,
0,	0},
/* n=7 */	
{0.949107912342759,	0.741531185599394,
0.405845151377397,	0,
-0.405845151377397,	-0.741531185599394,
-0.949107912342759,	0,
0,	0,
0,	0,
0,	0,
0,	0},
/* n=8 */	
{0.96269856497336,	0.796666477413627,
0.525532409916329,	0.18343464249565,
-0.18343464249565,	-0.525532409916329,
-0.796666477413627,	-0.96269856497336,
0,	0,
0,	0,
0,	0,

0,	0},
/* n=9 */	
{0.968160239507626,	0.836031107326636,
0.613371432700591,	0.324253423403809,
0,	-0.324253423403809,
-0.613371432700591,	-0.836031107326636,
-0.968160239507626,	0,
0,	0,
0,	0,
0,	0},
/* n=10 */	
{0.973906528517172,	0.865063366688985,
0.679409568299024,	0.433395394129247,
0.148874338981631,	-0.148874338981631,
-0.433395394129247,	-0.679409568299024,
-0.865063366688985,	-0.973906528517172,
0,	0,
0,	0,
0,	0},
/* n=11 */	
{0.978228658146,	0.887062599768,
0.730152005574,	0.519096129207,
0.269543155952,	0,
-0.269543155952,	-0.519096129207,
-0.730152005574,	-0.887062599768,
-0.978228658146,	0,
0,	0,
0,	0},
/* n=12 */	
{0.981560634247,	0.90411725637,
0.769902671494,	0.587317954287,
0.367831498998,	0.125233408511,
-0.125233408511,	-0.367831498998,
-0.587317954287,	-0.769902671494,
-0.90411725637,	-0.981560634247,
0,	0,
0,	0},
/* n=13 */	
{0.984183054719,	0.917598399223,
0.801578090733,	0.64234933944,
0.448492751036,	0.230458315955,
0,	-0.230458315955,
-0.448492751036,	-0.64234933944,
-0.801578090733,	-0.917598399223,
-0.984183054719,	0,
0,	0},



```

/* n=14 */
{0.986283808697,          0.928434883664,
 0.82720131507,           0.687292904812,
 0.515248636358,         0.319112368928,
 0.108054948707,         -0.108054948707,
 -0.319112368928,        -0.515248636358,
 -0.687292904812,        -0.82720131507,
 -0.928434883664,        -0.986283808697,
 0,                        0},
/* n=15 */
{0.98799251802,          0.937273392401,
 0.84820658341,          0.72441773136,
 0.570972172609,         0.394151347078,
 0.201194093997,         0,
 -0.201194093997,        -0.394151347078,
 -0.570972172609,        -0.72441773136,
 -0.84820658341,         -0.937273392401,
 -0.98799251802,         0},
/* n=16 */
{0.989400934992,          0.944575023075,
 0.865631202388,          0.755404408355,
 0.617876244403,         0.458016777657,
 0.281603550779,         0.095012509838,
 -0.095012509838,        -0.281603550779,
 -0.458016777657,        -0.617876244403,
 -0.755404408355,        -0.865631202388,
 -0.944575023075,        -0.989400934992}};

int i;
double suma=0;
for(i=1;i<=ord_pol;i++)
suma+=0.5*(ld-ls)*A[ord_pol][i-1]*f(0.5*(ld-ls)*
U[ord_pol][i-1]+0.5*(ls+ld));
return suma;
}

```

### 5.3. COMPARAREA METODELOR DE INTEGRARE NUMERICĂ A FUNCȚIILOR DE O SINGURĂ VARIABILĂ

Dintre toate metodele de integrare numerică, metoda cuadraturii gaussiene este cea mai precisă, realizând aceeași precizie ca și metoda lui Simpson cu un număr dublu de puncte de divizare și ca metoda trapezului cu un număr de patru ori mai mare de puncte de divizare.

Pentru aceeași precizie de calcul a integralei numerice, eficiența crește sau timpul de calcul al calculatorului scade după cum utilizăm în ordine metoda trapezului, metoda lui Simpson și metoda cuadraturii gaussiene.

## 5.4. CALCULUL NUMERIC AL INTEGRALELOR IMPROPRII

**Definiția 5.1:** Se numește *integrală improprie*, integrala pentru care cel puțin una dintre limitele de integrare este infinită și funcția este continuă pe intervalul de integrare sau funcția are puncte de discontinuitate de speța întâi sau a doua și limitele de integrare sunt finite.

Integralele improprii de forma

$$\int_{-\infty}^{+\infty} f(x)dx, \quad \int_a^{+\infty} f(x)dx, \quad \int_{-\infty}^a f(x)dx \quad (5.63)$$

pot fi aduse la forma

$$\int_a^{+\infty} f(x)dx \quad (5.64)$$

Ca atare, se va studia integrala improprie de această formă.

Dacă funcția de integrat definită pe intervalul  $[a, \infty]$  este integrabilă pe acest interval și există limita:

$$\lim_{A \rightarrow \infty} \int_a^A f(x)dx = k \quad (5.65)$$

atunci

$$\int_a^{+\infty} f(x)dx = k \quad (5.66)$$

În acest caz integrala improprie este convergentă. Când limita nu există sau este infinită, atunci integrala improprie este divergentă. Valoarea lui  $A$  se poate lua suficient de mare pentru ca

$$\left| \int_A^{\infty} f(x)dx \right| < \varepsilon \quad (5.67)$$

unde  $\varepsilon$  este o constantă pozitivă suficient de mică. În acest caz integrala improprie

$$\int_a^{\infty} f(x)dx \equiv \int_a^A f(x)dx \quad (5.68)$$

integrală ce poate fi calculată cu una dintre metodele studiate în paragraful 5.1.

Funcțiile care pe intervalul de integrare  $[a, b]$  au un punct de discontinuitate de speța întâi  $c \in [a, b]$  au proprietatea că:

$$f(c-0) = \lim_{\substack{x \rightarrow c \\ x < c}} f(x) \quad \text{și} \quad f(c+0) = \lim_{\substack{x \rightarrow c \\ x > c}} f(x) \quad (5.69)$$

iar  $f(c) \neq f(c-0)$  sau  $f(c) = f(c-0)$  și  $f(c) \neq f(c+0)$  sau  $f(c) = f(c+0)$ .

În cazul acestor funcții:

$$\int_a^b f(x) dx = \int_a^c f_1(x) dx + \int_c^b f_2(x) dx \quad (5.70)$$

unde

$$f_1(x) = \begin{cases} f(x) & \text{pentru } a < x < c \\ f(c-0) & \text{pentru } x = c \end{cases}$$

$$f_2(x) = \begin{cases} f(x) & \text{pentru } c < x < b \\ f(c+0) & \text{pentru } x = c \end{cases}$$

Dacă integrala (5.70) există, spunem că integrala improprie este convergentă și valoarea ei poate fi calculată cu ajutorul unei metode studiată în paragraful 5.1.

Funcția  $f(x)$  are un punct de discontinuitate de speța a doua  $c \in [a, b]$  dacă cel puțin una din limitele (5.69) are valoarea infinită. În acest caz

$$\int_a^b f(x) dx = \int_a^{c-\varepsilon} f(x) dx + \int_{c+\varepsilon}^b f(x) dx \quad (5.71)$$

unde  $\varepsilon$  poate fi luat suficient de mic astfel ca  $\left| \int_{c-\varepsilon}^{c+\varepsilon} f(x) dx \right| < \varepsilon_1$ ,  $\varepsilon_1 > 0$

de valoare foarte mică, care reprezintă și eroarea de calcul a integralei. Integrala (5.71) poate fi rezolvată cu una dintre metodele studiate în paragraful 5.1. (metoda dreptunghiului, metoda trapezului, metoda lui Richardson, metoda lui Simpson sau metoda cuadraturii).

## 5.5. CALCULUL NUMERIC AL INTEGRALELOR DUBLE

Pentru simplitate vom considera domeniul de integrare al funcției de două variabile un dreptunghi (Fig. 5.5)

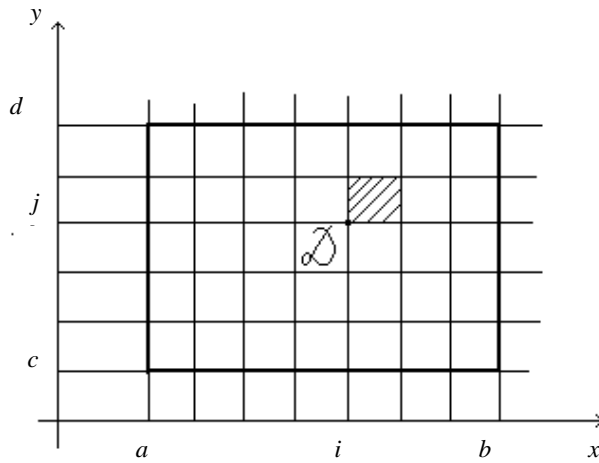


Fig. 5.5. Reprezentarea grafică a dreptunghiului de integrare

$$\iint_D f(x,y)dx = \int_a^b \int_c^d f(x,y)dx dy$$

(5.72)

reprezintă integrala dublă din funcția de două variabile  $f(x, y)$ . Pentru calculul valorii acestei integrale vom utiliza formula de cubatură a trapezului sau formula de cubatură a lui Simpson care sunt prezentate în continuare.

### 5.5.1. FORMULA DE CUBATURĂ A TRAPEZULUI

Se împart în subintervale de lungimi egale intervalele  $[a, b]$  și  $[c, d]$

$$h = \frac{b-a}{n}, \quad k = \frac{d-c}{m} \quad (5.73)$$

și se consideră dreptunghiul cu vârfurile  $[x_i, y_i], [x_{i+1}, y_i], [x_{i+1}, y_{i+1}], [x_i, y_{i+1}]$ , unde  $x_i = a + i \cdot h$ ,  $y_j = c + j \cdot k$ .

Pentru dreptunghiul dat care conține vârful  $[x_i, y_i]$  se calculează integrala  $I_{ij}$  aplicând formula trapezului.

$$\begin{aligned} I_{ij} &= \int_{x_i}^{x_{i+1}} dx \int_{y_j}^{y_{j+1}} f(x,y)dx dy \approx \int_{x_i}^{x_{i+1}} \left\{ \frac{k}{2} [f(x, y_j) + f(x, y_{j+1})] \right\} dx = \\ &= \frac{k}{2} \left[ \int_{x_i}^{x_{i+1}} f(x, y_j) dx + \int_{x_i}^{x_{i+1}} f(x, y_{j+1}) dx \right] = \\ &= \frac{k \cdot h}{4} [f(x_i, y_j) + f(x_i, y_{j+1}) + f(x_{i+1}, y_j) + f(x_{i+1}, y_{j+1})] \end{aligned} \quad (5.74)$$

Integrala pe întreg dreptunghiul  $[a, b, c, d]$  este:

$$I \cong \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} I_{ij} = \frac{kh}{4} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} [f(x_i, y_j) + f(x_i, y_{j+1}) + f(x_{i+1}, y_j) + f(x_{i+1}, y_{j+1})] \quad (5.75)$$

expresie cunoscută sub numele de *formula de cubatură a trapezului*.

### 5.5.1.1. Algoritmul 5.6. Metoda cubaturii trapezului

```

{Variabile
a: limita stângă a intervalului de integrare pe axa Ox, reală;
b: limita dreaptă a intervalului de integrare pe axa Ox, reală;
c: limita stângă a intervalului de integrare pe axa Oy, reală;
d: limita dreaptă a intervalului de integrare pe axa Oy, reală;
n: numărul de subintervale pe axa Ox, întreg;
m: numărul de subintervale pe axa Oy, întreg;
h: valoarea lungimii unui subinterval cu diviziunea n, reală;
k: valoarea lungimii unui subinterval cu diviziunea m, reală;
sum: valoarea integralei, reală;
{
    calculează  $h = \frac{b-a}{n}$  ;

    calculează  $k = \frac{d-c}{m}$  ;
    sum=0;
    pentru i=1 până la n-1
    pentru j=1 până la m-1
    calculează

    sum=sum+((h*k)/4)*(f(a+i*h,c+j*k)+f(a+i*h,c+(j+1)*k) +
    +f(a+(i+1)h,c+j*k)+f(a+(i+1)h,(j+1)*k))
    tipărește valoarea integralei sum;
}
}_

```

### 5.5.1.2. Implementarea algoritmului 5.6

```

/* Funcția care implementează metoda de cubatură a trapezului */
/* Funcția întoarce integrala unei funcții de două variabile */
double CubaturaTrapez( double (*f)(double,double),
    double sx,
    double dx,
    double sy,
    double dy,
    int nx,
    int ny)
{double suma=0,h,k;
  int i,j;
  h=(dx-sx)/nx;

```

```

k=(dy-sy)/ny;
for(i=0;i<=nx-1;i++)
for(j=0;j<=ny-1;j++)
suma+=0.25*h*k*(f(sx+i*h,sy+j*k)+f(sx+i*h,sy+(j+1)*k)+
f(sx+(i+1)*h,sy+j*k)+f(sx+(i+1)*h,sy+(j+1)*k));
return suma;
}

```

### 5.5.2. FORMULA LUI SIMPSON DE CUBATURĂ

Pentru același dreptunghi  $[a, b, c, d]$  reprezentat în fig. 5.5 vom aplica formula lui Simpson de integrare. Vom considera dreptunghiul de integrare cu vârfurile  $[x_i, y_i], [x_{i+1}, y_i], [x_{i+1}, y_{i+1}], [x_i, y_{i+1}]$  și cu punctul central  $[x_i, y_i]$

$$\begin{aligned}
 I_{ij} &\cong \int_{x_{i-1}}^{x_{i+1}} \int_{y_{j-1}}^{y_{j+1}} f(x, y) dx dy = \int_{x_{i-1}}^{x_{i+1}} \frac{k}{3} [f(x, y_{j-1}) + 4f(x, y_j) + f(x, y_{j+1})] dx = \\
 &= \frac{kh}{9} \{ f(x_{i-1}, y_{j-1}) + f(x_{i+1}, y_{j-1}) + f(x_{i-1}, y_{j+1}) + f(x_{i+1}, y_{j+1}) + \\
 &\quad + 4[f(x_i, y_{j+1}) + f(x_i, y_{j-1}) + f(x_{i-1}, y_j) + f(x_{i+1}, y_j)] \} + 16f(x_i, y_i)
 \end{aligned} \tag{5.76}$$

Integrala pe întreg dreptunghiul  $[a, b, c, d]$  este dată de formula de cubatură a lui Simpson:

$$\begin{aligned}
 I &= \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} I_{ij} = \frac{kh}{9} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} \{ f(x_{i-1}, y_{j-1}) + f(x_{i+1}, y_{j-1}) + f(x_{i-1}, y_{j+1}) + f(x_{i+1}, y_{j+1}) + \\
 &\quad + 4[f(x_i, y_{j+1}) + f(x_i, y_{j-1}) + f(x_{i-1}, y_j) + f(x_{i+1}, y_j)] + 16f(x_i, y_j) \} \tag{5.77}
 \end{aligned}$$

#### 5.5.2.1. Algoritmul 5.7. Metoda lui Simpson de cubatură

{ Variabile

*a*: limita stângă a intervalului de integrare pe axa Ox, reală;  
*b*: limita dreaptă a intervalului de integrare pe axa Ox, reală;  
*c*: limita stângă a intervalului de integrare pe axa Oy, reală;  
*d*: limita dreaptă a intervalului de integrare pe axa Oy, reală;  
*n*: numărul de subintervale pe axa Ox, întreg;  
*m*: numărul de subintervale pe axa Oy, întreg;  
*h*: valoarea lungimii unui subinterval cu diviziunea n, reală;  
*k*: valoarea lungimii unui subinterval cu diviziunea m, reală;  
*sum*: valoarea integralei, reală;

```

{ calculează  $h = \frac{b-a}{n}$  ;
  calculează  $k = \frac{d-c}{m}$  ;
  sum=0; i=1;
  repetă
    j=1;
    repetă
      sum=sum+((h*k)/9)*(f(a+(i-1)*h,c+(j-1)*k)+f(a+(i+1)*h,c+(j-1)*k) +
        +f(a+(i-1)*h,c+(j+1)*k)+4*(f(a+i*h,(j+1)*k))+
        +f(a+i*h,c+(j-1)*k)+f(a+(i-1)*h,c+j*k)+f(a+(i+1)*h,j*k)+
        + 16*f(a+i*h,c+j*k))
      j=j+2;
    până când j>m-1;
    i=i+2;
  până când i>n-1;

  tipărește valoarea integralei sum;
}
}_

```

### 5.5.2.2. Implementarea algoritmului 5.7

*/\* Funcția care implementează metoda lui Simpson de cubatură.  
 Funcția întoarce valoarea integralei duble.\*/*

```

double CubaturaSimpson( double (*f)(double,double),
                        double sx,
                        double dx,
                        double sy,
                        double dy,
                        int nx,
                        int ny)
{
  double suma=0,h,k;
  int i,j;
  h=(dx-sx)/nx;
  k=(dy-sy)/ny;
  for(i=1;i<=nx-1;i+=2)
  for(j=1;j<=ny-1;j+=2)
    suma+=h*k*( f(sx+(i-1)*h,sy+(j-1)*k)+f(sx+(i+1)*h,sy+(j-1)*k)+
      f(sx+(i-1)*h,sy+(j+1)*k)+f(sx+(i+1)*h,sy+(j+1)*k)+

```

```

4*f(sx+i*h,sy+(j+1)*k)+4*f(sx+i*h,sy+(j-1)*k)+
4*f(sx+(i-1)*h,sy+j*k)+4*f(sx+(i+1)*h,sy+j*k)+
16*f(sx+i*h,sy+j*k))/9;
return suma;
}

```

## 5.6. APLICAȚII

1. Se dă funcția  $f(x) = \frac{x^3}{1 + \cos(1+x)} \cdot e^{x^2} \cdot (1 + \sin x^2)$

pentru care se cere integrala de la 0 la 3.

Valoarea integralei este dată în tabelul 5.1

Tabelul 5.1.

Metoda	Numărul de subintervale	Valoarea integralei
Dreptunghiului	10000	1394.642843
Trapezului	10000	1395.704297
Richardson	5000 și 10000	1395.703984
Simpson	10000	1395.703984
Cuadratura Gauss	Grad pol Legendre n=15	1395.7031

2. Se consideră funcția de două variabile:

$$f(x,y) = \frac{x^2 + y^2}{1 + 2 \cdot x \cdot y} \cdot \exp(1+x) \cdot \sin(x+y+2)$$

Se cere valoarea integralei din funcția dată pe domeniul  $x \in [0,2]$ ;  $y \in [0,2]$ .

Valoarea integralei este dată în tabelul 5.2.

Tabelul 5.2

Metoda	Nr. Pct. Pe Ox	Nr. Pct. Pe Oy	Valoarea integralei
Cubatura trapezului	100	100	-24.730047
Cubatura Simpson	100	100	-24.733155



# 6

## INTERPOLAREA\*

Interpolarea este una dintre metodele de aproximare a funcțiilor. Considerăm dată o funcție sub formă de tabel. Cunoscând valorile funcției în anumite puncte pentru care funcția este definită, se pune problema cunoașterii valorilor funcției în alte puncte ale domeniului de definiție, în care funcția este necunoscută. Această problemă se pune în cazul reprezentării grafice a funcției tabelate sau în cazul necesității cunoașterii valorii funcției într-un punct în care nu este cunoscută.

În cazul cunoașterii expresiei analitice a funcției  $y = f(x)$ , valorile necunoscute ale funcției se calculează prin introducerea argumentului corespunzător în funcție. Dacă nu se cunoaște expresia analitică a funcției, se aproximează funcția cu o altă funcție, care o aproximează cel mai bine pe prima între punctele unde dorim valoarea funcției. Aproximarea funcției date cu o funcție liniară sau cu o funcție polinomială de un anumit grad poate da valori foarte apropiate de valorile funcției.

### 6.1. INTERPOLAREA POLINOMIALĂ

Se consideră funcția dată prin tabelul 6.1:

Tabelul 6.1.

x	$x_0$	$x_1$	.	$x_k$	.	$x_n$
y	$y_0$	$y_1$	.	$y_k$	.	$y_n$

Cunoaștem valoarea funcției în  $n+1$  puncte. Prin  $n+1$  puncte se poate duce un polinom de gradul  $n$ , unic determinat.

Fie polinomul:

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

Conform tabelului 6.1 avem sistemul:

$$\begin{cases} a_n x_0^n + a_{n-1} x_0^{n-1} + \dots + a_1 x_0 + a_0 = y_0 \\ a_n x_1^n + a_{n-1} x_1^{n-1} + \dots + a_1 x_1 + a_0 = y_1 \\ \dots \dots \dots \\ a_n x_n^n + a_{n-1} x_n^{n-1} + \dots + a_1 x_n + a_0 = y_n \end{cases}$$

\*) *Bibliografie:* [1],[6],[7],[15],[21],[22]

Acest sistem are  $n+1$  ecuații cu  $n+1$  necunoscute,  $a_0, a_1, a_2, \dots, a_{n-2}, a_{n-1}, a_n$ .

Considerăm sistemul omogen:

$$\left\{ \begin{array}{l} a_n x_0^n + a_{n-1} x_0^{n-1} + \cdots + a_1 x_0 + a_0 = 0 \\ a_n x_1^n + a_{n-1} x_1^{n-1} + \cdots + a_1 x_1 + a_0 = 0 \\ \hline a_n x_n^n + a_{n-1} x_n^{n-1} + \cdots + a_1 x_n + a_0 = 0 \end{array} \right. \quad (6.3)$$

Eroarea de trunchiere este dată de diferența dintre funcția  $f(x)$  de interpolat și polinomul de interpolare al lui Lagrange.

$$e_T = f(x) - \sum_{i=0}^n y_i \prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j} = F(x) \quad (6.9)$$

Construim funcția

$$G(x) = (x - x_0)(x - x_1) \dots (x - x_n) \quad (6.10)$$

Cu funcțiile  $F(x)$ , (6.9) și  $G(x)$ , (6.10) formăm următoarea funcție :

$$H(t) = F(x)G(t) - F(t)G(x) \quad (6.11)$$

care are proprietățile :

1.  $H(x_j) = 0$  pentru  $j = 0, \dots, n$ , deoarece înlocuind  $x = x_j$  în formula (6.9) rezultă  $F(x_j) = 0$ ;  $j = 0, 2, \dots, n$  și  $G(x_j) = 0$ ;  $j = 0, 2, \dots, n$ ;

2.  $H(x) = 0$

Pe baza teoremei valorii medii rezultă că există  $n+2$  puncte  $\xi_0, \xi_1, \xi_2, \dots, \xi_{n+1}$  pentru care derivata  $H'(t) = 0$ ,  $H'(\xi_0), H'(\xi_1), H'(\xi_2), \dots, H'(\xi_n), H'(\xi_{n+1})$   $\xi_i \in [x_0, x_n]$ ,  $i = 0, \dots, n+1$ .

Continuând aplicarea teoremei valorii medii, se ajunge în final la egalitatea :

$$H^{(n+1)}(\xi) = 0 \quad \text{unde} \quad x_0 < \xi < x_n$$

Din (6.11) rezultă:

$$\begin{aligned} H^{(n+1)}(t) &= F(x)G^{(n+1)}(t) - F^{(n+1)}(t)G(x) = (n+1)!F(x) - f^{(n+1)}(t)G(x) \\ H^{(n+1)}(\xi) &= 0 = (n+1)!F(x) - f^{(n+1)}(\xi)G(x) \end{aligned} \quad (6.12)$$

sau

$$e_T = F(x) = -\frac{f^{(n+1)}(\xi)}{(n+1)!}G(x) \quad x_0 < \xi < x_n \quad (6.13)$$

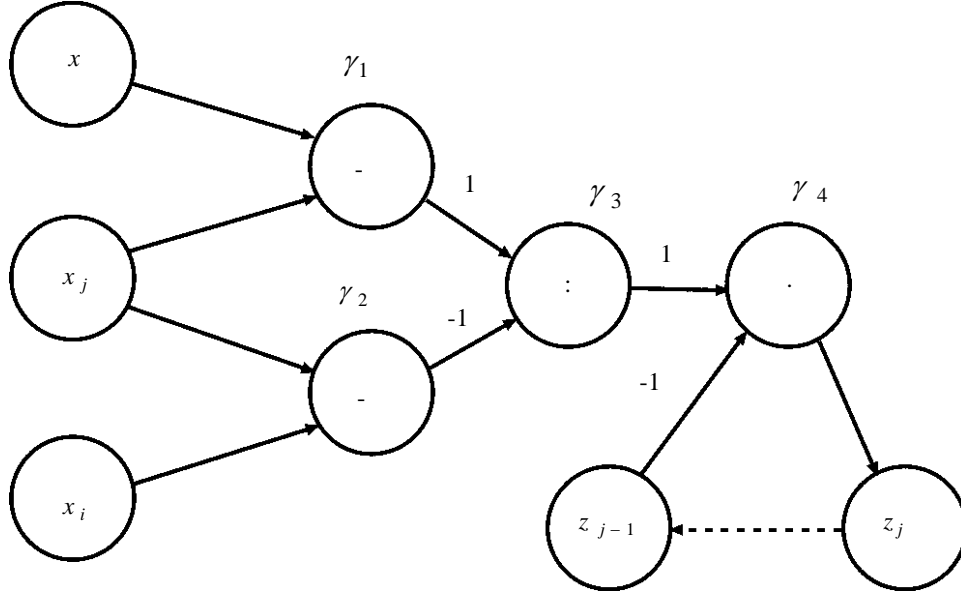
formulă ce reprezintă eroarea de trunchiere pentru interpolarea lagranjiană.

### 6.1.2 EROAREA DE ROTUNJIRE ÎN INTERPOLAREA LAGRANJIANĂ

Considerăm polinomul lui Lagrange dat de relația (6.8) pe care o scriem sub forma:

$$P(x) = \sum_{i=0}^{n-1} y_i z_i \quad \text{unde} \quad z_i = \prod_{j=0, j \neq i}^{n-1} \frac{x - x_j}{x_i - x_j} \quad \text{și} \quad p_i = \sum_{k=0}^i y_k z_k, \quad P(x) = p_n \quad (6.14)$$

Construim graficul de procedură pentru calculul erorii de rotunjire a expresiei (6.14) care reprezintă produsul de ordinul  $i$  din expresia polinomului lui Lagrange. În expresia produsului există operații de scădere, împărțire și înmulțire. Pentru fiecare nod al grafului unde se realizează o operație aritmetică se dă eroarea de rotunjire de tip simetric. Pentru calculul erorii de rotunjire totale a expresiei polinomului lui Lagrange se realizează un graf de procedură și pentru expresia sumei din (6.14).

Fig.6.1. Graful de procedură a expresiei  $z$ .

Considerăm toate valorile  $x_i$ ,  $i=1,2,\dots,n$ , fără erori și notăm cu  $\pi_k^i$ ,  $k=0,\dots,n-1$ ;  $i=0,1,\dots,n-1$ , erorile termenilor produsului. În aceste condiții pentru  $z_0$  avem eroarea de rotunjire  $\pi_0=0$ , deoarece  $z_0$  inițializat este 1 și are eroarea zero.

$$\begin{aligned}\pi_0 &= 0 \\ \pi_0^i &= \gamma_{11}^i - \gamma_{12}^i \gamma_{31}^i + \gamma_{41}^i \\ \pi_1^i &= \gamma_{12}^i - \gamma_{22}^i + \gamma_{32}^i + \gamma_{42}^i + \pi_0^i\end{aligned}\quad (6.15)$$

$$\pi_{n-1}^i = \gamma_{1,n-1}^i - \gamma_{2,n-1}^i + \gamma_{3,n-1}^i + \gamma_{4,n-1}^i + \pi_{n-2}^i$$

În expresia (6.15) lipsește termenul  $\pi_i^i$ . Din (6.15) rezultă:

$$\pi_{n-1}^i = \sum_{\substack{j=0 \\ i \neq j}}^{n-1} (\gamma_{1j}^i - \gamma_{2j}^i + \gamma_{3j}^i + \gamma_{4j}^i) \quad (6.16)$$

Dacă considerăm  $\gamma_{kj} \leq 10^{-t} = \gamma$  unde  $t$  este mantisa calculatorului și notăm cu  $\pi_i$  eroarea produsului  $z_i$ , avem

$$\pi_i \leq 4(n-1)\gamma \quad (6.17)$$

sau pentru  $z_i$ ,  $i=0,1,2,\dots,n-1$ , avem

$$\pi_i \leq 4(n-1)\gamma \quad i=0,1,2,\dots,n-1 \quad (6.18)$$

Construim graful de procedură pentru expresia  $p_i$ .

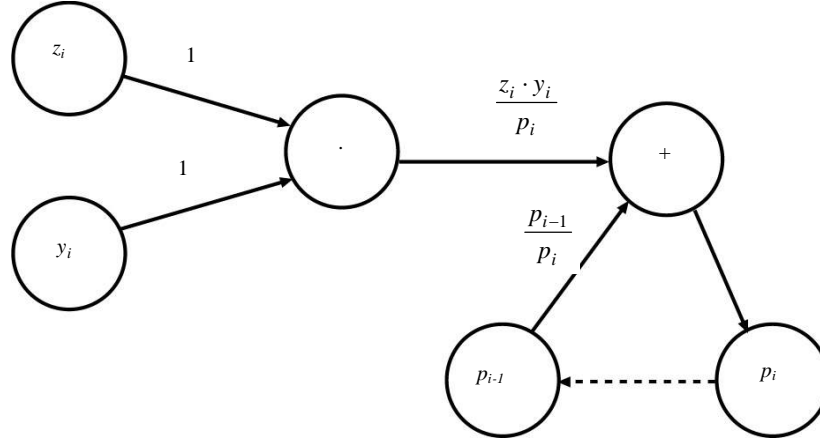


Fig.6.2. Graful de procedură al expresiei  $p_i = \sum_{k=0}^{n-1} y_k z_k$

Se consideră eroarea inițială în punctul  $p_i$  nulă, erorile de rotunjire în nodurile 5,6 le notăm cu  $\gamma_{5i}, \gamma_{6i}, i = 0, 1, \dots, n-1$  și notăm cu  $\varepsilon_i, i = 0, 1, \dots, n-1$  erorile termenilor sumei.

$$\begin{aligned}\varepsilon_0 &= (\pi_0 + e_0 + \gamma_{50}) \frac{z_0 y_0}{p_0} + \gamma_{60} \\ \varepsilon_1 &= (\pi_1 + e_1 + \gamma_{51}) \frac{z_1 y_1}{p_1} + \varepsilon_1 \frac{p_1}{p_2} + \gamma_{61} \\ \varepsilon_{n-1} &= (\pi_{n-1} + e_{n-1} + \gamma_{5,n-1}) \frac{z_n y_{n-1}}{p_{n-1}} + \varepsilon_{n-1} \frac{p_{n-2}}{p_{n-1}} + \gamma_{6,n-1}\end{aligned}\quad (6.19)$$

Știm că  $\pi_i \leq 4(n-1)\gamma$  pentru  $i=0, 1, \dots, n-1$  și considerăm că  $|e_i| < e$  pentru  $i=0, 1, \dots, n-1$ .

În aceste condiții avem :  $|\gamma_{kj}| < \gamma$

$$\varepsilon_{n-1} |p_{n-1}| \leq [(4n-3)\gamma + e] \sum_{i=0}^{n-1} |z_i y_i| + \gamma \sum_{i=0}^{n-1} |p_i| \leq [(4n-3)\gamma + e] \sum_{i=0}^{n-1} p_i + \gamma \sum_{i=0}^{n-1} |p_i|$$

Se mai poate face majorarea  $\sum_{i=0}^{n-1} |p_i| \leq np_{n-1}$  și rezultă:

$$\varepsilon_{n-1} \leq 2\gamma n(2n-1) + en \quad (6.20)$$

expresie care reprezintă eroarea de rotunjire pentru interpolarea lagranjiană.

### 6.1.3. PARTICULARIZĂRI ALE POLINOMULUI LUI LAGRANGE

Dacă considerăm două puncte, interpolarea devine liniară

$$y = y_1 \frac{x - x_2}{x_1 - x_2} + y_2 \frac{x - x_1}{x_2 - x_1} \quad (6.21)$$

Eroarea de trunchiere a interpolării liniare devine:

$$e_T = \frac{f''(\xi)}{2!} (x - x_1)(x - x_2) \quad x_1 < \xi < x_2 \quad (6.22)$$

și eroarea de rotunjire:

$$\varepsilon_2 \leq 12\gamma + 2e \quad (6.23)$$

unde  $\gamma$  și  $e$  sunt date în paragraful 6.1.2.

Pentru trei puncte, interpolarea devine pătratică.

#### 6.1.4. ALGORITMUL 6.1. POLINOMUL LUI LAGRANGE

```
{Variabile
n : numărul de puncte cunoscute, întreg ;
x : abscisele punctelor cunoscute, vector ;
y : ordonatele punctelor cunoscute, vector ;
X̄ : punctul în care se calculează interpolarea, real ;
i,j : contori, întregi ;
sum : variabilă ce reține suma, real ;
prod : variabilă ce reține produsul, real ;
{ sum=0;
pentru i=0..n
    { prod=1;
    pentru j=0..n dacă j ≠ i atunci
        calculează prod = prod *  $\frac{\bar{x} - x_j}{x_i - x_j}$ ;
        sum = sum + yi * prod;
    }

Valoarea interpolată este sum;

}.
```

#### 6.1.5. IMPLEMENTAREA ALGORITMULUI 6.1

```
/* Funcția care implementează metoda lui Lagrange de interpolare
Funcția întoarce valoarea interpolată */
double Lagrange(int n,
                double x[],
                double y[],
                double point)
{
    int i,j;
    double sum=0,prod;
    for(i=0;i<=n;i++)
    {
```

```

prod=1;
for(j=0;j<=n;j++)if(j!=i)prod*=(point-x[j])/(x[i]-x[j]);
sum+=y[i]*prod;
}
return sum;
}

```

## 6.2. POLINOMUL DE INTERPOLARE DE SPEȚA ÎNTÂI AL LUI NEWTON

Acest polinom de interpolare se exprimă funcție de diferențele finite. Fie  $f:[a,b] \rightarrow \mathbf{R}$  și rețeaua  $x_1, x_2, x_3, \dots, x_n$  cu pasul constant  $h$ .

**Definiția 6.1.** Se numește diferență finită de ordinul întâi expresia :

$$\Delta f(x) = f(x+h) - f(x) \quad (6.24)$$

unde  $h$  este pasul constant, iar diferența finită de ordinul  $n$  expresia :

$$\Delta^n f(x) = \Delta(\Delta^{n-1} f(x)) \quad (6.25)$$

Diferențele finite au următoarele proprietăți :

1. Operatorul diferență finită este liniar :

$$\Delta(c_1 f_1 + c_2 f_2) = c_1 \Delta f_1 + c_2 \Delta f_2 \quad (6.26)$$

2. Diferența finită de ordinul  $n$  se calculează cu formula :

$$\Delta^n f(x) = \sum_{k=0}^n (-1)^k C_n^k f(x + (n-k)h) \quad (6.27)$$

3. Diferențele finite mai pot fi obținute și cu ajutorul tabelului 6.2

Tabelul 6.2

$x_i$	$y_i$	$\Delta y_i$	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$	$\Delta^6 y_i$
$x_0$	$y_0$	$\Delta y_0$					
$x_1$	$y_1$	$\Delta y_1$	$\Delta^2 y_0$	$\Delta^3 y_0$			
$x_2$	$y_2$	$\Delta y_2$	$\Delta^2 y_1$	$\Delta^3 y_1$	$\Delta^4 y_0$	$\Delta^5 y_0$	
$x_3$	$y_3$	$\Delta y_3$	$\Delta^2 y_2$	$\Delta^3 y_2$	$\Delta^4 y_1$	$\Delta^5 y_1$	$\Delta^6 y_0$
$x_4$	$y_4$	$\Delta y_4$	$\Delta^2 y_3$	$\Delta^3 y_3$	$\Delta^4 y_2$		
$x_5$	$y_5$	$\Delta y_5$	$\Delta^2 y_4$				
$x_6$	$y_6$						

**Definiția 6.2.** Se numește *putere generalizată de ordinul  $n$  a lui  $x$*  expresia:

$$x^{[n]} = x(x-h)(x-2h)\dots(x-(n-1)h) \quad (6.28)$$

Pentru  $h=0$  puterea generalizată coincide cu puterea obișnuită.

1. Diferența finită a puterii generalizate este:

$$\Delta x^{[n]} = nhx^{[n-1]} \quad (6.29)$$

2. Diferența finită de ordinul  $k$  a puterii generalizate este:

$$\Delta^k x^{[n]} = n(n-1)(n-2)\dots(n-k+1)x^{[n-k]} \quad (6.30)$$

Fie funcția tabelată dată în tabelul (6.1), unde rețeaua  $x_0, x_1, x_2, x_3, \dots, x_n$  este cu pasul constant  $h$ .

Prin cele  $n+1$  puncte trece un polinom de gradul  $n$  pe care îl căutăm sub forma

$$P_n(x) = C_0 + C_1(x-x_0)^{[1]} + C_2(x-x_0)^{[2]} + \dots + C_n(x-x_0)^{[n]} \quad (6.31)$$

unde  $(x-x_0)^{[i]} = (x-x_0)(x-x_1)\dots(x-x_{i-1})$ ,  $i=1, 2, \dots, n$

și coeficienții  $C_0, C_1, \dots, C_n$  sunt necunoscute pe care le vom calcula. Se observă că

$$P_n(x_0) = y_0 = C_0 \quad (6.32)$$

Calculăm diferența finită de ordinul întâi :

$$\Delta P_n(x) = C_1h + 2C_2h(x-x_0)^{[1]} + \dots + nC_nh(x-x_0)^{[n-1]} \quad (6.33)$$

Făcând substituția  $x = x_0$  rezultă  $\Delta P_n(x_0) = C_11!h$ .

Se poate calcula  $C_1 = \frac{\Delta P_n(x_0)}{1!h}$  (6.34)

Se continuă calculul diferențelor finite în punctul  $x_0$  și se observă că :

$$C_k = \frac{\Delta^{(k)} P_n(x_0)}{k!h^k}, \quad \Delta^k P_n(x_0) = \Delta^k y_0, \quad k=0, 1, 2, \dots, n. \quad (6.35)$$

Ținând cont de formulele de calcul ale coeficienților, polinomul lui Newton de interpolare de speța întâi poate fi scris astfel:

$$P_n(x_0) = y_0 + \frac{\Delta y_0}{1!h}(x-x_0)^{[1]} + \frac{\Delta^2 y_0}{2!h^2}(x-x_0)^{[2]} + \dots + \frac{\Delta^n y_0}{n!h^n}(x-x_0)^{[n]} \quad (6.36)$$

Deoarece în calculul coeficienților s-au utilizat diferențele finite la dreapta (tabelul 6.2), polinomul poartă denumirea de polinom a lui Newton de interpolare de speța întâi.

### 6.2.1. ALGORITMUL 6.2. NEWTON 1

*{Variabile*

*n: numărul de puncte cunoscute, întreg;*

*h: pasul constant între abscisele cunoscute, real;*

*y: ordonatele punctelor, vector;*

*xp: abscisa punctului în care se face interpolarea;*

*x<sub>0</sub>: abscisa primului punct cunoscut, real;*

*sum: variabila ce reține sumele parțiale, real;*

*prod: variabila ce reține produsele, real;*

*i, j: contoare, întregi;*

*{ sum=y<sub>0</sub>;*

*prod=1;*

*pentru i=1...n*

*{pentru j=0...n-i*



```

    calculează  $y_j = y_{j+1} - y_j$ 
     $prod = prod * (x_p - (x_0 + (i-1)*h)) * \frac{1}{h} * \frac{1}{i};$ 
     $sum = sum + y_0 * prod;$  }
    valoarea interpolată este sum ;
  } }.

```

### 6.2.2. IMPLEMENTAREA ALGORITMULUI 6.2

```

/* Funcția care implementează metoda de interpolare a lui Newton
   de speța întâia.
   Funcția întoarce valoarea interpolată
   */
double Newton1(int n,
               double vi,
               double pas,
               double y[],
               double point)
{
    double sum=y[0],prod=1;
    int i,j;
    for(i=1;i<=n;i++)
    {
        for(j=0;j<=n-i;j++)y[j]=y[j+1]-y[j];
        prod*=(point-(vi+(i-1)*pas))/(i*pas);
        sum+=y[0]*prod;
    }
    return sum;
}

```

## 6.3. POLINOMUL DE INTERPOLARE DE SPEȚA A DOUA AL LUI NEWTON

Pentru funcția dată în tabelul 6.1 se caută un polinom de gradul  $n$  care trece prin cele  $n+1$  puncte, sub forma:

$$P_n(x) = C_0 + C_1(x - x_n) + C_2(x - x_n)(x - x_{n-1}) + \dots + C_n(x - x_n)(x - x_{n-1})\dots(x - x_1) \quad (6.37)$$

Polinomul mai poate fi scris și funcție de puterea generalizată astfel

$$P_n(x) = C_0 + C_1(x - x_n)^{[1]} + C_2(x - x_{n-1})^{[2]} + C_3(x - x_{n-2})^{[3]} + \dots + C_n(x - x_1)^{[n]} \quad (6.38)$$

Se observă că  $P_n(x_n) = y_n = C_0$  (6.39)

Calculăm diferența finită de ordinul întâi

$$\Delta P_n(x) = C_1 1!h + 2C_2h(x - x_{n-1})^{[1]} + 3C_3h(x - x_{n-2})^{[2]} + \dots + nC_nh(x - x_1)^{[n-1]} \quad (6.40)$$

$$\Delta P_n(x_{n-1}) = \Delta y_{n-1} = C_1 1!h \text{ pentru } x = x_{n-1} \text{ și rezultă } C_1 = \frac{\Delta y_{n-1}}{1!h}.$$

Continuând calculele diferențelor finite în punctele  $x_{n-2}, x_{n-3}, \dots, x_{n-k}$ , pentru rangul  $k$  rezultă formula de calcul al coeficientului  $C_k$ :

$$C_k = \frac{\Delta^{(k)} y_{n-k}}{k!h^k} \quad (6.41)$$

Substituind  $k = 0, 1, 2, \dots, n$  în formula (6.41) se obțin coeficienții polinomului. Ținând cont de formula (6.41) polinomul de gradul  $n$  (6.38) poate fi scris sub forma:

$$P_n(x) = y_n + \frac{\Delta y_{n-1}}{1!h} (x - x_n)^{[1]} + \frac{\Delta^2 y_{n-2}}{2!h^2} (x - x_{n-1})^{[2]} + \dots + \frac{\Delta^n y_0}{n!h^n} (x - x_1)^{[n]} \quad (6.42)$$

Acest polinom este numit *polinomul lui Newton de interpolare de speța a doua* deoarece s-au utilizat diferențele finite la stânga (tabelul 6.2). Dacă punctul de aproximare a funcției se găsește în apropierea lui  $x_n$ , se recomandă utilizarea metodei de speța a doua deoarece dă erori mai mici.

### 6.3.1. ALGORITMUL 6.3. NEWTON 2

```
{Variabile
n:numărul de puncte cunoscute, întreg;
h:pasul constant între abscisele cunoscute, real;
y:ordonatele punctelor, vector;
xp:abscisa punctului în care se face interpolarea;
xn:abscisa maximă a punctelor cunoscute, real;
sum:variabila ce reține sumele parțiale, real;
prod:variabila ce reține produsele, real;
i,j:contoare, întregi;
{ sum=yn;
prod=1;
pentru i=1...n
{pentru j=n...i
  calculează yj=yj-yj-1
  prod = prod * (xp - (xn - (i-1)*h)) * 1/h * 1/i;
  sum=sum+yn*prod;
}
valoarea interpolată este sum;
}
```

### 6.3.2. IMPLEMENTAREA ALGORITMULUI 6.4

*/\* Funcția care implementează metoda de interpolare a lui Newton*

```

de speța a doua.
Funcția întoarce valoarea interpolată
*/
double Newton2(int n,
                double vi,
                double pas,
                double y[],
                double point)
{
    double sum,prod,vf;
    int i,j;
    vf=vi+n*pas;
    sum=y[n];prod=1;
    for(i=1;i<=n;i++)
    {
        for(j=n;j>=i;j--)y[j]=y[j]-y[j-1];
        prod*=(point-(vf-(i-1)*pas))/(i*pas);
        sum+=y[n]*prod;
    }
    return sum;
}

```

## 6.4. POLINOMUL LUI NEWTON DE INTERPOLARE CU DIFERENȚE DIVIZATE

Fie funcția  $f(x)$  dată sub forma celei prezentate în tabelul (6.1) unde rețeaua  $x_0, x_1, x_2, \dots, x_n$  din domeniul de definiție al funcției nu are pas constant.

**Definiția 6.3.** Se numește *diferență divizată de ordinul  $k+i$*  a funcției  $f$  expresia :

$$f(x_{i-1}, x_i, x_{i+1}, \dots, x_{i+k}) = \frac{f(x_i, x_{i+1}, \dots, x_{i+k}) - f(x_{i-1}, x_i, \dots, x_{i+k-1})}{x_{i+k} - x_{i-1}} \quad (6.43)$$

Se determină polinomul de gradul  $n$ , de forma :

$$P_n(x) = C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) + \dots + C_n(x - x_0)(x - x_1) \dots (x - x_{n-1}) \quad (6.44)$$

cunoscând  $n+1$  puncte  $(x_i, y_i)$ ,  $i=0, \dots, n$ , care verifică polinomul.

Se observă că  $P_n(x_0) = y_0 = C_0$ .

Calculăm diferența divizată de ordinul întâi pentru polinomul  $P_n(x)$  și se face  $x = x_1$

$$P_n(x_0, x_1) = C_1.$$

Calculând în continuare, se determină diferența divizată de ordinul  $k$  și luând pe  $x = x_k$  se obține valoarea coeficientului  $C_k$  :

$$C_k = P_n(x_0, x_1, x_2, \dots, x_k), \quad k=0, 1, \dots, n. \quad (6.45)$$

Ținând cont de formula (6.45), polinomul (6.44) se scrie sub forma:

$$P_n(x) = y_0 + P_n(x_0, x_1)(x - x_0) + P_n(x_0, x_1, x_2)(x - x_0)(x - x_1) + \dots + P_n(x_0, \dots, x_n)(x - x_0)(x - x_1) \dots (x - x_{n-1}) \quad (6.46)$$

unde fiecare diferență divizată se calculează cu ajutorul formulei (6.43). Polinomul obținut (6.46) poartă numele de *polinomul lui Newton de interpolare cu diferențe divizate*.

### 6.4.1. ALGORITMUL 6.4. NEWTON 3

```
{Variabile
n : numărul de puncte date ale funcției, întreg ;
x : abscisele punctelor date, vector ;
y : ordonatele punctelor date, vector ;
x̄ : punctul în care se interpolează funcția, real ;
i, j : contoare, întregi ;
sum : variabilă ce reține sumele parțiale, real ;
prod : variabilă ce reține produsele parțiale, real ;
{ sum=y0;
prod=1;
pentru i=1...n
pentru j=0...n-i calculează yj =  $\frac{y_{j+1} - y_j}{x_{j+i} - x_j}$ ;
prod= prod*( x̄ -xi-1);
sum=sum+y0*prod;
}
valoarea interpolată este sum;
}
```

### 6.4.2. IMPLEMENTAREA ALGORITMULUI 6.4

```
/* Funcția care implementează metoda de interpolare a lui Newton
cu diferențe divizate.
Funcția întoarce valoarea interpolată.
*/
double NewtonDD(int n,
double x[],
double y[],
double point)
{
int i, j;
double sum, prod;
sum=y[0]; prod=1;
for(i=1; i<=n; i++)
```

```

{
  for(j=0;j<=n-i;j++)y[j]=(y[j+1]-y[j])/(x[j+i]-x[j]);
  prod*=(point-x[i-1]);
  sum+=y[0]*prod;
}
return sum;
}

```

## 6.5. METODA LUI AITKEN DE INTERPOLARE

Metoda lui Aitken de interpolare dă același rezultat ca și metoda lui Lagrange de interpolare doar că prin această metodă nu se determină un polinom, ci se realizează mai multe interpolări liniare. Cu fiecare interpolare, numărul punctelor rămase se micșorează cu o unitate, iar funcția ce trece prin două puncte de la etapa curentă crește în grad, astfel că în final, dacă sunt date  $n+1$  puncte, se obține o funcție de gradul  $n$ .

Considerăm funcția dată în tabelul 6.1

Tabelul 6.3

$x$	$y$					
$x_0$	$y_0$					
$x_1$	$y_1$	$y_{01}$				
$x_2$	$y_2$	$y_{02}$	$y_{012}$			
$x_3$	$y_3$	$y_{03}$	$y_{013}$	$y_{0123}$		
$x_4$	$y_4$	$y_{04}$	$y_{014}$	$y_{0124}$	----	
-----	----	----	----	----	----	
$x_n$	$y_n$	$y_{0n}$	$y_{01n}$	$y_{012n}$	----	$y_{0123...n}$

$$y_{0j} = \frac{x_j - \bar{x}}{x_j - x_0} y_0 + \frac{\bar{x} - x_0}{x_j - x_0} y_j \quad j=1, 2, 3, \dots, n$$

$$y_{01j} = \frac{x_j - \bar{x}}{x_j - x_1} y_{01} + \frac{\bar{x} - x_1}{x_j - x_1} y_{0j} \quad j=2, 3, \dots, n \quad (6.47)$$

$$y_{012...n} = \frac{x_n - \bar{x}}{x_n - x_{n-1}} y_{012...(n-2)(n-1)} + \frac{\bar{x} - x_{n-1}}{x_n - x_{n-1}} y_{012...(n-2)n} \quad (6.48)$$

Valoarea interpolată a funcției tabelate în punctul  $\bar{x}$  este  $y_{012...n}$ .

### 6.5.1. ALGORITMUL 6.5. METODA LUI AITKEN

{Variabile

*n* : numărul de puncte date ale funcției, întreg ;  
*x* : abscisele punctelor date, vector ;  
*y* : ordonatele punctelor date, vector ;  
*x<sub>p</sub>*: punctul în care se interpolează funcția, real ;  
*i,j* : contoare, întregi ;  
 {  
     pentru *i*=1...*n*  
     pentru *j*=*i*...*n*  
     calculează  
         
$$y_j = y_{i-1} \frac{x_p - x_j}{x_{i-1} - x_j} + y_j \frac{x_p - x_{i-1}}{x_j - x_{i-1}};$$
  
     }  
     valoarea interpolată este *y<sub>n</sub>* ;  
 }

### 6.5.2. IMPLEMENTAREA ALGORITMULUI 6.5

```

/* Funcția care implementează metoda lui Aitken de interpolare
Funcția întoarce valoarea interpolată */
double Aitken(int n,
               double x[],
               double y[],
               double point)
{
    int i,j;
    for(i=1;i<=n;i++)
        for(j=i;j<=n;j++)
            y[j]=y[i-1]*(point-x[j])/(x[i-1]-x[j])+y[j]*(point-x[i-1])/(x[j]-x[i-1]);
    return y[n];
}
  
```

## 6.6. INTERPOLAREA CU FUNCȚII SPLINE

Cuvântul splin provine din engleză și înseamnă o riglă elastică de care dacă se agață greutatea poate fi făcută să treacă prin diferite puncte dorite, cuprinse între capetele riglei.

Considerăm dată o funcție tabelată, reprezentată în tabelul 6.4.

Tabelul 6.4

<i>x</i>	<i>x</i> <sub>1</sub>	<i>x</i> <sub>2</sub>	<i>x</i> <sub>3</sub>	----	<i>x</i> <sub><i>n</i></sub>
<i>y</i>	<i>y</i> <sub>1</sub>	<i>y</i> <sub>2</sub>	<i>y</i> <sub>3</sub>	----	<i>y</i> <sub><i>n</i></sub>

Considerăm  $x_1 = a$  și  $x_n = b$ ,  $[a, b]$  fiind intervalul de definiție a funcției  $f$ , iar abscisele  $x_1, x_2, \dots, x_n$  o diviziune  $\Delta$  a intervalului de definiție. Valorile funcției sunt

$$y_i = f(x_i), \quad i = 1, 2, \dots, n \quad (6.49)$$

**Definiția 6.4.** Se numește *funcție spline de ordinul  $n$  relativ la diviziunea  $\Delta$*  a intervalului  $[a, b]$  o funcție  $S: [a, b] \rightarrow \mathbf{R}$  de clasă  $C^{m-1}[a, b]$  ale cărei restricții  $S_i(x)$  pe fiecare interval  $[x_i, x_{i+1}]$  al diviziunii sunt polinoame de ordinul  $m$ , adică:

$$S_i(x) = P_m^i(x), \quad \text{dacă} \quad x \in [x_i, x_{i+1}], \quad i = 1, 2, \dots, (n-1) \quad (6.50)$$

Funcția  $S(x)$  este netedă pe porțiuni deoarece are primele  $(m-1)$  derivate continue pe  $[a, b]$ , iar derivata de ordinul  $m$  este discontinuă în  $x_i$ ,  $i = 1, 2, \dots, n$ . Gradul de netezire al funcției este  $m$ . Restricțiile funcției sunt polinoamele:

$$S_i(x) = A_i x^m + B_i x^{m-1} + C_i x^{m-2} + E_i x^{m-3} + \dots + R_i \quad (6.51)$$

dacă,  $x \in [x_i, x_{i+1}]$ ,  $i = 1, 2, \dots, (n-1)$

Aceste funcții sunt derivabile până la  $(m-1)$  și sunt continue împreună cu derivatele. Derivata de ordinul  $(m-1)$  a lui  $S_i(x)$  pe intervalul  $[x_i, x_{i+1}]$  este o funcție liniară și trece prin punctele  $(x_i, D_i)$  și  $(x_{i+1}, D_{i+1})$  unde,  $D_i = S_i^{(m-1)}(x)$   $i = 1, 2, \dots, n$ .

Rezultă ecuația liniară:

$$S_i^{(m-1)}(x) = \frac{D_{i+1}(x - x_i) + D_i(x_{i+1} - x)}{h_i} \quad (6.52)$$

unde  $h_i = x_{i+1} - x_i$ ,  $i = 1, 2, \dots, (n-1)$

Integrând de  $m-1$  ori relația (6.52) se obține:

$$S_i^{(m-2)}(x) = \frac{D_{i+1}(x - x_i)^2 - D_i(x_{i+1} - x)^2}{2h_i} + C_{1i} \quad (6.53)$$

$$S_i^{(m-3)}(x) = \frac{D_{i+1}(x - x_i)^3 + D_i(x_{i+1} - x)^3}{6h_i} + C_{1i}x + C_{2i} \quad (6.54)$$

$$S_i'(x) = \frac{D_{i+1}(x - x_i)^{m-1} + (-1)^{m-2}(x_{i+1} - x)^{m-1}D_i}{(n-1)!h_i} + \frac{C_{1i}x^{m-3}}{(m-3)!} + \frac{C_{2i}x^{m-4}}{(m-4)!} + \dots + C_{m-1,i} \quad (6.55)$$

$$S_i(x) = \frac{D_{i+1}(x - x_i)^m + (-1)^{m-1}(x_{i+1} - x)^{m-1}D_i}{n!h_i} + \frac{C_{1i}x^{m-2}}{(m-2)!} + \frac{C_{2i}x^{m-3}}{(m-3)!} + \dots + C_{m-2,i}x + C_{m-1,i} \quad (6.56)$$

Pentru întreg intervalul  $[a, b]$  rezultă un sistem liniar punând condiția ca  $S_i(x_i) = y_i$ ,  $i = 1, 2, \dots, n$  și continuitatea celor  $(m-1)$  ecuații în toate punctele  $x_i$ . În extremele  $x_1$  și  $x_n$  se scrie polinomul lui Lagrange, îl derivăm până la  $m-1$  și aflăm corespunzător valorile derivatelor în  $x_1$  și  $x_n$ . Rezultă necunoscutele:

$D_i, D_{i+1}, C_{1i}, \dots, C_{m-1,i}$  pentru fiecare interval. În acest caz sunt  $n + (m-1) + (n-1)$  necunoscute și  $n + (m-1) + (n-1)$  ecuații. Sistemul fiind liniar se rezolvă cu una din metodele de la capitolul 3.

Algoritmul și programul s-au realizat pentru restricțiile  $S_i$  polinoame de gradul 3.

În acest caz:

$$S_i(x) = A_i x^3 + B_i x^2 + C_i x + E_i \quad (6.57)$$

$$\begin{aligned}
S''_i(x) &= \frac{D_{i+1}(x-x_i) + D_i(x_{i+1}-x)}{h_i} \\
S'_i(x) &= \frac{D_{i+1}(x-x_i)^2 - D_i(x_{i+1}-x)^2}{2h_i} + C_{1i} \\
S_i(x) &= \frac{D_{i+1}(x-x_i)^3 - D_i(x_{i+1}-x)^3}{6h_i} + C_{1i}x + C_{2i}
\end{aligned} \tag{6.58}$$

Din  $S(x_i) = y_i$  și  $S(x_{i+1}) = y_{i+1}$  rezultă:

$$\begin{aligned}
C_{1i} &= \frac{y_{i+1} - y_i}{h_i} - \frac{D_{i+1} - D_i}{6} h_i \\
C_{2i} &= \frac{x_{i+1}y_i - x_iy_{i+1}}{h_i} - \frac{D_i x_{i+1} - D_{i+1} x_i}{6} h_i
\end{aligned}$$

Din identificarea relațiilor (6.57) și (6.58) rezultă :

$$A_i = \frac{D_{i+1} - D_i}{6h_i} ; \quad B_i = \frac{D_i x_{i+1} - D_{i+1} x_i}{2h_i} \tag{6.59}$$

$$\begin{aligned}
C_i &= \frac{D_{i+1}x_i^2 - D_i x_{i+1}^2}{2h_i} + \frac{y_{i+1} - y_i}{h_i} - \frac{D_{i+1} - D_i}{6} h_i \\
E_i &= \frac{x_{i+1}^3 D_i - x_i^3 D_{i+1}}{6h_i} + \frac{y_i x_{i+1} - y_{i+1} x_i}{h_i} - \frac{D_i x_{i+1} - D_{i+1} x_i}{6} h_i
\end{aligned} \tag{6.60}$$

Din continuitatea primei derivate în punctul  $x_i$ ,  $S'_{i-1}(x_i) = S'_i(x_i)$  rezultă:

$$h_{i-1}D_{i-1} + 2(h_i + h_{i-1})D_i + h_i D_{i+1} = \left( \frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \right) 6 \tag{6.61}$$

Considerând derivatele de ordinul întâi în punctele  $x_1$  și  $x_2$  egale cu:

$$y'_1 = \frac{y_2 - y_1}{x_2 - x_1} - \frac{y_3 - y_2}{x_3 - x_2} + \frac{y_3 - y_1}{x_3 - x_1} \tag{6.62}$$

respectiv

$$y'_n = -\frac{y_{n-1} - y_{n-2}}{x_{n-1} - x_{n-2}} + \frac{y_n - y_{n-1}}{x_n - x_{n-1}} + \frac{y_n - y_{n-2}}{x_n - x_{n-2}} \tag{6.63}$$

rezultă sistemul tridiagonal în  $D_i$   $i = 1, 2, 3, \dots, n$



$$\begin{cases}
 2h_1 D_1 + h_1 D_2 = 6 \left( \frac{y_2 - y_1}{h_1} - y'_1 \right) \\
 h_1 D_1 + 2(h_1 + h_2) D_2 + h_2 D_3 = 6 \left( \frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_3} \right) \\
 \dots\dots\dots \\
 h_{i-1} D_{i-1} + 2(h_{i-1} + h_i) D_i + h_i D_{i+1} = 6 \left( \frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i+1}}{h_i} \right) \\
 \dots\dots\dots \\
 h_{n-2} D_{n-2} + 2(h_{n-2} + h_{n-1}) D_{n-1} + h_{n-1} D_n = 6 \left( \frac{y_n - y_{n-1}}{h_{n-1}} - \frac{y_{n-1} - y_{n-2}}{h_{n-1}} \right) \\
 h_{n-1} D_{n-1} + 2h_{n-1} D_n = 6 \left( y'_n - \frac{y_n - y_{n-1}}{h_i} \right)
 \end{cases} \quad (6.64)$$

unde  $y'_1$  și  $y'_n$  sunt date de expresiile (6.62), respectiv (6.63).

Din sistemul (6.64) rezultă valorile lui  $D_i$ ,  $i = 1, 2, \dots, n$ . Din (6.59) rezultă coeficienții restricțiilor pe fiecare interval, restricții care aproximează funcția dată. Dacă se dă  $\bar{x}$  în care trebuie calculată funcția, se stabilește intervalul în care se găsește  $\bar{x}$  și se calculează valoarea restricției funcției pe acest interval în punctul  $\bar{x}$ .

### 6.6.1. ALGORITMUL 6.6. FUNCȚII SPLINE

*{ Variabile*

$n$  : numărul de puncte date ale funcției, întreg ;

$x$  : abscisele punctelor date, vector ;

$y$  : ordonatele punctelor date, vector ;

$x_p$  : punctul în care se interpolează funcția, real ;

$A, B, C$  : vectorii elementelor diagonale ale sistemului tridiagonal;

$TL$  : vectorul termenilor liberi al sistemului tridiagonal;

$h$  : vectorul pașilor punctelor pe abscisă;

$S$  : vectorul soluțiilor sistemului tridiagonal;

$derx1$  : derivata în punctul  $x1$ , real;

$derx2$  : derivata în punctul  $x2$ , real;

$num$  : variabila ce dă numărul intervalului în care se găsește  $x_p$ , întreg;

$valint$  : variabilă ce reține valoarea polinomului de gradul trei în punctul

$x_p$ , real;

$i, j$  : contoare, întregi;

{ pentru  $i = 1$  până la  $n-1$

calculează  $h_i = x_{i+1} - x_i$ ;

calculează  $derx1 = \frac{y_2 - y_1}{x_2 - x_1} - \frac{y_3 - y_2}{x_3 - x_2} + \frac{y_3 - y_1}{x_3 - x_1}$  ;

calculează  $derxn = -\frac{y_{n-1} - y_{n-2}}{x_{n-1} - x_{n-2}} + \frac{y_n - y_{n-1}}{x_n - x_{n-1}} + \frac{y_n - y_{n-2}}{x_n - x_{n-2}};$   
 construiește sistemul tridiagonal

$$\left\{ \begin{array}{l} 2.h_1.D_1 + h_1.D_2 + 0.D_3 + \text{-----} + 0.D_n = 6\left(\frac{y_2 - y_1}{h_1} - derx1\right) \\ 0.D_1 + h_1.D_2 + 2(h_1 + h_2)D_2 + h_2.D_3 + 0.D_4 + \text{-----} + 0.D_n = 6\left(\frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_{i-1}}\right) \\ \text{-----} \\ 0.D_1 + 0.D_2 + 0.D_3 + \text{-----} + h_{n-1}D_{n-1} + 2.h_{n-1}.D_n = 6\left(derxn - \frac{y_n - y_{n-1}}{h_{n-1}}\right) \end{array} \right.$$

Rezolvă sistemul conform algoritmului (3.8) din capitolul 3.

Calculează  $S = \text{Tridiagonal}(A, B, C, TL);$

/\* Se caută intervalul care conține punctul de interpolare \*/

$i = 1$

repetă

$i = i + 1;$

până când  $x_p < x_s[i];$

Cu formulele (6.59) calculează valorile  $A_i, B_i, C_i, E_i$  pentru  $i$  găsit;

calculează  $\text{val int} = A_s \cdot x_p^3 + B_s \cdot x_p^2 + C_s \cdot x_p + E_s;$

tipărește  $\text{val int}$

}

## 6.6.2. IMPLEMENTAREA ALGORITMULUI 6.6

/\* Funcția care implementează interpolarea prin funcții spline cubice

Funcția întoarce

1 dacă se găsește valoarea ;

0 dacă sistemul tridiagonal nu se poate rezolva;

2 dacă valoarea de interpolat nu se află în interval;

\*/

int SPLINE(int ord,

double xi[],

double yi[],

double point,

double \*valoare

)

{

static double a[NrMax], b[NrMax], c[NrMax], d[NrMax], der2[NrMax];

```

int i,cod,loc;
double d1,d2;
if( (point<xi[1])||(point>xi[ord]) )return 2;
/*Pregătesc sistemul tridiagonal pentru aflarea derivatei a doua */
/*Calculez prima derivată în capete */
d1=(yi[2]-yi[1])/(xi[2]-xi[1])-(yi[3]-yi[2])/(xi[3]-xi[2])+(yi[3]-yi[1])
/(xi[3]-xi[1]);
d2=-(yi[ord]-yi[ord-2])/(xi[ord]-xi[ord-2])+(yi[ord]-yi[ord-1])
/(xi[ord]-xi[ord-1])+(yi[ord]-yi[ord-2])/(xi[ord]-xi[ord-2]);
/*Calculez coeficienții primei linii */
a[1]=0;
b[1]=2*(xi[2]-xi[1]);
c[1]=xi[2]-xi[1];
d[1]=6*( (yi[2]-yi[1])/(xi[2]-xi[1])-d1);
/*Calculez coeficienții liniilor 2...ord-1*/
for(i=2;i<=ord-1;i++)
{
    a[i]=xi[i]-xi[i-1];
    b[i]=2*(xi[i+1]-xi[i-1]);
    c[i]=xi[i+1]-xi[i];
    d[i]=6*( (yi[i+1]-yi[i])/(xi[i+1]-xi[i])-(yi[i]-yi[i-1])
/(xi[i]-xi[i-1]) );
}

/* Calculez coeficienții ultimei linii */
a[ord]=xi[ord]-xi[ord-1];
b[ord]=2*(xi[ord]-xi[ord-1]);
c[ord]=0;
d[ord]=6*(d2-(yi[ord]-yi[ord-1])/(xi[ord]-xi[ord-1]));
cod=SolveTridi(ord,a,b,c,d,der2);
if(cod==0)return 0;
/*Calculez coeficienții funcțiilor Spline locale
Utilizez aceleași variabile ca pentru sistemul tridiagonal */
for(i=1;i<=ord-1;i++)
{
    a[i]=(der2[i+1]-der2[i])/( 6*(xi[i+1]-xi[i]));
    b[i]=(der2[i]*xi[i+1]-der2[i+1]*xi[i])/(2*(xi[i+1]-xi[i]));
    c[i]=( der2[i+1]*pow(xi[i],2)-der2[i]*pow(xi[i+1],2) )
/(2*(xi[i+1]-xi[i]))+(yi[i+1]-yi[i])
/(xi[i+1]-xi[i])-a[i]*pow(xi[i+1]-xi[i],2);
    d[i]=(der2[i]*pow(xi[i+1],3)-der2[i+1]*pow(xi[i],3))
/(6*(xi[i+1]-xi[i]))+(yi[i]*xi[i+1]-yi[i+1]*xi[i])
/(xi[i+1]-xi[i])-b[i]*pow( ( xi[i+1]-xi[i]),2)/3;
}
/*Localizez valoarea de interpolat pentru a afla ce funcție îi aplic */
loc=1;

```

```

while (point < xi[loc]) loc++;
*valoare = a[loc]*pow(point,3) + b[loc]*pow(point,2) + c[loc]*point + d[loc];
return 1;
}

```

## 6.7. INTERPOLAREA FUNCȚIILOR PERIODICE

Se consideră funcția periodică  $f: [a, b] \rightarrow \mathbf{R}$  cu proprietatea că  $f(a) = f(b)$  unde  $T = b - a$  este perioada funcției.

Experimental s-au obținut  $n$  valori ale funcției pe perioada  $[a, b]$  prezentate în tabelul 6.5 și se cere o valoare a funcției  $f$  într-un punct  $x_p \in [a, b]$ , dar  $x_p \neq x_i$ ,  $i = 1, 2, \dots, n$ .

Tabelul 6.5

$x$	$x_1 = a$	$x_2$	$x_3$	----	$x_{2n} = b$
$y$	$y_1$	$y_2$	$y_3$	----	$y_{2n}$

Pentru determinarea acestei valori se determină un polinom de interpolare a funcției periodice.

**Definiția 6.5.** Se numește sistem fundamental de funcții periodice șirul de funcții  $\varphi$ ,  $i=0, 1, 2, \dots, n$  definite pe  $[a, b]$ , dacă: sunt continue, liniar independente pe  $[a, b]$ ,  $\varphi(a) = \varphi(b)$  pentru orice  $i=1, 2, \dots, n$ , iar funcția

$$H(x) = \sum_{i=0}^n a_i \varphi_i \quad \text{cu} \quad \sum_{i=0}^n a_i^2 > 0 \quad (6.65)$$

are cel mult  $n$  rădăcini pe  $[a, b]$ .

Se face transformarea  $t = \frac{2\pi(x-a)}{b-a}$  pentru a transforma intervalul  $[a, b]$  în intervalul  $[0, 2\pi]$ . În acest caz se consideră sistemul de funcții periodice fundamental pe  $[a, b]$ .

$$1, \cos x, \sin x, \cos 2x, \sin 2x, \dots, \cos nx, \sin nx \quad (6.66)$$

**Propoziția 1:** Fie funcția periodică  $f: [0, 2\pi] \rightarrow \mathbf{R}$  și punctele  $(x_i, f(x_i))$ ,  $i=0, 1, \dots, n$  unde  $x_i \in [0, 2\pi]$ ,  $x_i \neq x_j$ . Atunci polinomul de interpolare a funcției date este:

$$I_n = \sum_{i=0}^n f(x_i) \prod_{\substack{j=0 \\ j \neq i}}^n \frac{\sin \frac{x - x_j}{2}}{\sin \frac{x_i - x_j}{2}} \quad (6.67)$$

Se observă o analogie între polinomul de interpolare al lui Lagrange și polinomul de interpolare a funcțiilor periodice (6.67).

### 6.7.1. ALGORITMUL 6.7. FUNCȚII PERIODICE

{Variabile

```

a : limita stângă a intervalului, real ;
b : limita dreaptă a intervalului, real ;
n : numărul de valori cunoscute ale funcției, întreg ;
x : abscisele punctelor cunoscute, vector ;
y : ordonatele punctelor cunoscute, vector ;
 $\bar{x}$  : punctul de interpolare a funcției, real;
i,j : contoare, întregi ;
sum : variabilă pentru a reține sumele, real;
prod : variabilă pentru a reține produsele, real ;

{  $T = \frac{\pi}{b-a}$  ;
  sum=0;
  pentru i=0,1,...,n
    {prod=1;
      pentru j=0,1,...,n dacă j≠i atunci
        calculează  $prod = prod * \frac{\sin[T*(x-x_j)]}{\sin[T*(x_i-x_j)]}$  ;
        calculează  $sum = sum + y_i * prod$ ;
      }
    valoarea interpolată este sum
  }
}
```

### 6.7.2. IMPLEMENTAREA ALGORITMULUI 6.7

```

/* Funcția care implementează metoda de interpolare a funcțiilor
periodice.
Funcția întoarce valoarea interpolată */
double IFPer(int n,
             double x[],
             double y[],
             double T,
             double point)
{
  int i,j;
  double sum=0,prod;

  for(i=0;i<=n;i++)
  {
    prod=1;
    for(j=0;j<=n;j++)
      if(j!=i) prod*=sin(pi*(point-x[j])/T)/sin(pi*(x[i]-x[j])/T);
    sum+=y[i]*prod;
  }
}
```

```

}
return sum;
}

```

## 6.8. INTERPOLAREA FUNCȚIILOR DE MAI MULTE VARIABLE

Vom considera o funcție de două variabile:  $f: \mathbf{E} \rightarrow \mathbf{R}$  unde  $\mathbf{E} \subset \mathbf{R}^2$  de forma  $f = g(x, y)$ . Se consideră următoarele puncte cunoscute  $(x_i, y_j, f(x_i, y_j))$  pentru  $i = 0, 1, 2, \dots, n$  și  $j = 0, 1, \dots, n$ .

Se pune problema determinării unui polinom de gradul  $n$  care să satisfacă condițiile  $P_n(x_i, y_i) = f(x_i, y_i)$  cu  $i, j = 0, 1, 2, \dots, n$ . Fie polinomul de grad  $n$  în două variabile:

$$P_n(x, y) = a_{00} + a_{10}x + a_{01}y + a_{11}xy + \dots + a_{1n}xy^n \quad (6.68)$$

Coeficienții polinomului pot fi puși sub formă matriceală astfel:

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} & \dots & a_{0(n-1)} & a_{0n} \\ a_{10} & a_{11} & a_{12} & a_{13} & \dots & a_{1(n-1)} & 0 \\ \hline a_{(n-2)0} & a_{(n-2)1} & a_{(n-2)2} & 0 & \dots & 0 & 0 \\ a_{(n-1)0} & a_{(n-1)1} & 0 & 0 & \dots & 0 & 0 \\ a_{n0} & 0 & 0 & 0 & \dots & 0 & 0 \end{pmatrix} \quad (6.69)$$

Pentru determinarea acestor coeficienți avem nevoie de  $\frac{(n+1)(n+2)}{2}$  ecuații.

Aceste ecuații le putem obține numai din punctele date  $(x_i, y_i, f(x_i, y_i))$  care trebuie să fie tot în număr de  $\frac{(n+1)(n+2)}{2}$ .

Ca urmare, punctele în care se cunoaște funcția este necesar să se dea sub formă triunghiulară:

$$\begin{pmatrix} (x_0, y_0) & (x_0, y_1) & (x_0, y_2) & \dots & (x_0, y_n) \\ (x_1, y_0) & (x_1, y_1) & (x_1, y_2) & \dots & 0 \\ \hline (x_{n-1}, y_0) & (x_{n-1}, y_1) & 0 & \dots & 0 \\ (x_n, y_0) & 0 & 0 & \dots & 0 \end{pmatrix} \quad (6.70)$$

În acest caz punctele nu sunt așezate pe o curbă de gradul  $n$  ceea ce conduce la un determinant  $\Delta \neq 0$ , deci la o determinare unică a coeficienților polinomului (6.68). Se caută un polinom de gradul  $n$  sub forma:

$$P_n(x, y) = a_{00} + a_{10}(x - x_0)^{[1]} + a_{01}(y - y_0)^{[1]} + a_{20}(x - x_0)^{[2]} + a_{11}(x - x_0)^{[1]}(y - y_0)^{[1]} + a_{02}(y - y_0)^{[2]} + \dots + a_{0n}(y - y_0)^{[n]} \quad (6.71)$$

Aplicăm diferențele finite pentru funcțiile de două variabile:

$$\Delta_{x^i y^j}^{i+j} f(x_0, y_0) = \Delta_{x^i y^j}^{i+j} P_n(x_0, y_0) = a_{ij} h^i k^j i! j! \quad (6.72)$$

Ținând cont de formula (6.72) în expresia (6.71) rezultă:

$$a_{ij} = \frac{\Delta_{x^i y^j}^{i+j} f(x_0, y_0)}{h^i k^j i! j!}$$

Introducând coeficienții calculați în expresia polinomului (6.71) rezultă polinomul de interpolare pentru funcții de două variabile:

$$P_n(x, y) = \sum_{\substack{i,j=0 \\ i+j \leq n}}^n \frac{\Delta_{x^i y^j}^{i+j} f(x_0, y_0)}{h^i k^j i! j!} (x - x_0)^{[i]} (y - y_0)^{[j]} \quad (6.73)$$

Formula se poate extinde la funcții multidimensionale, dar trebuie să fim atenți la modul de selecție a punctelor în care se alege funcția pentru a putea fi posibil calculul coeficienților polinomului de interpolare.

În aplicații se utilizează des pentru interpolarea funcțiilor de două variabile, polinomul lui Lagrange extins. Considerăm  $(n+1)(m+1)$  puncte distincte în care este dată funcția  $f(x_i, y_j), i = 0, 1, \dots, n; j = 0, 1, \dots, m$ . Se caută polinomul  $P(x, y)$  de grad cel mult  $n$  în  $x$  și  $m$  în  $y$  astfel ca:  $P(x_i, y_j) = f(x_i, y_j), i = 0, 1, \dots, n; j = 0, 1, 2, \dots, m$ . Polinomul lui Lagrange pentru două variabile are forma:

$$L(x, y) = \sum_{i=0}^n \sum_{j=0}^m f(x_i, y_j) \prod_{\substack{k=0 \\ k \neq i}}^n \frac{x - x_k}{x_i - x_k} \prod_{\substack{k=0 \\ k \neq j}}^m \frac{y - y_k}{y_j - y_k}. \quad (6.74)$$

Pentru acest polinom sunt prezentate în continuare algoritmul și implementarea algoritmului în limbaj C.

### 6.8.1. ALGORITMUL 6.8. FUNCȚII DE DOUĂ VARIABLE

*{Variabile*

$n, m$ : întregi, numărul de puncte pe  $Ox$  și  $Oy$ ;

$x$ : vectorul coordonatelor punctelor pe  $Ox$ ;

$y$ : vectorul coordonatelor punctelor pe  $y$ ;

```

z:matricea valorilor funcției;
pcx:coordonata pe Ox a punctului de interpolare;
pcy:coordonata pe Oy a punctului de interpolare;
i,j,k:intregi,contori;
prodx,prody:produse parțiale;
valoare:sume parțiale;
{
    valoare=0;
    pentru i=0 la n
    {
        prodx=1;
        pentru k=0 la n dacă k≠i
            calculează prodx=(pcx-x[k])/(x[i]-x[k]);
            pentru j=0 la m
            {
                prody=1;
                pentru k=0 la m dacă k≠j
                    calculează prody=(pcy-y[k])/(y[j]-y[k]);
                    calculează valoare=z(i,j)*prodx*prody;
            }
        }
    }
    funcția în punctul de interpolare este valoare;
}

```

### 6.8.2. IMPLEMENTAREA ALGORITMULUI 6.8

```

double Lagrange2(int n, int m,
    double x[],
    double y[],
    double z[][NMAX],
    double pcx,
    double pcy
)
{
    int i,j,k;
    double prodx,prody,valoare;
    valoare=0;
    for(i=0;i<=n;i++)
    {
        prodx=1;
        for(k=0;k<=n;k++)if(k!=i)
            prodx*=(pcx-x[k])/(x[i]-x[k]);
        for(j=0;j<=m;j++)
        {

```



```

        prody=1;
        for(k=0;k<=m;k++)
            if(k!=j)
                prody*=(pcy-y[k])/(y[j]-y[k]);
        valoare+=z[i][j]*prodx*prody;
    }
}
return valoare;
}

```

## 6.9. APLICAȚII

1. Se consideră datele experimentale obținute pentru caracteristica rezistenței termice joncțiune termică  $R_{jt} [^{\circ}C/W]$  funcție de lungimea terminalului  $l [inch]$ , pentru o dioda Zener prezentate în tabelul următor:

Tabelul 6.6

$l$ [inch]	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$R_{jt}$ [oC/w]	70	140	175	200	225	250	265	280	290	300

Se cere să se determine puncte între valorile date pentru o reprezentare grafică cât mai precisă.

S-a determinat valoarea funcției pentru  $l=0.45$  inch. S-au obținut pentru rezistența termică joncțiune terminal următoarele valori:

cu metoda lui Lagrange	238.3757 $^{\circ}C/W$ ;
cu metoda lui Newton de speța întâi	238.3770 $^{\circ}C/W$ ;
cu metoda lui Newton de speța a doua	238.3783 $^{\circ}C/W$ ;
cu metoda lui Aitken	238.3757 $^{\circ}C/W$ ;

# 7

## METODE DE OPTIMIZARE\*

Metodele de optimizare se clasifică, în raport cu problema care se pune pentru funcția țintă sau scop, astfel:

1. Metode ce determină expresia analitică a funcției, care aproximează cel mai bine o funcție tabelată dată prin puncte;
2. Metode ce determină diferiți parametri ai funcției scop (țintă), pentru a obține un extrem al funcției.

### 7.1. METODA CELOR MAI MICI PĂTRATE

Presupunem că avem o funcție definită printr-un tabel de valori. Dacă tabelul este obținut în urma unor măsurători fizice, atunci elementele lui pot avea erori. Elementele care diferă mult de celelalte pot fi eliminate. Problema care se pune este să determinăm funcția analitică care aproximează cel mai bine datele din tabel sau curba care trece prin punctele tabelului.

Considerăm că funcția tabelată poate fi aproximată cu funcția:  $\bar{y} = f(x)$ . Deoarece funcția tabelată este cunoscută numai în anumite puncte, în număr de  $m$ , în aceste puncte avem următoarele erori:

$$\begin{cases} e_1 = y_1 - \bar{y}_1 = y_1 - f(x_1) \\ e_2 = y_2 - \bar{y}_2 = y_2 - f(x_2) \\ \text{-----} \\ e_m = y_m - \bar{y}_m = y_m - f(x_m) \end{cases} \quad (7.1)$$

care ar putea fi calculate dacă am cunoaște funcția  $\bar{y} = f(x)$ .

Pentru determinarea funcției, vom anticipa întâi o funcție care să se asemeze foarte mult cu curba realizată cu punctele tabelului. Această funcție poate fi: liniară, hiperbolică, logaritmică, exponențială, geometrică, trigonometrică etc.

Al doilea pas este determinarea funcției din condițiile ca suma abaterilor:

---

\*Bibliografie: [7], [14],[15],[20]

$$e = \sum_{i=1}^m (y_i - \bar{y}_i) \quad (7.2)$$

să fie minimă.

În cazul unei drepte dată prin două puncte, abaterea (7.2) este nulă pentru toate dreptele ce trec prin mijlocul segmentului format de cele două puncte. Ca urmare problema nu este unic determinată. Această problemă se elimină dacă considerăm valoarea absolută a erorilor:

$$|e| = \sum_{i=1}^m |y_i - \bar{y}_i| \quad (7.2a)$$

Inconvenientul în cazul formulei (7.2a) îl prezintă funcția valoare absolută care nu are derivată în punctele în care se anulează funcția. Pentru eliminarea acestor inconveniente s-a trecut la suma pătratelor erorilor, pe care o notăm cu  $E$ .

$$E = \sum_{i=1}^m (y_i - \bar{y}_i)^2 \quad (7.3)$$

Metoda de optimizare a primit denumirea de *metoda celor mai mici pătrate*, conform formulei (7.3) pentru că determină funcția  $\bar{y} = f(x)$  pentru eroarea maximă și mai poartă denumirea și de *regresie* deoarece problema este de deducere a funcției cunoscând valorile ei într-un anumit număr de puncte.

### 7.1.1. REGRESIA LINIARĂ

Se consideră funcția tabelată:

Tabelul 7.1

$x$	$x_1$	$x_2$	$x_3$	$x_4$	.....	$x_m$
$y$	$y_1$	$y_2$	$y_3$	$y_4$	.....	$y_m$

unde  $m$  reprezintă numărul de măsurători sau de valori ale funcției. Se cere să se determine funcția liniară de forma generală

$$y = ax + b \quad (7.4)$$

care să aproximeze cel mai bine funcția tabelată (tabelul 7.1) astfel ca eroarea:

$$E = \sum_{i=1}^m (ax_i + b - y_i)^2 \quad (7.5)$$

să fie minimă. Necunoscutele  $a$  și  $b$  se determină din condiția de minim a lui  $E$  care se realizează pentru anularea derivatelor parțiale în raport cu  $a$  și  $b$  ale lui  $E$ :

$$\begin{cases} \frac{\partial E}{\partial a} = \sum_{i=1}^m 2(ax_i + b - y_i) \cdot (-1) = 0 \\ \frac{\partial E}{\partial b} = \sum_{i=1}^m 2(ax_i + b - y_i) \cdot (-x_i) = 0 \end{cases} \quad (7.6)$$

Rezultă sistemul în necunoscutele  $a$  și  $b$  :

$$\begin{cases} m \cdot b + \left( \sum_{i=1}^m x_i \right) a = \sum_{i=1}^m y_i \\ \left( \sum_{i=1}^m x_i \right) b + \left( \sum_{i=1}^m x_i^2 \right) a = \sum_{i=1}^m x_i y_i \end{cases} \quad (7.7)$$

Soluțiile sistemului liniar (7.7) în  $a$  și  $b$  sunt:

$$b = \frac{\sum_{i=1}^m x_i^2 \sum_{i=1}^m y_i - \sum_{i=1}^m x_i \sum_{i=1}^m x_i y_i}{m \sum_{i=1}^m x_i^2 - \left( \sum_{i=1}^m x_i \right)^2} \quad (7.8)$$

$$a = \frac{m \sum_{i=1}^m x_i y_i - \sum_{i=1}^m x_i \sum_{i=1}^m y_i}{m \sum_{i=1}^m x_i^2 - \left( \sum_{i=1}^m x_i \right)^2} \quad (7.9)$$

Numitorii expresiilor lui  $a$  și  $b$  se anulează numai dacă  $x_i$  sunt identice, caz exclus. Înlocuind valorile lui  $a$  și  $b$  în ecuația (7.4) avem funcția liniară care aproximează cel mai bine funcția tabelată dată (7.1).

#### 7.1.1.1. Algoritm 7.1. Regresia liniară

```
{Variabile
x:abscisele funcției,vector;
y:ordonatele funcției,vector;
m: numărul experiențelor, întreg;
sx: variabila pentru suma absciselor, real;
sy: variabila pentru suma ordonatelor, real;
sxx:variabila pentru sumele de forma x2, real;
sxy: variabila pentru sumele de forma xy, real;
a:coeficientul lui x, real;
b: termenul liber, real;
i: contor, întreg;
{
  sx = 0; sy = 0; sxy = 0; sxx = 0;
  pentru i = 1... m
  {
    calculează sx = sx + xi;
    calculează sy = sy + yi;
    calculează sxy = sxy + xi * yi;
    calculează sxx = sxx + xi * xi;
  }
}
```

$$\text{calculează } a = \frac{m \cdot s_{xy} - s_x \cdot s_y}{m \cdot s_{xx} - s_x \cdot s_x};$$

$$\text{calculează } b = \frac{s_{xx} \cdot s_y - s_x \cdot s_{xy}}{m \cdot s_{xx} - s_x \cdot s_x};$$

}

Funcția este  $y = ax + b$ ;

}

### 7.1.1.2. Implementarea algoritmului 7.1

```
/*Funcția ce implementează regresia liniară */
void RegLin(int nrp,
            double x[],
            double y[],
            double *coef1,
            double *coef2)
{
    double sx=0,sy=0,sxx=0,sxy=0;
    int i;
    for (i=1;i<=nrp;i++)
    {
        sx+=x[i];
        sy+=y[i];
        sxy+=x[i]*y[i];
        sxx+=x[i]*x[i];
    }
    *coef1=(nrp*sxy-sx*sy)/(nrp*sxx-sx*sx);
    *coef2=(sy*sxx-sx*sxy)/(nrp*sxx-sx*sx);
}
```

### 7.1.2. REGRESIA POLINOMIALĂ

Dacă regresia liniară nu este satisfăcătoare se caută o funcție polinom de un anumit grad (2, 3, 4...n).

Fie funcția dată în tabelul (7.1). Considerăm că reprezentarea grafică a acestei funcții se aseamănă foarte mult cu curba unui polinom de gradul  $n$  de forma:

$$y = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0 \quad (7.10)$$

Notăm cu  $y_i$  expresia:

$$y_i = a_n x_i^n + a_{n-1} x_i^{n-1} + \dots + a_1 x_i + a_0 \quad (7.11)$$

Se minimizează funcția țintă:

$$E = \sum_{i=1}^m \left( y_i - a_n x_i^n - a_{n-1} x_i^{n-1} - \dots - a_1 x_i - a_0 \right)^2 \quad (7.12)$$

unde  $m$  reprezintă numărul de valori ale funcției tabelate.

Necunoscutele sunt:  $a_n, a_{n-1}, \dots, a_0$ .

(7.13)

Se derivează în raport cu necunoscutele (7.13) și se obține sistemul:

$$\left\{ \begin{array}{l} m a_0 + \left( \sum_{i=1}^m x_i \right) a_1 + \left( \sum_{i=1}^m x_i^2 \right) a_2 + \dots + \left( \sum_{i=1}^m x_i^k \right) a_k + \dots + \left( \sum_{i=1}^m x_i^n \right) a_n = \sum y_i \\ \left( \sum_{i=1}^m x_i \right) a_0 + \left( \sum_{i=1}^m x_i^2 \right) a_1 + \left( \sum_{i=1}^m x_i^3 \right) a_2 + \dots + \left( \sum_{i=1}^m x_i^{k+1} \right) a_k + \dots + \left( \sum_{i=1}^m x_i^{n+1} \right) a_n = \sum x_i y_i \\ \hline \left( \sum_{i=1}^m x_i^p \right) a_0 + \left( \sum_{i=1}^m x_i^{p+1} \right) a_1 + \left( \sum_{i=1}^m x_i^{p+2} \right) a_2 + \dots + \left( \sum_{i=1}^m x_i^{p+k} \right) a_k + \dots + \left( \sum_{i=1}^m x_i^{p+n} \right) a_n = \sum x_i^p y_i \\ \hline \left( \sum_{i=1}^m x_i^n \right) a_0 + \left( \sum_{i=1}^m x_i^{n+1} \right) a_1 + \left( \sum_{i=1}^m x_i^{n+2} \right) a_2 + \dots + \left( \sum_{i=1}^m x_i^{n+k} \right) a_k + \dots + \left( \sum_{i=1}^m x_i^{2n} \right) a_n = \sum x_i^n y_i \end{array} \right. \quad (7.14)$$

Sistemul obținut este un sistem liniar în necunoscutele  $a_0, a_1, a_2, \dots, a_n$  și se rezolvă cu una dintre metodele cunoscute în capitolul 3.

#### 7.1.2.1. Algoritmul 7.2. Metoda polinomială

{ Variabile

$X$ : abscisele funcției, vector;

$Y$ : ordonatele funcției, vector;

$m$ : numărul de puncte cunoscute ale funcției, întreg;

$n$ : gradul polinomului, întreg;

{

    pentru  $i = 1 \dots n+1$

    {

$B[i] = 0$ ;

        pentru  $j = 1, \dots, n+1$   $A[i, j] = 0$

    }

    pentru  $i = 1 \dots n+1$

    {

        pentru  $k = 1 \dots m$  calculează  $B[i] = B[i] + y_k * \text{pow}(x_k, i-1)$ ;

        pentru  $j = 1, \dots, n+1$

        pentru  $k = 1 \dots m$  calculează  $A[i, j] = A[i, j] + \text{pow}(x_k, i+j-2)$ ;

    }

    Metoda\_Rez\_Sist.( $n+1, A, B, Sol$ ); (\*se rezolvă sistemul\*)

    }

        coeficienții polinomului sunt  $Sol$ ;

}

### 7.1.2.2. Implementarea algoritmului 7.2

```

/* Funcția întoarce:
   0 dacă gradul polinomului este mai mare ca numărul de puncte
   1 în caz de succes
*/
int RegPol(int nrp,
           double xx[],
           double yy[],
           int grdp,
           double coef[])
{
    static double mat[NrMax][NrMax], b[NrMax];
    int i, j, k, putere;
    if (nrp <= grdp) return 0;
    for (i = 1; i <= grdp + 1; i++)
        for (j = 1; j <= grdp + 1; j++)
        {
            mat[i][j] = 0;
            putere = i + j - 2;
            for (k = 1; k <= nrp; k++) mat[i][j] += pow(xx[k], putere);
        }
    for (i = 1; i <= grdp + 1; i++)
    {
        b[i] = 0;
        putere = i - 1;
        for (k = 1; k <= nrp; k++) b[i] = b[i] + yy[k] * pow(xx[k], putere);
    }
    GAUSS(grdp + 1, mat, b, coef);
    return 1;
}

```

### 7.1.3. REGRESIA HIPERBOLICĂ

Fie funcția numerică dată în tabelul 7.1. Reprezentarea grafică prin puncte a acestei funcții se aseamănă foarte bine cu o funcție hiperbolică. Se pune problema determinării funcției hiperbolice

$$y = \frac{1}{ax+b} \quad (7.15)$$

care aproximează cel mai bine funcția numerică. Pentru comoditatea calculului se inversează funcția hiperbolică și se construiește funcția eroare:

$$E = \sum_{i=1}^m \left( \frac{1}{y_i} - ax_i - b \right)^2 \quad (7.16)$$

Prin derivare în raport cu  $a$  și după aceea cu  $b$  se obține sistemul liniar:

$$\begin{cases} mb + \left( \sum_{i=1}^m x_i \right) a = \sum_{i=1}^m \frac{1}{y_i} \\ \left( \sum_{i=1}^m x_i \right) b + \left( \sum_{i=1}^m x_i^2 \right) a = \sum_{i=1}^m \frac{x_i}{y_i} \end{cases} \quad (7.17)$$

Soluțiile sistemului (7.17) sunt:

$$a = \frac{\sum_{i=1}^m x_i \sum_{i=1}^m \frac{1}{y_i} - m \sum_{i=1}^m \frac{x_i}{y_i}}{\left( \sum_{i=1}^m x_i \right)^2 - m \sum_{i=1}^m x_i^2} \quad (7.18)$$

$$b = \frac{\sum_{i=1}^m \frac{1}{y_i} \sum_{i=1}^m x_i^2 - \sum_{i=1}^m \frac{x_i}{y_i} \sum_{i=1}^m x_i}{m \sum_{i=1}^m x_i^2 - \left( \sum_{i=1}^m x_i \right)^2} \quad (7.19)$$

Înlocuim valorile lui  $a$  și  $b$  în funcția hiperbolică (7.15) și se obține funcția care aproximează cel mai bine funcția numerică din tabelul 7.1.

### 7.1.3.1. Algoritm 7.3. Regresia hiperbolică

*{Variabile*  
 $x$ : abscisele funcției numerice, real;  
 $y$ : ordonatele funcției numerice, real;  
 $m$ : numărul experiențelor, întreg;  
 $s_x$ : variabila ce conține sumele absciselor, real;  
 $s_y$ : variabila ce conține suma inverselor ordonatelor, real;  
 $s_{xx}$ : variabila ce conține pătratul absciselor, real;  
 $s_{xy}$ : variabila ce conține suma raportului dintre abscise și ordonate, real;  
 $a, b$ : coeficienții funcției hiperbolice, real;  
 $i$ : contor, întreg;  
{  
 $s_x = 0$ ;  $s_y = 0$ ;  $s_{xy} = 0$ ;  $s_{xx} = 0$ ;  
pentru  $i = 1 \dots m$   
{  
calculează  $s_x = s_x + x_i$ ;  
calculează  $s_y = s_y + \frac{1}{y_i}$ ;  
calculează  $s_{xy} = s_{xy} + \frac{x_i}{y_i}$ ;  
calculează  $s_{xx} = s_{xx} + x_i * x_i$ ;



```

}
calculează  $a = \frac{s_x \cdot s_y - m \cdot s_{xy}}{m \cdot s_{xx} - s_x \cdot s_x}$ ;
calculează  $b = \frac{s_{xx} \cdot s_y - s_{xy} \cdot s_x}{m \cdot s_{xx} - s_x \cdot s_x}$ ;

Funcția este  $y = \frac{1}{ax+b}$ ;
}

```

### 7.1.3.2. Implementarea algoritmului 7.3

```

/* Regresia hiperbolică */
Programul principal poate fi realizat
de către cititor ca un exercițiu. În continuare este dată funcția:
/* Funcția ce implementează regresia hiperbolică */
void RegHip(int nrp,
             double *x,
             double *y,
             double *coef1,
             double *coef2)
{
    double sx=0,sy=0,sxx=0,sxy=0;
    int i;
    for (i=1;i<=nrp;i++)
    {
        sx+=x[i];
        sy+=1/y[i];
        sxy+=x[i]/y[i];
        sxx+=x[i]*x[i];
    }

    *coef1=(nrp*sxy-sx*sy)/(nrp*sxx-sx*sx);
    *coef2=(sy*sxx-sx*sxy)/(nrp*sxx-sx*sx);
}

```

### 7.1.4. REGRESIA EXPONENȚIALĂ

Se constată că funcția numerică din tabelul 7.1 reprezentată grafic, se aseamănă foarte mult cu o exponențială. Considerăm funcția exponențială de forma generală:

$$y = a \cdot b^x \quad (7.20)$$

în care  $a$  și  $b$  sunt constante pozitive.

Pentru un calcul comod al constantelor  $a$  și  $b$  se logaritmează funcția (7.20)

$$y = \ln a + x \ln b \quad (7.21)$$

și se determină constantele astfel ca eroarea:

$$E = \sum_{i=1}^m (\ln a + x_i \ln b - \ln y_i)^2 \quad (7.22)$$

să fie minimă.

Prin derivarea parțială a funcției  $E$  în raport cu  $a$  și  $b$  se obține următorul sistem:

$$\begin{cases} m \ln a + \left( \sum_{i=1}^m x_i \right) \ln b = \sum_{i=1}^m \ln y_i \\ \left( \sum_{i=1}^m x_i \right) \ln a + \left( \sum_{i=1}^m x_i^2 \right) \ln b = \sum_{i=1}^m x_i \ln y_i \end{cases} \quad (7.23)$$

Soluțiile sistemului obținut (7.23) sunt:

$$a = \exp \left\{ \frac{\left[ \sum_{i=1}^m \ln y_i \sum_{i=1}^m x_i^2 - \sum_{i=1}^m x_i \sum_{i=1}^m \ln y_i x_i \right]}{\left[ m \sum_{i=1}^m x_i^2 - \left( \sum_{i=1}^m x_i \right)^2 \right]} \right\} \quad (7.24)$$

$$b = \exp \left\{ \frac{\left[ m \sum_{i=1}^m x_i \ln y_i - \sum_{i=1}^m x_i \sum_{i=1}^m \ln y_i \right]}{\left[ m \sum_{i=1}^m x_i^2 - \left( \sum_{i=1}^m x_i \right)^2 \right]} \right\} \quad (7.25)$$

#### 7.1.4.1. Algoritm 7.4. Regresia exponențială

*{Variabile*

$x$ : abscisele funcției numerice, real;

$y$ : ordonatele funcției numerice, real;

$m$ : numărul absciselor funcției, întreg;

$s_x$ : variabilă a sumei absciselor, real;

$s_{xx}$ : variabilă a sumei pătratelor absciselor, real;

$s_{\ln y}$ : variabilă a sumei logaritmilor din ordonate, real;

$s_{x \ln y}$ : variabilă a sumei absciselor înmulțite cu logaritmul ordonatelor, real;

{

$s_x = 0$ ;  $s_{\ln y} = 0$ ;  $s_{x \ln y} = 0$ ;  $s_{xx} = 0$ ;

pentru  $i = 1 \dots m$

{

calculează  $s_x = s_x + x_i$ ;

```

    calculează  $s_{xx} = s_{xx} + x_i * x_i$ ;
    calculează  $s_{lny} = s_{lny} + \ln y_i$ ;
    calculează  $s_{xln y} = s_{xln y} + x_i \ln y_i$ ;
}

calculează  $a = \exp\left\{\frac{s_{ln y} \cdot s_{xx} - s_x \cdot s_{xln y}}{m \cdot s_{xx} - s_x \cdot s_x}\right\}$ ;

calculează  $b = \exp\left\{\frac{ms_{xln y} - s_{xln y}}{m \cdot s_{xx} - s_x \cdot s_x}\right\}$ ;

    Funcția este  $y = ab^x$  ;
}

```

#### 7.1.4.2. Implementarea algoritmului 7.4

```

/* (Regresia exponențială )*/
/*Funcția ce implementează regresia exponențială */
void RegExp(int nrp,
            double x[],
            double y[],
            double *coef1,
            double *coef2)
{
    double sx=0, sy=0, sxx=0, sxy=0;
    int i;
    for (i=1; i<=nrp; i++)
    {
        sx+=x[i];
        sy+=log(y[i]);
        sxy+=x[i]*log(y[i]);
        sxx+=x[i]*x[i];
    }
    *coef2=exp((nrp*sxy-sx*sy)/(nrp*sxx-sx*sx));
    *coef1=exp((sy*sxx-sx*sxy)/(nrp*sxx-sx*sx));
}

```

#### 7.1.5. REGRESIA GEOMETRICĂ

Dacă funcția numerică din tabelul 7.1 se aseamănă cu o funcție de tip geometric, vom căuta să determinăm funcția de forma:

$$y = ax^b \quad (7.26)$$

care să aproximeze cel mai bine funcția numerică.

Se logaritmează cel mai bine funcția (7.26) și se construiește funcția:

$$E = \sum_{i=1}^m (\ln a + b \ln x_i - \ln y_i)^2 \quad (7.27)$$

Se calculează valorile lui  $a$  și  $b$  astfel ca funcția  $E$  să fie minimă. Prin derivarea parțială în raport cu  $a$  și  $b$  a expresiei  $E$  și egalarea derivatelor cu zero se obține următorul sistem în necunoscutele  $a$  și  $b$ :

$$\begin{cases} m \ln a + b \left( \sum_{i=1}^m \ln x_i \right) = \sum_{i=1}^m \ln y_i \\ \left( \sum_{i=1}^m \ln x_i \right) \ln a + b \sum_{i=1}^m (\ln x_i)^2 = \sum_{i=1}^m \ln x_i \ln y_i \end{cases} \quad (7.28)$$

Rezolvând sistemul se obțin următoarele valori pentru  $a$  și  $b$ :

$$a = \exp \frac{\left( \sum_{i=1}^m \ln(y_i) \right) \left( \sum_{i=1}^m \ln^2(x_i) \right) - \left( \sum_{i=1}^m \ln(x_i) \right) \left( \sum_{i=1}^m \ln(x_i) \ln(y_i) \right)}{m \left( \sum_{i=1}^m \ln^2(x_i) \right) - \left( \sum_{i=1}^m \ln(x_i) \right)^2} \quad (7.29)$$

$$b = \frac{m \left( \sum_{i=1}^m \ln(y_i) \ln(x_i) \right) - \left( \sum_{i=1}^m \ln(x_i) \right) \left( \sum_{i=1}^m \ln(y_i) \right)}{m \left( \sum_{i=1}^m \ln^2(x_i) \right) - \left( \sum_{i=1}^m \ln(x_i) \right)^2} \quad (7.30)$$

Cu aceste valori funcția  $y = ax^b$  aproximează cel mai bine funcția tabelată (7.1) care se aseamănă cel mai bine cu o funcție de tip geometric.

### 7.1.5.1. Algoritmul 7.5. Regresia geometrică

*{ Variabile*

$x$ : abscisele funcției numerice, vector;

$y$ : ordonatele funcției numerice, vector;

$m$ : numărul de abscise, întreg;

$s_{\ln x}$ : variabilă a sumei logaritmilor absciselor, real;

$s_{\ln y}$ : variabilă a sumei logaritmilor ordonatelor, real;

$s_{\ln x \ln y}$ : variabilă sumei produsului logaritmilor absciselor și a logaritmilor ordonatelor, real;

$s_{\ln x \ln x}$ : variabila sumei pătratelor logaritmilor absciselor, real;

$a, b$ : coeficienții, real;

$i$ : contor, întreg;

{

$s_{\ln x} = 0$ ;  $s_{\ln y} = 0$ ;  $s_{\ln x \ln y} = 0$ ;  $s_{\ln x \ln x} = 0$ ;

pentru  $i = 1 \dots m$

{

calculează  $s_{\ln x} = s_{\ln x} + \ln(x_i)$ ;

calculează  $s_{\ln x \ln x} = s_{\ln x \ln x} + \ln(x_i) * \ln(x_i)$ ;

calculează  $s_{\ln y} = s_{\ln y} + \ln y_i$ ;

```

    calculează  $s_{\ln x \ln y} = s_{\ln x \ln y} + \ln(x_i) * \ln(y_i);$ 
}

    calculează  $a = \exp \left\{ \frac{s_{\ln y} \cdot s_{\ln x \ln x} - s_{\ln x} \cdot s_{\ln x \ln y}}{m \cdot s_{\ln x \ln x} - s_{\ln x} \cdot s_{\ln x}} \right\};$ 

    calculează  $b = \frac{m \cdot s_{\ln x \ln y} - s_{\ln x} \cdot s_{\ln y}}{m \cdot s_{\ln x \ln x} - s_{\ln x} \cdot s_{\ln x}};$ 

    Funcția este  $y = ax^b$  ;
}

```

### 7.1.5.2. Implementarea algoritmului 7.5

```

/*Funcția ce implementează regresia geometrică */
void RegGeo(int nrp,
            double x[],
            double y[],
            double *coef1,
            double *coef2)
{
    double sx=0, sy=0, sxx=0, sxy=0;
    int i;
    for (i=1; i<=nrp; i++)
    {
        sx+=log(x[i]);
        sy+=log(y[i]);
        sxy+=log(x[i])*log(y[i]);
        sxx+=log(x[i])*log(x[i]);
    }
    *coef2=(nrp*sxy-sx*sy)/(nrp*sxx-sx*sx);
    *coef1=exp((sy*sxx-sx*sxy)/(nrp*sxx-sx*sx));
}

```

### 7.1.6. REGRESIA TRIGONOMETRICĂ

Se consideră funcția numerică dată în tabelul 7.1 și funcția trigonometrică de forma:

$$y = a + b \cos \omega x \quad (7.31)$$

care aproximează cel mai bine funcția numerică.

Se determină constantele  $a$  și  $b$  astfel ca funcția de  $a$  și  $b$

$$E = \sum_{i=1}^m (y_i - a - b \cos \omega x_i)^2 \quad (7.32)$$

să fie minimă.

Prin egalarea cu zero a derivatelor parțiale în raport cu  $a$  și  $b$  ale funcției  $E$  se obține următorul sistem în necunoscutele  $a$  și  $b$ :

$$\begin{cases} ma + \left( \sum_{i=1}^m \cos \omega x_i \right) b = \sum_{i=1}^m y_i \\ \left( \sum_{i=1}^m \cos \omega x_i \right) a + \left( \sum_{i=1}^m \cos^2 \omega x_i \right) b = \sum_{i=1}^m y_i \cos \omega x_i \end{cases} \quad (7.33)$$

Soluțiile sistemului (7.32) sunt:

$$a = \frac{\left( \sum_{i=1}^m y_i \right) \left( \sum_{i=1}^m \cos^2 \omega x_i \right) - \left( \sum_{i=1}^m \cos \omega x_i \right) \left( \sum_{i=1}^m y_i \cos \omega x_i \right)}{m \left( \sum_{i=1}^m \cos^2 \omega x_i \right) - \left( \sum_{i=1}^m \cos \omega x_i \right)^2} \quad (7.34)$$

$$b = \frac{m \left( \sum_{i=1}^m y_i \cos \omega x_i \right) - \left( \sum_{i=1}^m \cos \omega x_i \right) \left( \sum_{i=1}^m y_i \right)}{m \left( \sum_{i=1}^m \cos^2 \omega x_i \right) - \left( \sum_{i=1}^m \cos \omega x_i \right)^2} \quad (7.35)$$

$\omega = 2 \cdot \pi \cdot f = \frac{2 \cdot \pi}{T}$  unde  $f$  -frecvența,  $\omega$  - pulsația, iar  $T$  -perioada.  $\omega$  se stabilește funcție de periodicitatea funcției numerice date.

#### 7.1.6.1. Algoritmul 7.6. Regresia trigonometrică

*{ Variabile*

$x$ : abscisele funcției numerice, vector;

$y$ : ordonatele funcției numerice, vector;

$m$ : numărul absciselor funcției numerice, întreg;

$\omega$ : pulsația, real;

$s_y$ : suma ordonatelor funcției numerice, real;

$s_{\cos 2x}$ : suma pătratelor cosinusurilor din abscise, real;

$s_{\cos x}$ : suma cosinusurilor din abscise, real;

$s_{y \cos x}$ : suma produsului ordonatelor cu cosinusul absciselor corespunzătoare, real;

$a, b$ : coeficienții funcției, real;

$i$ : contor, întreg;

*{*

$s_y = 0; s_{\cos 2x} = 0; s_{\cos x} = 0; s_{y \cos x} = 0;$

pentru  $i = 1 \dots m$

*{*

calculează  $s_y = s_y + y_i;$

calculează  $s_{\cos 2x} = s_{\cos 2x} + \cos(\omega x_i) * \cos(\omega x_i);$

calculează  $s_{\cos x} = s_{\cos x} + \cos(\omega x_i);$

```

    calculează  $s_{y\cos x} = s_{y\cos x} + y_i \cdot \cos(\omega x_i);$ 
}

    calculează  $a = \frac{s_y \cdot s_{\cos 2x} - s_{\cos x} \cdot s_{y\cos x}}{m \cdot s_{\cos 2x} - s_{\cos x} \cdot s_{\cos x}};$ 

    calculează  $b = \frac{m \cdot s_{y\cos x} - s_{\cos x} \cdot s_y}{m \cdot s_{\cos 2x} - s_{\cos x} \cdot s_{\cos x}};$ 
}

    Funcția este  $f = a + b \cos(\omega x);$ 
}

```

### 7.1.6.2. Implementarea algoritmului 7.6

```

void reg_trig( int n,
               double x[],
               double y[],
               double w,
               double *coef1,
               double *coef2)
{
    int i;
    double sx=0, sy=0, sxx=0, sxy=0;
    for(i=1; i<=n; i++)
    {
        sx+=cos(w*x[i]);
        sy+=y[i];
        sxy+=y[i]*cos(w*x[i]);
        sxx+=cos(w*x[i])*cos(w*x[i]);
    }
    *coef1=(sy*sxx-sx*sxy)/(n*sxx-sx*sx);
    *coef2=(n*sxy-sx*sy)/(n*sxx-sx*sx);
}

```

### 7.1.7. REGRESIA MULTIPLĂ

Această regresie se referă la funcțiile de mai multe variabile. Considerând o funcție numerică de  $n$  variabile  $y = f(x_1, x_2, \dots, x_n)$  și un tip de funcție analitică de  $n$  variabile care se aseamănă cu cea numerică, se pune problema determinării funcției analitice astfel ca ea să aproximeze cel mai bine funcția numerică. Această funcție se determină prin minimizarea sumei pătratelor erorilor în punctele funcției numerice. Pentru exemplificare considerăm funcția de două variabile numerică  $(x, y, z)$  unde  $z = g(x, y)$ . Suprafața punctelor  $z$  se poate asemana cu un plan, hiperboloid de rotație, elipsoid de rotație, cilindru, con etc.

Vom considera funcția numerică dată în figura 7.1

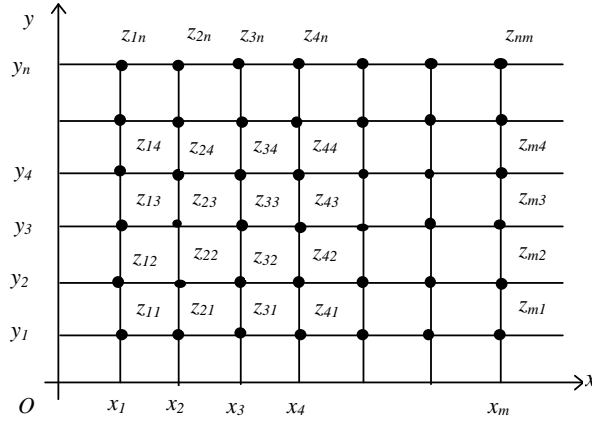


Fig. 7.1. Reprezentarea funcției multiple.

Considerăm că punctele  $z_1, z_2, \dots, z_n$  se află aproximativ pe un plan:

$$z = Ax + By + C \quad (7.36)$$

Se pune problema determinării acestui plan, adică a valorilor constantelor  $A, B, C$  astfel ca planul să aproximeze cel mai bine funcția numerică dată în tabelul 7.2. Se construiește funcția sumelor pătratelor erorilor în punctele rețelei:

$$E = \sum_{i=1}^m \sum_{j=1}^n \left( z_{ij} - Ax_i - By_j - C \right)^2 \quad (7.37)$$

Prin egalarea derivatelor parțiale ale funcției  $E$  în raport cu necunoscutele  $A, B, C$ , rezultă un sistem liniar funcție de aceste necunoscute:

$$\begin{cases} mnC + \left( \sum_{i=1}^m x_i \right) nA + \left( \sum_{j=1}^n y_j \right) mB = \sum_{i=1}^m \sum_{j=1}^n z_{ij} \\ \left( \sum_{i=1}^m x_i \right) nC + \left( \sum_{i=1}^m x_i^2 \right) nA + \left( \sum_{i=1}^m \sum_{j=1}^n x_i y_j \right) B = \sum_{i=1}^m \sum_{j=1}^n x_i z_{ij} \\ \left( \sum_{j=1}^n y_j \right) mC + \left( \sum_{i=1}^m \sum_{j=1}^n x_i y_j \right) A + \left( \sum_{j=1}^n y_j^2 \right) mB = \sum_{i=1}^m \sum_{j=1}^n y_j z_{ij} \end{cases} \quad (7.38)$$

Utilizând una dintre metodele numerice de rezolvare a sistemelor liniare din capitolul 3 se determină  $A, B, C$ , deci planul căutat.

### 7.1.7.1. Algoritm 7.7. Regresia multiplă



```

{Variabile
x: argumentele pe axa Ox a funcției numerice, vector;
y: argumentele pe axa Oy a funcției numerice, vector;
z: valorile funcției numerice, matrice;
m: numărul de argumente pe Ox, întreg ;
n: numărul de argumente pe Oy, întreg;
sx: sumele argumentelor de pe Ox, real;
sy: suma parțială a argumentelor de pe Oy, real;
sxx: suma parțială a pătratelor argumentelor x, real;
sxy: suma parțială a produselor argumentelor, real;
syy: suma parțială a pătratelor argumentelor y, real;
sz, sxz, syz: sume parțiale ale termenilor liberi ai sistemului;
i: contor, întreg;
{
sx = 0; sy = 0; sxy = 0; sxx = 0; syy = 0; sxz = 0; syz = 0; sz = 0;
pentru i = 1,..., m
    pentru j = 1,..., m
        calculează sx = sx + xi;
        calculează sxx = sxx + xi * xi;
        calculează sy = sy + yi;
        calculează syy = syy + yi * yi;
        calculează sxy = sxy + xi * yi;
        calculează sz = sz + zij; calculează sxz = sxz + xi * zij;
        calculează syz = syz + yj * zij;
    }
{ METODA (n,m, sx, sy, sxy, sxx, syy, SOL ); // rezolvă sistemul
}
Planul este z = Ax + By + C
}

```

#### 7.1.7.2. Implementarea algoritmului 7.7

```

/* Funcția întoarce -1 în caz de eroare 0 altfel */
int reg_mul( int m, /* pe Ox */
             int n, /* pe Oy */
             double x[],
             double y[],
             double z[][NMAX],
             double *coef1,
             double *coef2,
             double *coef3)
{
    int i,j;
    double mat[4][4], tl[4], det, det1, det2, det3;
    for(i=1; i<=3; i++)
    {

```

```

        tl[i]=0;
        for(j=1;j<=3;j++) mat[i][j]=0;
    }
    mat[1][1]=n*m;
    for(i=1;i<=m;i++)
    {
        mat[1][2]+=n*x[i];
        mat[2][2]+=n*x[i]*x[i];
    }
    for(j=1;j<=n;j++)
    {
        mat[1][3]+=m*y[j];
        mat[3][3]+=m*y[j]*y[j];
    }
    for (i=1;i<=m;i++)
        for(j=1;j<=n;j++)
        {
            mat[2][3]+=x[i]*y[j];
            tl[1]+=z[i][j];
            tl[2]+=x[i]*z[i][j];
            tl[3]+=y[j]*z[i][j];
        }
    mat[2][1]=mat[1][2];
    mat[3][1]=mat[1][3];
    mat[3][2]=mat[2][3];
    det=mat[1][1]*mat[2][2]*mat[3][3]+mat[2][1]*mat[3][2]*mat[1][3]
] +
    mat[1][2]*mat[2][3]*mat[3][1]-mat[3][1]*mat[2][2]*mat[1][3]-
    mat[1][1]*mat[3][2]*mat[2][3]-mat[2][1]*mat[1][2]*mat[3][3];
    if (det==0 ) return -1;
    det1=tl[1]*mat[2][2]*mat[3][3]+tl[2]*mat[3][2]*mat[1][3]+
    mat[1][2]*mat[2][3]*tl[3]-tl[3]*mat[2][2]*mat[1][3]-
    tl[1]*mat[3][2]*mat[2][3]-tl[2]*mat[1][2]*mat[3][3];
    det2=mat[1][1]*tl[2]*mat[3][3]+mat[2][1]*tl[3]*mat[1][3]+
    tl[1]*mat[2][3]*mat[3][1]-mat[3][1]*tl[2]*mat[1][3]-
    mat[1][1]*tl[3]*mat[2][3]-mat[2][1]*tl[1]*mat[3][3];
    det3=mat[1][1]*mat[2][2]*tl[3]+mat[2][1]*mat[3][2]*tl[1]+
    mat[1][2]*tl[2]*mat[3][1]-mat[3][1]*mat[2][2]*tl[1]-
    mat[1][1]*mat[3][2]*tl[2]-mat[2][1]*mat[1][2]*tl[3];
    *coef1=det1/det;
    *coef2=det2/det;
    *coef3=det3/det;
    return 0;
}

```

## 7.2. OPTIMIZAREA NELINIARĂ FĂRĂ RESTRICȚII

În cadrul acestui paragraf vor fi studiate două metode pentru aflarea minimumului unei funcții de  $n$  variabile în lipsa relațiilor de restricție. În general, problemele de minimizare sunt cu relații de restricție, dar aceste probleme pot fi reduse la probleme de optimizare fără restricții într-o serie de aplicații.

În condițiile în care nu sunt restricții, funcția țintă

$$F(X) = F(x_1, x_2, \dots, x_n) \quad (7.39)$$

care are derivatele parțiale continue, are un minim local într-un punct  $X_k$  dacă este îndeplinită inegalitatea:

$$F(X_k) \leq F(X) \quad (7.40)$$

pentru orice vector  $X$  din vecinătatea lui  $X_k$ .

Din analiza matematică cunoaștem condiția necesară pentru ca o funcție  $F(X)$  să aibă un minim într-un punct  $X$  din domeniul de definiție al funcției:

- existența și rezolvabilitatea sistemului

$$\frac{\partial F(x)}{\partial x_i} = 0, \quad i = 1, 2, \dots, n; \quad (7.41)$$

- matricea

$$J = \begin{bmatrix} \frac{\partial^2 F(x)}{\partial x_1^2} & \frac{\partial^2 F(x)}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 F(x)}{\partial x_1 \partial x_n} \\ \text{-----} \\ \frac{\partial^2 F(x)}{\partial x_n \partial x_1} & \frac{\partial^2 F(x)}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 F(x)}{\partial x_n^2} \end{bmatrix} \quad (7.42)$$

să fie pozitiv definită.

Dintre metodele de optimizare neliniară fără restricții amintim: *metode aleatoare de căutare*, *metode de căutare unidirecțională* și *metode de gradient*.

*Metodele aleatoare de căutare* au la bază o schimbare aleatoare a componentelor vectorului  $X$ , pentru care se calculează valoarea funcției  $F(X)$  și se compară cu valoarea precedentă a funcției.

*Metodele de căutare unidimensională* constau în modificarea succesivă a componentelor vectorului și compararea valorilor funcției în punctele respective.

*Metodele de gradient* efectuează modificarea simultană a componentelor vectorului  $X$ , în așa fel încât să rezulte o evoluție a funcției  $F(X)$  opusă gradientului. Aceste metode se bazează pe generarea aleatoare a unor numere într-un domeniu prestabilit.

### 7.2.1. METODELE ALEATOARE DE CĂUTARE

Procesul este iterativ și pentru fiecare etapă se generează o succesiune de numere aleatoare având o densitate de probabilitate uniformă în domeniul de variație admis, cu care se modifică componentele vectorului.

Metoda drumului aleator, una dintre metodele des utilizate, constă în modificarea vectorului de poziție pentru noul punct astfel:

$$X_{k+1} = X_k + \alpha \cdot r \quad (7.43)$$

unde  $r$  este vectorul unitate care are direcția aleatoare,  $X_k$  este vectorul de poziție al punctului anterior și  $\alpha$  un scalar. Se calculează valoarea funcției în noul punct și se compară cu valoarea funcției în vechiul punct. Dacă, după un anumit număr de iterații valoarea lui  $F$  nu se micșorează, se reduce scalarul  $\alpha$  până ce valoarea lui devine mai mică decât o eroare de calcul dată,  $er > 0$ .

### 7.2.1.1. Algoritm 7.8. Metoda drumului aleator

```
{ Variabile
X:vectorul de poziție al punctului curent;
alfa:scalarul, real;
er:eroarea de calcul, reală;
sol:soluția optimizării, reală;
i:contor, întreg;
minim:minimul dintre valorile funcțiilor, real;
valcurentă:valoarea minimă curentă a funcției, real;
xcurent[2];valoarea anterioară și curentă a abscisei punctului, real;
maxiter:numărul maxim de iterații, întreg;
{
  maxiter=1000;
  atâta timp cât ( alfa0>eps )
  {contor=0
  minim=F(x0[0],x0[1]);
  calculează
  xcurent[0]=x0[0]+alfa0*r;
  xcurent[1]=x0[1]+alfa0*r;
  val_curenta=F(xcurent[0],xcurent[1]);
  i=i+1;
  }atâta timp cât ((contor<=maxiter) și (valcurenta>minim) );
  dacă (valcurenta<minim)
  {
    minim=valcurenta;
    x0[0]=xcurent[0];
    x0[1]=xcurent[1];
  }
  dacă (contor>maxiter)alfa0=0.001;
}
sol[0]=x0[0];
sol[1]=x0[1];
stop
}
```

### 7.2.1.2. Implementarea algoritmului 7.8

```

int căutare_aleatoare( double (*f)( double, double),
                      double x0[],
                      double gama0,
                      double eps,
                      double rez[])
{
    double minim, val_curenta, x_curent[2];
    int contor, maxiter;
    maxiter=1000;
    while( gama0>eps )
    { contor=0
      minim=f(x0[0],x0[1]);
      do{
        x_curent[0]=x0[0]+gama0*rand(101)/100*pow(-1,rand(101) % 2);
        x_curent[1]=x0[1]+gama0*rand(101)/100*pow(-1,rand(101) % 2);
        val_curenta=f(x_curent[0],x_curent[1]);
        contor++;
      }while ((contor<=maxiter) && (val_curenta>minim) );
      if (val_curenta< -1e10) return 1;
      if(val_curenta<minim)
      { minim=val_curenta;
        x0[0]=x_curent[0];
        x0[1]=x_curent[1];
      }
      if (contor>maxiter)gama0-=0.001;
    }
    rez[0]=x0[0];
    rez[1]=x0[1];
    return 0;
}

```

### 7.2.2. METODA CĂUTĂRII UNIDIMENSIONALE

Această metodă determină minimul unei funcții  $F(X)$  prin modificarea componentelor vectorului  $X$  pe rând, începând cu prima componentă. Se modifică prima componentă atâta timp cât se obține o valoare mai mică a funcției în punctul curent decât în cel precedent. Când această condiție nu mai este îndeplinită se trece la următoarea componentă a vectorului  $X$  și se procedează la fel până se baleiază toate componentele vectorului considerând, în acest caz, că s-a realizat un ciclu. Se poate

începe un nou ciclu luând ca punct inițial minimul obținut. La fiecare iterație se determină un nou punct funcție de precedentul astfel:

$$X_{k+1} = X_k + pas \cdot E_k \quad (7.44)$$

unde  $E_k$  reprezintă vectorul  $E_k = (0, 0, \dots, 1, \dots, 0)$ , cifra 1 fiind pe locul  $k$ ,  $k=0, 1, \dots, n$ , iar  $pas$  reprezintă pasul cu care se modifică componentele. După un ciclu încheiat se trece la un nou ciclu cu vectorul inițial cel pentru care s-a obținut minimul în ciclul încheiat (precedent), luând un nou pas mai mic, obținut din pasul precedent înmulțit cu o rație subunitară. Se continuă iterația până când  $pas < \varepsilon$  sau numărul de iterații depășește un număr maxim dat.  $\varepsilon$  reprezintă eroarea de calcul a punctului de minim.

### 7.2.2.1. Algoritm 7.9. Metoda căutării unidimensionale

```
{Variabile
X:vectorul coordonatelor punctelor;
pas:pasul de modificare al coordonatelor, real;
E: matricea unitate de ordinul n ce are pe orizontală vectorii  $E_k$ ;
k:contor, întreg;
minimcurent:minimul obținut până la pasul respectiv, real;
Maxiter:numărul maxim de iterații, întreg;
t:valoare subunitară cu care se multiplică pasul, real;
p:valoarea inițială a pasului, real;
{Maxiter=1000;
t=0.1;
repetă
pas=p;
minimcurent=F(X[0],X[1]);
pentru k=1 până la n calculează
    X[1]=X[0]+pas* $E_k$ ;
    valcurentă=minimcurent;
    p=p*t;
    până când pas< er;
}
minimul este valcurentă;
}
```

### 7.2.2.2. Implementarea algoritmului 7.9

```
int căutare_unidimensională( double (*f)( double, double),
                           double x0[],
                           double pas,
                           double eps,
                           double rez[])
```

```

{
double min_prec,min_curent,min_ant;
int i,nrp,st,dr;
min_curent=f(x0[0],x0[1]);
do
{
min_ant=min_curent;
for(i=0;i<2;i++)
{
nrp=0;
st=0;
dr=1;
do
{
nrp++;
x0[i]+=-st*pas+dr*pas;
min_prec=min_curent;
min_curent=f(x0[0],x0[1]);
if ( (nrp==1) && (min_curent>min_prec) )
{
x0[i]-=2*pas;
nrp++;
min_curent=f(x0[0],x0[1]);
st=1;
dr=0;
}
}
while ( (min_curent<min_prec) );
x0[i]+=st*pas-dr*pas;
min_curent=f(x0[0],x0[1]);
}
if(min_curent<-1e10) {
rez[0]=x0[0];
rez[1]=x0[1];
return 1;
}
}
while( fabs(min_ant-min_curent)>eps);
rez[0]=x0[0];
rez[1]=x0[1];
return 0;
}

```

### 7.3. APLICAȚII

1. Se dau datele experimentale din tabelul 7.2:

Tabelul 7.2.

$x$	1	2	3	4	5	6	7	8	9
$y$	1.44	1.728	2.0736	2.4883	2.9859	3.5831	4.2998	5.1597	6.191

Analizând rezultatele vedem că funcția are evoluția unei funcții exponențiale. Se aplică programul pentru regresia exponențială și se obține funcția:

$$y = 1.2(1.2)^x$$

2. Se dă funcția:

$$z = x^2 + y^2 - 2x - 4y + 3$$

Să se determine minimul funcției cu metoda drumului aleator, luând punctul de start (15,34) și  $\varepsilon = 0.000001$  și cu metoda căutării unidimensionale luând punctul de start (10,10) și  $\varepsilon = 0.001$ .

Cu metoda drumului aleator s-a obținut minimul în punctul (0.99518, 1.99978), iar minimul funcției  $z_{\min} = -1.99977$ .

Aplicând metoda căutării unidimensionale s-a obținut minimul în punctul (1, 2), iar minimul funcției  $z_{\min} = -2$ .



# 8

## REZOLVAREA NUMERICĂ A ECUAȚIILOR ȘI SISTEMELOR DIFERENȚIALE\*

În acest capitol sunt prezentate metodele de rezolvare a ecuațiilor diferențiale ordinare, a ecuațiilor cu derivate parțiale și a sistemelor de ecuații diferențiale.

### 8.1. REZOLVAREA NUMERICĂ A ECUAȚIILOR DIFERENȚIALE ORDINARE DE ORDINUL ÎNTÂI

Se consideră ecuațiile de tipul:

$$y' = f(x, y) \quad (8.1)$$

cu condiția inițială:

$$y(x_0) = y_0 \quad (8.2)$$

Ecuația diferențială (8.1) definește o curbă în planul  $xOy$ . În fiecare punct al curbei se dă valoarea derivatei funcției de  $x$  și  $y$ . Ecuația este satisfăcută de o familie de curbe, iar condiția (8.2) dă curba din familie care trece printr-un anumit punct din plan  $(x_0, y_0)$ . Soluția analitică a ecuației (8.1) este o expresie a lui  $y$  funcție de  $x$ . Metodele numerice ne ajută să determinăm puncte ale soluției ecuației date.

O soluție numerică a ecuației se poate obține plecând din punctul dat  $(x_0, y_0)$ , știind că dacă înlocuim valorile date, în ecuația diferențială (8.1) se poate obține panta curbei căutate în acest punct. După calculul pantei curbei în punctul  $x_0$  se avansează cu un pas mic pe tangenta în punctul  $(x_0, y_0)$

$$y = f(x_0, y_0)(x - x_0) + y_0 \quad (8.3)$$

Dacă considerăm punctul de coordonate  $(y_1, x_1)$  pe tangentă și pasul de creștere al lui  $x_1$  față de  $x_0$ ,  $h$  avem

$$x_1 = x_0 + h \text{ și } y_1 = f(x_0, y_0)h + y_0 \quad (8.4)$$

deci un nou punct cunoscut  $(x_1, y_1)$ , foarte aproape de curba soluției căutate cu cât  $h$  este mai mic, (fig. 8.1). Procedul de calcul se repetă cu calculul pantei la curbă în punctul  $(x_1, y_1)$  și determinarea unui alt punct pe tangenta la curbă în punctul  $(x_1, y_1)$  cu  $x_2 = x_1 + h$  și  $y_2$  corespunzător. Continuând în acest mod se obține o succesiune de segmente de dreaptă care aproximează curba soluției a ecuației diferențiale.

---

\**Bibliografie* : [6], [7], [9], [12], [17]

Aproximarea curbei soluție cu segmente de dreaptă poate da erori și succesiunea de segmente de dreaptă poate să se depărteze considerabil de curba soluției. Această problemă reprezintă problema stabilității procesului de rezolvare a ecuației diferențiale

și trebuie să-i acordăm o atenție deosebită. Trebuie implementată o metodă prin care, în loc să aproximăm soluția printr-o succesiune de drepte, să se ia în considerare și curba soluție adevărată.

Există două tipuri de metode:

a) *Metode directe* în care soluția nu se iterează și se folosește numai o informație asupra curbei într-un punct. Astfel de metode constau în rezolvarea ecuațiilor prin serii Taylor, care însă nu sunt practice. Metode practice Runge-Kutta pretind un mare număr de evaluări ale funcțiilor, iar eroarea este dificil de evaluat.

b) *Metode indirecte* în care se poate estima punctul următor de pe curbă folosind mai puține estimări ale funcției, dar care necesită iterații pentru a ajunge la o valoare suficient de precisă. Aceste metode sunt denumite predictor-corrector și prin aplicarea lor se poate obține o estimare a erorii.

### 8.1.1. METODA SERIILOR LUI TAYLOR

Această metodă asigură soluția oricărei ecuații diferențiale, dar are o aplicabilitate redusă din cauza dificultăților de rezolvare. Totuși, această metodă servește pentru compararea cu metodele practice pentru a le stabili ordinul metodei.

Dezvoltăm pe  $y(x)$  în vecinătatea lui  $x = x_m$ ,

$$y(x) = y_m + y'_m(x-x_m) + \frac{y''_m}{2!}(x-x_m)^2 + \dots + \frac{y^{(j)}_m}{j!}(x-x_m)^j + \frac{y^{(j+1)}_m(\xi)}{(j+1)!}(x-x_m)^{j+1} \quad (8.5)$$

unde  $y^{(j)}_m$  este derivata de ordinul  $j$  a lui  $y(x)$  în punctul  $x = x_m$ , iar  $x_m < \xi < x$

Utilizând relația  $x = x_m + h$  obținem:

$$y_{m+1} = y_m + hy'_m + \frac{h^2}{2}y''_m + \dots + \frac{y^{(j)}_m}{j!}h^j + \frac{y^{(j+1)}_m(\xi)}{(j+1)!}h^{j+1} \quad (8.6)$$

Ultimul termen al relației (8.6) reprezintă eroarea de trunchiere. Aproximația va fi cu atât mai bună cu cât  $j$  este mai mare.

$$y'_m = f(x_m, y_m) \quad (8.7)$$

Derivând funcția  $y' = f(x, y)$  obținem:

$$y'' = \frac{\partial f(x, y)}{\partial x} + \frac{\partial f(x, y)}{\partial y} \cdot \frac{dy}{dx} = f_x + f_y \cdot f \quad (8.8)$$

$$\text{unde} \quad f_x = \frac{\partial f}{\partial x} \quad \text{și} \quad f_y = \frac{\partial f}{\partial y} \quad (8.9)$$

Pentru  $x = x_m$  rezultă:

$$y''_m = f_x + f_y \cdot f \quad (8.10)$$

Pentru  $j = 2$  rezultă:

$$y_{m+1} = y_m + hy'_m + \frac{h^2}{2}y''_m + \frac{y'''_m(\xi)}{6}h^3 \quad (8.11)$$

sau

$$y_{m+1} = y_m + h \left[ f + \frac{h}{2} (f_x + f \cdot f_y) \right] \quad (8.12)$$

cu eroarea de trunchiere

$$e_T = \frac{y'''(\xi)}{6} h^3, \quad x_m < \xi < x \quad (8.13)$$

Punând  $m = 0$  în ecuația (8.12) se obține pentru  $x_1 = x_0 + h$  valoarea soluției  $y_1$  deci punctul  $(x_1, y_1)$ . Pentru  $m = 1$  rezultă punctul  $(x_2, y_2)$ . Continuând, se obțin punctele soluției ecuației diferențiale, cu erorile de trunchiere corespunzătoare, care se măresc în timp. Această metodă este directă deoarece pentru calculul lui  $y_{m+1}$  sunt necesare numai valorile lui  $y_m$  și  $x_m$ . Pentru o eroare mai mică trebuie să mărim ordinul termenilor utilizați în dezvoltarea Taylor. Dacă dorim  $y_m'''$  acesta are expresia:

$$y_m''' = f_{xx} + 2ff_{xy} + f^2 f_{yy} + f_x f_y + f f_y^2 \quad (8.14)$$

unde s-a notat:  $f_{xx} = \frac{\partial^2 f}{\partial x^2}$ ,  $f_{xy} = \frac{\partial^2 f}{\partial x \partial y}$  și  $f_{yy} = \frac{\partial^2 f}{\partial y^2}$

Complexitatea acestor derivate face practic inutilizabilă această metodă.

### 8.1.2. METODELE RUNGE-KUTTA

Metodele Runge-Kutta se caracterizează prin următoarele proprietăți:

1. Sunt *metode directe*, deci pentru calculul lui  $y_{m+1}$  sunt necesare informațiile de la punctul precedent  $x_m$  și  $y_m$ .
2. Sunt echivalente cu seriile Taylor până la termenii  $h^p$ , unde  $p$  este diferit pentru metode diferite și se numește ordinul metodei.
3. Nu necesită evaluarea nici unei derivate a funcției  $f$ , ci numai valoarea funcției. Proprietatea a treia a metodelor Runge-Kutta le recomandă pentru utilizarea lor practică. Se pune problema evaluării funcției soluție a ecuației diferențiale (8.1) în mai multe puncte. Geometric, problema constă în determinarea punctului  $(x_{m+1}, y_{m+1})$  cunoscând punctul  $(x_m, y_m)$  și că  $x_{m+1} = x_m + h$ . În punctul  $(x_m, y_m)$  se duce tangenta la curbă deoarece știm panta curbei în acest punct.

$$y = y_m + y'_m (x - x_m) \quad (8.15)$$

$$\text{unde} \quad y'_m = f(x_m, y_m) \quad (8.16)$$

$$\text{și} \quad x_{m+1} = x_m + h \quad (8.17)$$

Înlocuind în (8.15) relațiile (8.16) și (8.17) rezultă:

$$x_{m+1} = x_m + h \quad (8.18)$$

iar eroarea este dată de segmentul  $e$ .

Dacă în dezvoltarea lui Taylor luăm  $j = 1$  rezultă:

$$y_{m+1} = y_m + h y'_m + \frac{y''(\xi)}{2} h^2 \quad (8.19)$$

unde

$$x_m < \xi < x_{m+1} \quad (8.20)$$

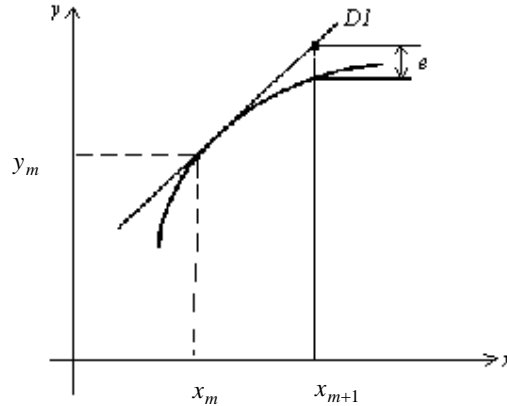


Fig.8.1. Determinarea grafică a soluției ecuației diferențiale.

Comparând relațiile (8.19) și (8.20) rezultă eroarea de trunchiere pentru metoda aplicată:

$$e_T = \frac{y''(\xi)}{2} h^2 \quad (8.21)$$

Formula (8.19) reprezintă metoda lui Euler de rezolvare a ecuațiilor diferențiale de ordinul întâi. Metoda este veche, are eroare de trunchiere relativ mare și eroarea de rotunjire sau trunchiere se mărește odată cu mărirea lui  $x$ .

### 8.1.2.1. Algoritm 8.1. Metoda lui Euler

{ Variabile

$x_0$ : abscisa punctului de start, real;

$y_0$ : ordonata punctului de start, real;

$h$ : pasul între abscisele punctelor de calcul ale soluției, real;

$y$ : ordonatele soluției numerice, vector;

$x$ : abscisele soluției numerice, vector;

$n$ : numărul de puncte în care se calculează soluția, întreg;

{

$x[0] = x_0$ ;  $y[0] = y_0$ ;  $h = \text{const}$ ;

  pentru  $i = 1, 2, \dots, n$

  calculează  $x[i] = x_0 + i * h$ ;

  calculează

$y[i+1] = y[i] + h * f(x[i], y[i])$ ;

Soluțiile numerice sunt  $x[i]$ ;  $y[i]$ ;  $i = 0, 1, 2, \dots, n$ ;

}

}

**8.1.2.2. Implementarea algoritmului 8.1**

*/\* Funcția care implementează algoritmul de rezolvare a ecuațiilor diferențiale ordinare prin metoda Euler.*

*\*/*

```
void EULERED( double(*f)(double x, double y),
              double x0,
              double y0,
              double pas,
              int nrp,
              double sol[])
```

```
{
  int i;
  sol[0]=y0;
  for(i=1;i<=nrp;i++)
    sol[i]=sol[i-1]+pas*f(x0+(i-1)*pas,sol[i-1]);
}
```

**8.1.3. METODELE RUNGE-KUTTA DE ORDINUL DOI**

Aceste metode se numesc de ordinul doi deoarece sunt echivalente cu dezvoltarea în serie Taylor până la termenii de ordinul doi.

**8.1.3.1. Metoda lui Euler îmbunătățită**

Această metodă face parte din acest grup de metode și utilizează media pantelor din punctele  $(x_m, y_m)$  și  $(x_{m+1}, y_{m+1})$  unde  $x_{m+1} = x_m + h$  iar  $y_{m+1} = y_m + hy'_m$  ordonată obținută din ecuația tangentei în punctul M la curbă cu panta  $y'_m = f(x_m, y_m)$ . Grafic metoda este prezentată în figura 8.2. În acest punct H de coordonate  $(x_{m+1}, y_m + hy'_m)$  se calculează panta la curbă care este:

$$y'_{m+1} = f(x_m + h, y_m + hy'_m) \quad (8.22)$$

Se face media dintre panta în punctul M (dreapta  $ML_1$ ) și panta în punctul H (dreapta  $HL_2$ ) și o notăm cu

$$z_m = \frac{1}{2} \left( f(x_m, y_m) + f(x_m + h, y_m + hy'_m) \right) \quad (8.23)$$

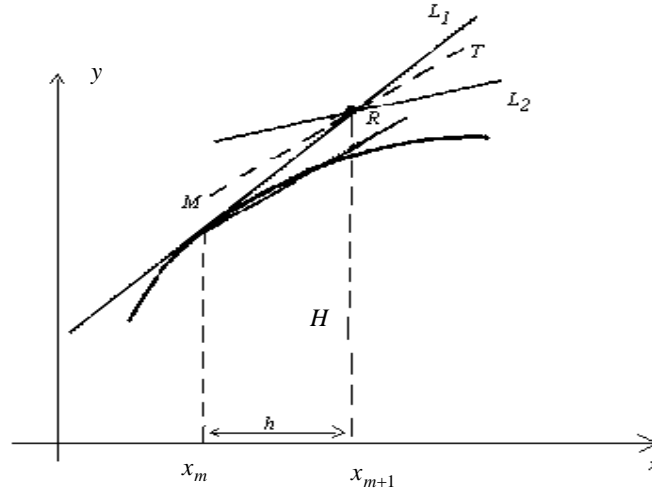


Fig.8.2. Calculul grafic al soluției prin metoda lui Euler îmbunătățită.

Dreapta de pantă  $z_m$  este HT. Cu această pantă  $z_m$  se trasează coarda prin M și se intersectează cu dreapta  $x = x_{m+1}$  rezultând punctul R, de coordonate  $(x_{m+1}, y_{m+1})$  al doilea punct de pe curba soluție, primul fiind considerat  $(x_m, y_m)$ . Ecuația coardei MR este:

$$y = y_m + 0.5 \cdot \left( f(x_m, y_m) + f(x_m + h, y_m + hy'_m) \right) (x - x_m) \quad (8.24)$$

Punctul R, care reprezintă soluția numerică a ecuației diferențiale, are ordonata:

$$y_{m+1} = y_m + 0.5 \cdot h \cdot \left( f(x_m, y_m) + f(x_m + h, y_m + hy'_m) \right) \quad (8.25)$$

unde  $y'_m = f(x_m, y_m)$ .

Această formulă (8.25) reprezintă formula de calcul a soluțiilor numerice pentru o ecuație diferențială ordinară de ordinul întâi.

#### 8.1.3.1.1. Precizia metodei lui Euler îmbunătățită

Se dezvoltă în serie Taylor funcția de două variabile  $f(x, y)$ :

$$f(x, y) = f(x_m, y_m) + (x - x_m)f_x(x_m, y_m) + (y - y_m)f_y(x_m, y_m) + \frac{1}{2} \left[ (x - x_m)^2 f_{xx}(\xi_0, \eta_0) + 2(x - x_m)(y - y_m)f_{xy}(\xi_0, \eta_0) + (y - y_m)^2 f_{yy}(\xi_0, \eta_0) \right] \quad (8.26)$$

unde  $\xi_0 \in [x, x_m]$  și  $\eta_0 \in [y, y_m]$

Dacă notăm  $x = x_m + h$  iar  $y = y_m + h \cdot y'_m$

unde  $y'_m = f(x_m, y_m)$  (8.27)

rezultă:

$$f(x_m + h, y_m + hy'_m) = f + hf_x + hff_y + \frac{1}{2}h^2 \left[ f_{xx}(\xi_0, y_0) + 2ff_{xy}(\xi_0, y_0) + f^2 f_{yy}(\xi_0, \eta_0) \right] \quad (8.28)$$

unde  $f = f(x_m, y_m)$ ,  $f_x = \frac{\partial f}{\partial x}(x_m, y_m)$ ,  $f_y = \frac{\partial f}{\partial y}(x_m, y_m)$  (8.29)

Înlocuind acest rezultat în (8.23) și (8.24) rezultă:

$$y_{m+1} = y_m + h \cdot \left[ f + \frac{h}{2}(f_x + ff_y) \right] + \frac{h^3}{4} \left[ f_{xy}(\xi_0, \eta_0) + 2ff_{xy}(\xi_0, \eta_0) + f^2 f_{yy}(\xi_0, \eta_0) \right] \quad (8.30)$$

unde  $\xi_0 \in [x_m, x_{m+1}]$  și  $\eta_0 \in [y_m, y_{m+1}]$

$$y_{m+1} = y_m + h \cdot \left[ f + 0.5h(f_x + ff_y) \right] \quad (8.31)$$

reprezintă formula de calcul din dezvoltarea Taylor de ordinul doi care este echivalentă cu formula de calcul a lui Runge-Kutta (8.25) și are eroarea de trunchiere:

$$e_T = h^3 \left\{ \frac{y'''(\xi)}{6} - \frac{1}{4} \left[ f_{xy}(\xi_0, \eta_0) + 2ff_{xy}(\xi_0, \eta_0) + f^2 f_{yy}(\xi_0, \eta_0) \right] \right\} \quad (8.32)$$

Dacă presupunem că  $f_{xy}, f_{xx}, f_{yy}$  și  $y'''$  sunt mai mici decât o constantă rezultă:

$$e_T \leq k \cdot h^3 \quad (8.33)$$

#### 8.1.3.1.2. Algoritmul 8.2. Metoda lui Euler îmbunătățită

```
{
  x0: abscisa punctului prin care trece graficul soluției, real;
  y0: ordonata punctului prin care trece graficul soluției, real;
  h: pasul între abscisele punctelor de calcul ale soluției, real;
  y: ordonatele soluției numerice, vector;
  x: abscisele soluției numerice, vector;
  n: numărul de puncte în care se calculează soluția, întreg;

  {
    x[0]=x0; y[0]=y0; h=const;
    pentru i = 1,...n
      {
        calculează x[i]=x[i-1]+i*h;
        calculează
        y[i]= y[i-1]+ 0.5*h*(f( x[i-1],y[i-1])+ f( x[i-1]+h, y[i-1]+h*f(x[i-1],y[i-1])));
      }
    Soluțiile numerice sunt x[i], y[i], i=0,1,2,...n;
  }
}
```

#### 8.1.3.1.3. Implementarea algoritmului 8.2

/\* Funcția care implementează algoritmul de rezolvare a ecuațiilor diferențiale ordinare prin metoda Euler îmbunătățită.

```

*/
void EULERIED( double(*f)(double x, double y),
               double x0,
               double y0,
               double pas,
               int nrp,
               double sol[])
{
    int i;
    sol[0]=y0;
    for(i=1;i<=nrp;i++)
        sol[i]=sol[i-1]+0.5*pas*( f(x0+(i-1)*pas,
        sol[i-1])+f(x0+i*pas,sol[i-1]+pas*f(x0+(i-1)*pas,sol[i-1])));
}

```

### 8.1.3.2. Metoda lui Euler modificată

În cadrul acestei metode nu se mediază pantele, ci se evaluează panta într-un punct care reprezintă media a două puncte. Considerăm curba din figura 8.3 în care se dă punctul inițial prin care să treacă curba soluție a ecuației diferențiale și dorim să determinăm cel de al doilea punct al soluției. Prin punctul  $(x_m, y_m)$  dat, cunoscut sau determinat, se duce tangenta MH și se determină pe această tangentă punctul H de coordonate  $(x_{m+1}, y_{m+1})$

Tangenta MH are ecuația

$$y = y_m + f(x_m, y_m)(x - x_m) \quad (8.34)$$

Prin intersecția cu dreapta  $x = x_m + h$  rezultă coordonatele punctului H

$$(x_{m+1} = x_m + h, \quad y_{m+1} = y_m + hf(x_m, y_m)) \quad (8.35)$$

prezentat în figura (8.3).

Metoda face media coordonatelor punctelor M și H și determină punctul

$$P\left(x_p = x_m + \frac{h}{2}, \quad y_p = y_m + \frac{h}{2} f(x_m, y_m)\right) \quad (8.36)$$

Se calculează panta soluției  $y$  în acest punct P și rezultă:

$$z_m = f\left(x_m + \frac{h}{2}, \quad y_m + \frac{h}{2} f(x_m, y_m)\right) \quad (8.37)$$

Se scrie ecuația dreptei care trece prin M de pantă  $z_m$  și o intersectăm cu dreapta

$$y = x_{m+1} = x_m + h$$

Dreapta MN are ecuația:

$$y = y_m + f\left(x_m + \frac{h}{2}, \quad y_m + \frac{h}{2} f(x_m, y_m)\right)(x - x_m) \quad (8.38)$$

Prin intersecția cu dreapta  $x = x_{m+1} = x_m + h$  rezultă ordonata:

$$y_{m+1} = y_m + hf\left(x_m + \frac{h}{2}, \quad y_m + \frac{h}{2} f(x_m, y_m)\right) \quad (8.39)$$

care reprezintă formula de calcul a ordonatelor soluției ecuației diferențiale.



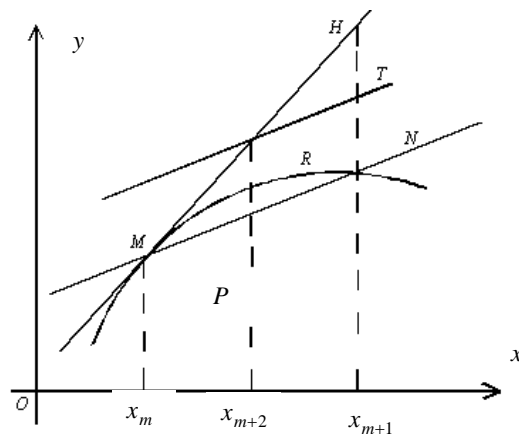


Fig.8.3.Calculul grafic al soluției ecuației diferențiale prin metoda lui Euler modificată.

Această metodă dă aceleași soluții ca și metoda lui Taylor până la  $h^2$ , deci este o metodă Runge-Kutta de ordinul doi.

$x_m + h/2$

#### 8.1.3.2.1. Algoritmul 8.3. Metoda lui Euler modificată

{Variable}

$x_0$ : abscisa punctului prin care trece graficul soluției, real;

$y_0$ : ordonata punctului prin care trece graficul soluției, real;

*h: pasul între abscisele punctelor de calcul ale soluției, real*

$y$ : ordonatele solutiei numerice, vector;

$x$ : abscisele soluției numerice, vector;

$n$ : numărul de puncte ale soluției, întreg;

```

{
x[0]= xo; y[0]= y0;
  pentru i = 1,...n
  {
    calculează x[i]= x[i-1]+i*h;
    calculează
    y[ i ]= y[i-1]+ h*f( x[i-1]+0.5*h, y[ i-1]+0.5*h*f(x[i-1],y[i-1])) ;
  }
  Soluțiile numerice ale ecuației sunt  x[ i ], y[ i ], i=0,...n;
}
}

```

#### 8.1.3.2.2. Implementarea algoritmului 8.3

/\* Funcția care implementează algoritmul de rezolvare a ecuațiilor  
diferențiale ordinare prin metoda Euler modificată.

```
*/
void EULERMED( double(*f)(double x, double y),
               double x0,
               double y0,
               double pas,
               int nrp,
               double sol[])
{
    int i;
    sol[0]=y0;
    for(i=1;i<=nrp;i++)
        sol[i]=sol[i-1]+pas*f(x0+(i-1)*pas+0.5*pas,sol[i-1])
                    +0.5*pas*f(x0+(i-1)*pas,sol[i-1]));
}
```

### 8.1.3.3. Asemănări între metodele Euler îmbunătățită și modificată

Ambele metode au formula de calcul a ordonatelor soluțiilor ecuației diferențiale

$$y_{m+1} = y_m + h \cdot u \quad (8.40)$$

unde  $u$  este panta  $z_m$  sau  $v_m$  care poate fi scrisă astfel:

$$u = a_1 f(x_m, y_m) + a_2 f(x_m + b_1 h, y_m + b_2 h) y'_m \quad (8.41)$$

unde

$$y'_m = f(x_m, y_m)$$

Pentru valorile  $a_1 = a_2 = \frac{1}{2}$  și  $b_1 = b_2 = 1$  avem panta  $z_m$ , în cazul aplicării metodei

Euler îmbunătățită, iar pentru valorile:  $a_1 = 0, a_2 = 1$  și  $b_1 = b_2 = 1/2$  avem panta  $v_m$ , în cazul aplicării metodei Euler modificată.

Dezvoltăm în serie Taylor funcția  $f(x, y)$  în jurul punctului  $(x_m, y_m)$ , dezvoltare prezentată în expresia (8.26) unde  $O(h^2)$  este restul și facem substituțiile:

$$x = x_m + b_1 h \quad \text{și} \quad y = y_m + b_2 h f$$

în această expresie:

$$f(x_m + b_1 h, y_m + b_2 h f) = f + b_1 h f_x + b_2 h f f_y + O(h^2) \quad (8.42)$$

Expresia lui  $u$  conform formulei (8.41) devine:

$$u = a_1 f + a_2 f + a_2 b_1 h f_x + a_2 b_2 h f f_y + O(h^2) \quad (8.43)$$

Înlocuind în formula (8.40) de calcul a valorilor funcției soluție a ecuației diferențiale rezultă:

$$y_{m+1} = y_m + h[(a_1 + a_2)f + h(a_2 b_1 f_x + a_2 b_2 f f_y)] + O(h^2) \quad (8.44)$$

Dacă comparăm această formulă (8.44) cu formula lui Taylor (8.30) rezultă:

$$a_1 + a_2 = 1; \quad a_2 b_1 = a_2 b_2 = \frac{1}{2}; \quad (8.45)$$

Deoarece avem trei ecuații cu patru necunoscute, alegem în mod arbitrar una din ele și exprimăm pe celelalte funcție de aceasta. Luăm  $a_2 = w$  și rezultă

$$a_1 = 1 - w, \quad b_1 = b_2 = \frac{1}{2w} \quad (8.46)$$

Formula de calcul (8.44) devine:

$$y_{m+1} = y_m + h \left[ (1-w)f(x_m, y_m) + w f\left(x_m + \frac{h}{2w}, y_m + \frac{h}{2w} f(x_m, y_m)\right) \right] + O(h^3) \quad (8.47)$$

Expresia (8.47) reprezintă formula de calcul a metodei Runge-Kutta de ordinul doi generală. Pentru  $w = \frac{1}{2}$  se obține metoda lui Euler îmbunătățită, iar pentru  $w = 1$  se obține metoda lui Euler modificată. Eroarea de trunchiere este:

$$e_T \approx k \cdot h^3 \quad (8.48)$$

În mod analog se pot dezvolta metodele Runge-Kutta de ordinul trei și patru.

#### 8.1.4. METODA RUNGE-KUTTA DE ORDINUL PATRU

Formula de calcul numeric a soluției ecuației diferențiale (8.1) este dată de expresia (8.49)

$$\begin{aligned} x_{m+1} &= x_m + h \\ y_{m+1} &= y_m + \frac{h}{6} [k_1 + 2k_2 + 2k_3 + k_4] \end{aligned} \quad (8.49)$$

unde

$$\begin{aligned} k_1 &= f(x_m, y_m) \\ k_2 &= f\left(x_m + \frac{h}{2}, y_m + \frac{h}{2} k_1\right) \\ k_3 &= f\left(x_m + \frac{h}{2}, y_m + \frac{h}{2} k_2\right) \\ k_4 &= f(x_m + h, y_m + h k_3) \end{aligned} \quad (8.50)$$

Eroarea de trunchiere a metodei este:

$$e_T \approx k \cdot h^5 \quad (8.51)$$

##### 8.1.4.1. Algoritmul 8.4. Metoda Runge-Kutta de ordinul 4

{ Variabile

$x_0$ : abscisa punctului prin care trece soluția, real;

$y_0$ : ordonata punctului prin care trece soluția, real;

$h$ : pasul între abscisele punctelor de calcul ale soluției, real;

$y$ : ordonatele soluției numerice, vector;

$x$ : abscisele soluției numerice, vector;

$n$ : numărul de puncte în care se calculează soluția, întreg;

```

{
  x[0]= x0; y[0]= y0 ;
  pentru i = 1,...,n
  {
    calculează x[ i-1 ]=x0+i*h;
    calculează
      k1 = f(x[ i-1 ], y[ i-1 ]);
    calculează
      k2 = f(x[ i-1 ]+0.5*h, y[ i-1 ]+ 0.5*h*k1);
    calculează
      k3 = f(x[ i-1 ]+0.5*h, y[ i-1 ]+ 0.5*h*k2);
    calculează
      k4 = f(x[ i-1 ]+h, y[ i-1 ]+ h*k4);
    calculează
      y[i] = y[i-1] +  $\frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$ ;
  }
  Soluțiile numerice ale ecuației sunt x[ i ], y[ i ], i=1,2,...,n;
}

```

#### 8.1.4.2. Implementarea algoritmului 8.4

```

/* Funcția care implementează algoritmul de rezolvare a ecuațiilor
diferențiale ordinare prin metoda Runge-Kutta de ordinul 4.
*/
void RK4( double(*f)(double x, double y),
          double x0,
          double y0,
          double pas,
          int nrp,
          double sol[])
{
  int i;
  double k1,k2,k3,k4;
  sol[0]=x0;
  for(i=1;i<=nrp;i++)
  {
    k1=f(x0+(i-1)*pas,sol[i-1]);
    k2=f(x0+(i-1)*pas+0.5*pas,sol[i-1]+0.5*pas*k1);
    k3=f(x0+(i-1)*pas+0.5*pas,sol[i-1]+0.5*pas*k2);
    k4=f(x0+i*pas,sol[i-1]+pas*k3);
    sol[i]=sol[i-1]+pas*(k1+2*k2+2*k3+k4)/6.0;
  }
}

```

**Concluzie:** Putem spune despre metodele Runge-Kutta că sunt metode cu viteză de calcul mare, deoarece calculul unui punct curent de pe curba soluției se face numai cu valorile calculate la punctul precedent. Precizia metodelor depinde de eroarea de trunchiere.

Dacă considerăm că în punctul  $x = x_0 + h$  valoarea exactă a soluției este  $y_m$ , cu metoda clasică de ordinul unu rezultă:

$$y_m = y_m^{(h)} + kh^2 \quad (8.52)$$

unde  $y_m^{(h)}$  arată că  $y_m$  s-a calculat cu pasul  $h$ . Se repetă calculul pentru pasul  $h/2$  și avem

$$y_m = y_m^{(h/2)} + k\left(\frac{h^2}{4}\right) \quad (8.53)$$

Din egalitatea relațiilor (8.52) și (8.53) rezultă:

$$y_m^{(h)} - y_m^{(h/2)} \approx -\frac{3}{4}h^2$$

iar eroarea de trunchiere devine:

$$e_T \approx kh^2 = \frac{4}{3}\left(y_m^{(h/2)} - y_m^{(h)}\right) \quad (8.54)$$

Aplicând metodele Runge - Kutta de ordinul doi rezultă:

$$e_T \approx kh^3 = \frac{8}{7}\left(y_m^{(h/2)} - y_m^{(h)}\right) \quad (8.55)$$

Metoda Runge - Kutta de ordinul patru are o eroare de trunchiere care poate fi estimată cu relația:

$$e_T \approx kh^5 = \frac{32}{31}\left(y_m^{(h/2)} - y_m^{(h)}\right) \quad (8.56)$$

dedusă prin același procedeu ca și la metoda Runge - Kutta de ordinul întâi.

În anumite condiții metodele Runge - Kutta pot da rezultate foarte imprecise chiar dacă erorile de trunchiere și rotunjire sunt mici. Aceasta se datorează faptului că erorile de trunchiere sau rotunjire pot crește odată cu  $x$ . Acest fenomen, care apare referitor la devierea mare a soluției a primit denumirea de *instabilitate parțială*, autorul fiind Mayers D.F. În acest caz instabilitatea depinde de ecuația diferențială, algoritm și dimensiunea intervalului.

### 8.1.5. METODE PREDICTOR-CORECTOR

Aceste metode prezic o valoare pentru soluție la pasul  $m+1$ ,  $y_{m+1}$ , apoi se utilizează o formulă pentru corecția ei. Formula de corecție se poate utiliza de mai multe ori pentru recorectări. Acest proces de iterație poate fi făcut eficient dacă se alege dimensiunea intervalului pentru un număr minim de iterații.

Considerăm formula predictor- corector de ordinul doi:

$$y_{m+1}^{(0)} = y_{m+1} + 2hf(x_m, y_m) \quad (8.57)$$

în care indicele (0) arată prima estimare a lui  $y_{m+1}$ .

Deoarece este necesar să se cunoască un punct anterior lui  $x_0$  se utilizează pentru pornirea metodei, metoda Runge-Kutta de ordinul doi.

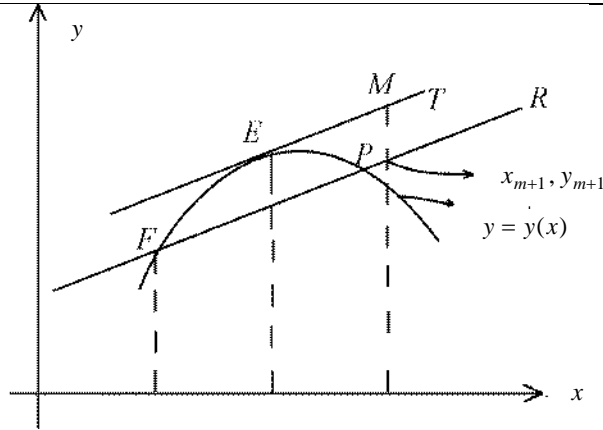


Fig. 8.4 Determinarea grafică a soluției prin metoda predictor - corector

Punctul prezis de prima estimare se calculează astfel: se duce tangenta ET, în punctul  $E(x_m, y_m)$  după care se duce coarda FR paralelă cu tangenta ET prin punctul  $F(x_{m-1}, y_{m-1})$  și intersectează dreapta  $x = x_{m+1}$  în punctul  $P(x_{m+1}, y_{m+1}^0)$  obținându-se prima valoare prezisă la estimarea zero. Se poate îmbunătăți valoarea  $y_{m+1}^0$  dacă se consideră panta în punctul  $(x_{m+1}, y_{m+1}^0)$  și se face media cu panta în  $(x_m, y_m)$ . Prin punctul  $(x_m, y_m)$  ducem o coardă cu această pantă care intersectează dreapta  $x = x_{m+1}$  în punctul  $(x_{m+1}, y_{m+1}^{(1)})$  unde  $y_{m+1}^{(1)}$  este estimarea de ordinul 1. Continuând acest procedeu se poate estima valoarea lui  $y_{m+1}^{(k)}$  până la o valoare  $k$ , până când

$$|y_{m+1}^{[k]} - y_{m+1}^0| < \varepsilon \quad (8.58)$$

unde  $\varepsilon$  este o eroare impusă.

Cu panta  $f(x_m, y_m)$  din punctul E se duce coarda prin F

$$y = y_{m-1} + f(x_m, y_m)(x - x_{m-1}) \quad (8.59)$$

Punctul de abscisă  $x_{m+1}$  pe această coardă are ordonata

$$y_{m+1}^{(0)} = y_{m-1} + 2hf(x_m, y_m) \text{ care reprezintă prima estimare a soluției.}$$

Ducem prin E coarda de pantă

$$z_m = \frac{1}{2} \left( f(x_m, y_m) + f(x_{m+1}, y_{m+1}^{(0)}) \right) \quad (8.60)$$

Punctul de abscisă  $x_{m+1}$  pe această coardă are ordonata

$$y_{m+1}^{(1)} = y_m + \frac{h}{2} \left[ f(x_m, y_m) + f(x_{m+1}, y_{m+1}^{(0)}) \right] \quad (8.61)$$

Procedând în mod analog pentru a  $k$ -a estimare avem

$$y_{m+1}^{(k)} = y_m + \frac{h}{2} \left[ f(x_m, y_m) + f(x_{m+1}, y_{m+1}^{(k-1)}) \right] \quad (8.62)$$

$$\text{Dacă } |y_{m+1}^{(k)} - y_{m+1}^{(k-1)}| < \varepsilon \text{ unde } \varepsilon > 0, \quad (8.63)$$

oricât de mic, iterațiile se opresc. Relația (8.63) este satisfăcută când metoda predictor-corector este convergentă.

$$y_{m+1}^{(k)} - y_{m+1}^{(k-1)} = \frac{h}{2} \left[ f(x_{m+1}, y_{m+1}^{(k-1)}) - f(x_{m+1}, y_{m+1}^{(k-2)}) \right] \quad (8.64)$$

Aplicând teorema valorii medii obținem

$$\left| y_{m+1}^{[k]} - y_{m+1}^{[k-1]} \right| = \frac{h}{2} \cdot \frac{\partial f}{\partial y} (y_{m+1}^{[k-1]} - y_{m+1}^{[k-2]}) \quad (8.65)$$

unde  $\frac{\partial f}{\partial y}$  este evaluată în  $x = x_{m+1}$  și  $y \in [y_{m+1}^{(k-1)}, y_{m+1}^{(k)}]$ .

Dacă  $\frac{\partial f}{\partial y}$  este mărginită de  $M > 0$  astfel ca:

$$\left| \frac{\partial f}{\partial y} \right| \leq M \quad (8.66)$$

$$\left| y_{m+1}^{(k)} - y_{m+1}^{(k-1)} \right| \leq \frac{M h}{2} \left| y_{m+1}^{(k-1)} - y_{m+1}^{(k-2)} \right|. \quad (8.67)$$

Analog

$$\left| y_{m+1}^{(k-1)} - y_{m+1}^{(k-2)} \right| \leq \frac{h M}{2} \left| y_{m+1}^{(k-2)} - y_{m+1}^{(k-3)} \right| \quad (8.68)$$

$$\text{sau} \quad \left| y_{m+1}^{(k)} - y_{m+1}^{(k-1)} \right| \leq \left( \frac{h M}{2} \right)^2 \left| y_{m+1}^{(k-2)} - y_{m+1}^{(k-3)} \right|. \quad (8.69)$$

Continuând calculul prin acest procedeu se ajunge la rezultatul:

$$\left| y_{m+1}^{(k)} - y_{m+1}^{(k-1)} \right| \leq \left( \frac{h M}{2} \right)^{k-1} \left| y_{m+1}^{(1)} - y_{m+1}^{(0)} \right| \quad (8.70)$$

$$\text{Dacă dimensiunea intervalului } h \text{ este bine aleasă adică } h < \frac{2}{M} \quad (8.71)$$

atunci metoda este convergentă. Viteza de convergență este cu atât mai mare cu cât  $h$  este mai mic.

### 8.1.5.1. Eroarea de trunchiere a metodei predictor-corrector

Pentru determinarea erorii de trunchiere se dezvoltă în serie Taylor funcția  $y(x)$  în jurul punctului  $x = x_m$

$$y(x) = y(x_m) + y'(x_m)(x-x_m) + \frac{y''(x_m)}{2}(x-x_m)^2 + \frac{1}{6}(x-x_m)^3 y'''(\xi) \quad (8.72)$$

unde  $x \leq \xi \leq x_m$

Considerând  $x = x_{m+1} = x_m + h$  rezultă:

$$y(x_{m+1}) = y(x_m) + h y'(x_m) + \frac{h^2}{2} y''(x_m) + \frac{h^3}{6} y'''(\xi_1) \quad (8.73)$$

unde  $x_m \leq \xi_1 \leq x_{m+1}$

Pentru  $x = x_{m-1} = x_m - h$  rezultă:

$$y(x_{m-1}) = y(x_m) - h y'(x_m) + \frac{h^2}{2} y''(x_m) - \frac{h^3}{6} y'''(\xi_2) \quad (8.74)$$

unde  $x_{m-1} \leq \xi_2 \leq x_m$

Scăzând din expresiile (8.67) și (8.68) rezultă:

$$y(x_{m+1}) = y(x_{m-1}) + 2h y'(x_m) + \frac{h^3}{3} y'''(\xi) \quad (8.75)$$

știind că

$$\frac{y'''(\xi_1) + y'''(\xi_2)}{2} = y'''(\xi) \quad \text{unde} \quad x_{m-1} \leq \xi \leq x_{m+1}$$

Comparând (8.69) cu (8.57) rezultă că eroarea metodei predictor- corector este :

$$e_T^{[p]} = \frac{h^3}{3} y'''(\xi) \quad \text{unde} \quad x_{m-1} \leq \xi \leq x_{m+1} \quad (8.76)$$

Formula (8.61) este analoagă formulei de la integrarea prin metoda trapezului unde am determinat eroarea

$$e_T^{(c)} = -\frac{h^3}{12} y'''(h) \quad \text{unde} \quad x_{m-1} \leq h \leq x_{m+1} \quad (8.77)$$

Calculăm valoarea adevărată  $y_m$  în punctul  $\xi_m$

$$y_m = y_m^{(0)} + \frac{h^2}{3} y'''(\xi) \quad (8.78)$$

iar după (8.71) avem

$$y_m = y_m^{(k)} - \frac{h^2}{12} y'''(h) \quad (8.79)$$

Scăzând relațiile (8.72) și (8.73) rezultă:

$$y_m^{(k)} - y_m^{(0)} = \frac{h^3}{12} [y'''(h) + 4y'''(\xi)]$$

Considerând  $y'''$  aproximativ constant pentru  $x_{m-1} \leq x \leq x_{m+1}$  se obține:

$$\frac{5h^3}{12} y''' = y_m^{(k)} - y_m^{(0)}$$

sau

$$e_T^{(c)} = -\frac{h^3}{12} y''' = \frac{1}{5} [y_m^{(0)} - y_m^{(k)}] \quad (8.80)$$

Convergența metodei este mai rapidă cu cât  $h$  este mai mic. Numărul de iterații nu trebuie să fie mare. Există dovezi care arată că cel mai eficient număr de iterații pentru metoda predictor-corector este doi.

### 8.1.5.2. Algoritm 8.5. Metoda predictor-corector

*{Variabile*

$x_0$  : abscisa punctului prin care trece soluția, real;

$y_0$  : ordonata punctului prin care trece soluția, real;

$h$  : pasul între abscisele punctelor de calcul ale soluției, real;

$y$  : ordonatele soluției numerice, vector;

$x$  : abscisele soluției numerice, vector;



$n$  : numărul de puncte în care se calculează soluția, întreg;

$er$  : eroarea de calcul, real;

```

{
  x[0]= x0; y[0]= y0;
  pentru i = 0,1,2,...,n
  {
    yi(0) = yi-1 + 2*h*f(x[i-1],y[i-1])
    k = 1;
    repetă
    calculează

    yi(k) = yi-1 + 0.5*[f(x[i-1],y[i-1]) + f(xi,yi(k-1))]
    k = k+1;

    până când

    |yi(k) - yi(k-1)| < e
  }
  Soluțiile numerice ale ecuației sunt x[i], y(k)[i], i=0,1,2,...,n;
}

```

### 8.1.5.3. Implementarea algoritmului 8.5

```

/* Funcția care implementează rezolvarea ecuațiilor diferențiale prin
   metoda predictor-corector 1.
*/
void PRED_COR( double(*f)(double x, double y),
               double x0,
               double y0,
               double pas,
               double eps,
               int nrp,
               double sol[])
{
  int i;
  double prec;
  /* start */
  sol[0]=y0;
  sol[1]=sol[0]+0.5*pas*(f(x0,sol[0])+f(x0+pas,sol[0]+
    pas*f(x0+pas,sol[0])));
  for(i=2;i<=nrp;i++)

```

```

{
  sol[i]=sol[i-2]+2*pas*f(x0+(i-1)*pas,sol[i-1]);
do
  {
    prec=sol[i];
    sol[i]=sol[i-1]+0.5*pas*( f(x0+(i-1)*pas,sol[i-1])+f(x0+i*pas,prec));
  }
  while( fabs(sol[i]-prec)>eps );
}
}

```

### 8.1.6. METODA MILNE

Această metodă folosește o pereche de formule de precizare și corectare, utilizează o integrare după metoda lui Simpson.

$$\begin{aligned}
 y_{k+1} &= y_{k-3} + \left(\frac{4h}{3}\right)(2y'_{k-2} - y'_{k-1} + 2y'_k) \\
 y_{k+1} &= y_{k-1} + \left(\frac{h}{3}\right)(y'_{k-1} + 4y'_k + y'_{k+1})
 \end{aligned} \tag{8.81}$$

Pentru aplicarea metodei din start trebuie cunoscute patru valori  $y_k, y_{k-1}, y_{k-2}, y_{k-3}$  care se obțin cu o metodă directă. Corecția soluției se face până când diferența dintre două valori consecutive ale soluției într-un punct devine mai mică decât o eroare  $\varepsilon$  impusă.

#### 8.1.6.1. Algoritmul 8.7. Metoda lui Milne

```

{Variabile
  x0 : abscisa punctului inițial al soluției, real;
  y0 : ordonata punctului inițial al soluției, real;
  h : pasul dintre punctele soluției, real;
  n : numărul de puncte în care se calculează soluția, întreg;
  {x[0]=x0; y[0]=y0;
  pentru k = 1, .. 3
    { calculează y_k = y_{k-1} + \left(\frac{h}{2}\right)[f(x_m, y_m) + f(x_m + h, y_m + hy'_m)];
      i = 3;
      repetă
        calculează y_{i+1}^{(0)} = y_{i-3} + \left(\frac{4h}{3}\right)(2y'_{i-2} - y'_{i-1} + 2y'_i);
        j = 1;
        repetă

```

```

calculază  $y_{i+1}^{(j)} = y_{i-1} + \left(\frac{h}{3}\right) \left( y_{i-1}' + y_i' + \left( y_{i+1}^{(j-1)} \right)' \right);$ 

j = j+1;
până când
 $\left| y_{i+1}^{(j)} - y_{i+1}^{(j-1)} \right| < e > 0 ;$ 

i = i+1;
până când i = n;
}
valorile unice ale soluției sunt  $y_i$  , i = 1, 2, ..., n
}
}

```

### 8.1.6.2. Implementarea algoritmului 8.7

```

/* Funcția care implementează metoda de rezolvare a ecuațiilor
diferențiale ordinare prin metoda Milne
*/
void MILNE( double(*f)(double x, double y),
            double x0,
            double y0,
            double pas,
            double eps,
            int nrp,
            double sol[])
{
    int i;
    double prec;
    /* start */
    sol[0]=y0;
    for(i=1;i<=3;i++)
        sol[i]=sol[i-1]+0.5*pas*( f(x0+(i-1)*pas,sol[i-1])
            +f(x0+i*pas,sol[i-1])+pas*f(x0+(i-1)*pas,sol[i-1]));
    for(i=4;i<=nrp;i++)
    {
        sol[i]=sol[i-4]+4*pas*( 2*f(x0+(i-3)*pas,sol[i-3])
            -f(x0+(i-2)*pas,sol[i-2])+2*f(x0+(i-1)*pas,sol[i-1]))/3;
        do
        {
            prec=sol[i];
            sol[i]=sol[i-2]+pas*( f(x0+(i-3)*pas,sol[i-3])
                +4*f(x0+(i-2)*pas,sol[i-2])+f(x0+i*pas,sol[i]))/3;
        }
        while( fabs(sol[i]-prec)>eps );
    }
}

```

## 8.2. INTEGRAREA NUMERICĂ A SISTEMELOR DE ECUAȚII DIFERENȚIALE DE ORDINUL ÎNTÂI ȘI A ECUAȚIILOR DIFERENȚIALE DE ORDINUL DOI

Ecuatiile de ordin superior cu condiții inițiale se referă la un sistem de ecuații diferențiale de ordinul întâi.

Considerăm ecuația diferențială de ordinul  $n$  de forma:

$$F(x, y, y', y'', \dots, y^{(n)}) = 0 \quad (8.82)$$

cu condițiile inițiale:

$$x = x_0, \quad y = y_0, \quad y' = y'_0, \dots, y^{(n-1)} = y^{(n-1)}_0. \quad (8.83)$$

Această ecuație diferențială poate fi scrisă sub forma unui sistem de ecuații diferențiale de ordinul întâi cu  $n$  funcții necunoscute.

Pe lângă funcția căutată  $y$  (soluția ecuației) mai introducem  $(n-1)$  necunoscute auxiliare:

$$y_1, y_2, \dots, y_{n-1} \quad (8.84)$$

legate de  $y$  prin ecuațiile:

$$\frac{dy}{dx} = y_1, \quad \frac{dy_1}{dx} = y_2, \quad \frac{dy_2}{dx} = y_3, \dots, \frac{dy_{n-2}}{dx} = y_{n-1}. \quad (8.85)$$

Se observă că

$$y_k = \frac{d^k y}{dx^k} = y^{(k)} \quad (8.86)$$

Ecuatia (8.82) o putem scrie sub forma

$$y^{(n)} = f(x, y, y', y'', \dots, y^{(n-1)}) \quad \text{sau} \quad (8.87)$$

$$\frac{dy_{n-1}}{dx} = f(x, y, y_1, y_2, \dots, y_{n-1})$$

Ecuatiile (8.87) și (8.85) formează un sistem de  $n$  ecuații diferențiale de ordinul întâi cu  $n$  funcții necunoscute. Dintre ecuațiile sistemului numai ultima are o formă mai generală, celelalte fiind mai speciale.

Condițiile inițiale ale sistemului vor fi:

pentru  $x = x_0$ , rezultă

$$y' = y'_0 = y_{10}, \quad y'' = y''_0 = y_{20}, \dots, y^{(n-1)} = y^{(n-1)}_0 = y_{(n-1)0} \quad (8.88)$$

Pentru fiecare ecuație se poate aplica una dintre metodele studiate pentru ecuațiile diferențiale de ordinul întâi. Dacă aplicăm metoda Runge - Kutta de ordinul patru se pleacă de la ultima ecuație (8.89) spre prima din (8.88).

Pentru  $i = 0, 1, 2, \dots, p$  rezultă:

$$\begin{cases}
 y_{n-1}^{(i+1)} = y_{n-1}^{(i)} + \frac{1}{6} \left( k_1^{(n-1)} + 2k_2^{(n-1)} + 2k_3^{(n-1)} + k_4^{(n-1)} \right) \\
 y_{n-2}^{(i+1)} = y_{n-2}^{(i)} + \frac{1}{6} \left( k_1^{(n-2)} + 2k_2^{(n-2)} + 2k_3^{(n-2)} + k_4^{(n-2)} \right) \\
 \text{-----} \\
 y_1^{(i+1)} = y_1^{(i)} + \frac{1}{6} \left( k_1^{(1)} + 2k_2^{(1)} + 2k_3^{(1)} + k_4^{(1)} \right) \\
 y^{(i+1)} = y^{(i)} + \frac{1}{6} \left( k_1 + 2k_2 + 2k_3 + k_4 \right)
 \end{cases} \quad (8.89)$$

unde

$$\begin{cases}
 k_1^{(n-1)} = hf \left( x_i, y_i, y_1^{(i)}, y_2^{(i)}, \dots, y_m^{(i)} \right) \\
 k_2^{(n-1)} = hf \left( x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_1^{(i)} + \frac{k_1^{(1)}}{2}, \dots, y_n^{(i)} + \frac{k_1^{(n-1)}}{2} \right) \\
 k_3^{(n-1)} = hf \left( x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_1^{(i)} + \frac{k_2^{(1)}}{2}, \dots, y_n^{(i)} + \frac{k_2^{(n-1)}}{2} \right) \\
 k_4^{(n-1)} = hf \left( x_i + h, y_i + k_3, y_1^{(i)} + k_3^{(1)}, y_2^{(i)} + k_3^{(2)}, \dots, y_m^{(i)} + k_3^{(n-1)} \right) \\
 k_1^{(n-2)} = h y_{n-1}^{(i)} \\
 k_2^{(n-2)} = h \left( y_{n-1}^{(i)} + \frac{k_1^{(n-1)}}{2} \right) \\
 k_3^{(n-2)} = h \left( y_{n-1}^{(i)} + \frac{k_2^{(n-1)}}{2} \right) \\
 k_4^{(n-2)} = h \left( y_{n-1}^{(i)} + k_3^{(n-1)} \right)
 \end{cases} \quad (8.91)$$

Analog se calculează și celelalte valori ale lui  $k$  și după înlocuire în  $y^{i+1}$  se obține formula de recurență pentru calculul soluțiilor numerice ale ecuației diferențiale.

Cazul general de tratare a problemei este foarte laborios, dar pentru cazul unei ecuații diferențiale de ordinul doi sau trei, calculele se simplifică mult. Pentru o ecuație de ordinul doi

$$y'' = f(x, y, y'), \quad (8.92)$$

rezolvată cu metoda Runge - Kutta de ordinul patru, rezultă sistemul de ecuații de ordinul trei:

$$\begin{cases}
 \frac{dy_1}{dx} = f(x, y, y_1) \\
 \frac{dy}{dx} = y_1
 \end{cases} \quad (8.93)$$

corespunzător lui (8.84) și (8.85).

Relațiile de calcul (8.90) și (8.91) devin:

$$\begin{aligned} y_1^{(i+1)} &= y_1^{(i)} + \frac{1}{6}(k_1^1 + 2k_2^1 + 2k_3^1 + k_4^1) \\ y_{i+1} &= y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \quad (8.94)$$

unde

$$\begin{cases} k_1' = hf(x_i, y_i, y_1^i) \\ k_2' = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_1^i + \frac{k_1^1}{2}\right) \\ k_3' = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_1^i + \frac{k_2^{(1)}}{2}\right) \\ k_4' = hf(x_i + h, y_i + k_3, y_1^i + k_3^1) \end{cases} \quad (8.95)$$

$$\begin{cases} k_1 = hy_1^i \\ k_2 = h\left(y_1^i + \frac{k_1}{2}\right) \\ k_3 = h\left(y_1^i + \frac{k_2}{2}\right) \\ k_4 = h(y_1^i + k_3) \end{cases} \quad (8.96)$$

Înlocuind (8.95) în (8.94) rezultă formulele de calcul:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + k_2 + k_3) \quad i = 0, 1, 2, \dots, n$$

unde

$$\begin{cases} y_1^{(i+1)} = y_1^{(i)} + \frac{1}{6}(k_1 + 2k_2 + k_3 + k_4) \\ k_1 = hf(x_i, y_i, y_1^{(i)}) \\ k_2 = hf\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}y_1^{(i)}, y_1^{(i)} + \frac{k_1}{2}\right) \\ k_3 = hf\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}y_1^{(i)} + \frac{h}{4}k_1, y_1^{(i)} + \frac{k_2}{2}\right) \\ k_4 = hf\left(x_i + h, y_i + h y_1^{(i)} + \frac{h}{2}k_2, y_1^{(i)} + k_3\right) \end{cases} \quad (8.97)$$

### 8.3. REZOLVAREA NUMERICĂ A ECUAȚIILOR DIFERENȚIALE CU DERIVATE PARȚIALE

Vom considera ecuațiile diferențiale cu derivate parțiale de ordinul doi de două variabile ce au forma generală:

$$a(x, y) \frac{\partial^2 u}{\partial x^2} + 2b(x, y) \frac{\partial^2 u}{\partial x \partial y} + c(x, y) \frac{\partial^2 u}{\partial y^2} + d(x, y) \frac{\partial u}{\partial x} + e(x, y) \frac{\partial u}{\partial y} + f(x, y) u = 0 \quad (8.98)$$

unde

$$a(x, y), b(x, y), c(x, y), d(x, y), e(x, y), f(x, y)$$

sunt funcții date într-un domeniu plan  $D$ . Funcția necunoscută  $u(x, y)$  și derivatele parțiale apar în ecuație la puterea întâi. Ecuația are proprietatea că dacă  $u_1(x, y)$  și  $u_2(x, y)$  sunt două soluții ale ecuației, atunci și combinația liniară

$$c_1 u_1 + c_2 u_2 \quad (8.99)$$

este tot o soluție a ecuației unde  $c_1$  și  $c_2$  sunt constante. Aceste tipuri de ecuații se clasifică după semnul valorii determinantului atașat formulei (8.98).

$$\Delta = \begin{vmatrix} a(x, y) & b(x, y) \\ b(x, y) & c(x, y) \end{vmatrix} \quad (8.100)$$

1. Dacă  $\Delta > 0$  ecuația se numește de tip eliptic
2. Dacă  $\Delta < 0$  ecuația se numește de tip hiperbolic
3. Dacă  $\Delta = 0$  ecuația se numește de tip parabolic

### 8.3.1. METODA DIFERENȚELOR FINITE

Această metodă constă în acoperirea domeniului  $D$  și o frontieră a lui  $C$ , cu o rețea de drepte paralele cu axele de coordonate. Pașii rețelei sunt constanți, de valoare  $h$  pentru axa  $Ox$  și de valoare  $k$  pentru axa  $Oy$ . Nodurile rețelei se împart și ele în două categorii: una în care punctele au toate vecinele în interiorul lui  $D$  și a doua pentru care cel puțin unul dintre vecinele lui este exterior domeniului  $D$  și aceasta formează frontiera  $C_1$  a rețelei. Se aproximează derivatele parțiale de ordinul unu și doi din ecuația (8.98) cu diferențele finite corespunzătoare:

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2} &\approx \frac{u(x+y, y) - 2u(x, y) + u(x-h, y)}{h^2} \\ \frac{\partial^2 u}{\partial y^2} &\approx \frac{u(x, y+h) - 2u(x, y) + u(x, y-h)}{k^2} \\ \frac{\partial^2 u}{\partial x \partial y} &\approx \frac{u(x+h, y+h) - u(x, y+h) - u(x+h, y) + u(x, y)}{h k} \\ \frac{\partial u}{\partial x} &\approx \frac{u(x+h, y) - u(x, y)}{h} \\ \frac{\partial u}{\partial y} &\approx \frac{u(x, y+k) - u(x, y)}{k} \end{aligned} \quad (8.101)$$

Ecuația (8.98) scrisă cu ajutorul diferențelor finite devine

$$\begin{aligned}
u(x, y) = & \frac{1}{2k^2a + 2h^2c + 2hkb + hk^2d + kh^2e + k^2h^2f} [k^2au(x+h, y) + k^2au(x-h, y) + \\
& + h^2cu(x, y+k) + k^2cu(x, y-k) + 2khbu(x+h, y+k) - 2khbu(x, y+k) - 2khbu(x+h, y) + \\
& + k^2hdu(x+h, y) + kh^2eu(x, y+h)]
\end{aligned} \quad (8.102)$$

Pentru cazul egalității valorilor pașilor  $h = k$  în ecuația obținută (8.102) rezultă:

$$\begin{aligned}
u(x, y) = & \frac{1}{2a + 2c + 2b + h d + h e + h^2 f} [a u(x+h, y) + a u(x-h, y) + c u(x, y+h) + \\
& + c u(x, y-h) + 2b u(x+h, y+h) - 2b u(x, y+h) - 2b u(x+h, y) + \\
& + h d u(x+h, y) + h e u(x, y+h)]
\end{aligned} \quad (8.103)$$

Dacă se consideră rețeaua prezentată în figura 8.1 și se țin seama de condițiile la limită, atunci pentru nodul (1,1) se obține ecuația:

$$\begin{aligned}
u(1,1) = & \frac{1}{2a_1 + 2c_1 + 2b_1 + h d_1 + h e_1 + h^2 f_1} \cdot \\
& [a_1 u(2,1) + c_1 u(1,2) + c_1 + 2b_1 u(2,2) - 2b_1 u(1,2) - 2b_1 u(2,1) + d_1 u(2,1) + e u(1,2)]
\end{aligned} \quad (8.104)$$

unde am considerat  $h=k=1$  pentru rețeaua din figura 8.5. Procedând la înlocuiri în fiecare punct al domeniului D se obține un sistem din care calculăm valorile funcției soluție a ecuației (8.98). Funcție de semnul lui  $\Delta$  se rezolvă tipuri de ecuații eliptice, hiperbolice sau parabolice.

Pentru  $a(x, y) = 1$ ,  $b(x, y) = 0$ ,  $c(x, y) = 1$ ,  $d(x, y) = 0$ ,  $e(x, y) = 0$ ,  $f(x, y) = 0$  rezultă o particularizare a ecuației (8.98) obținându-se ecuația lui Laplace.

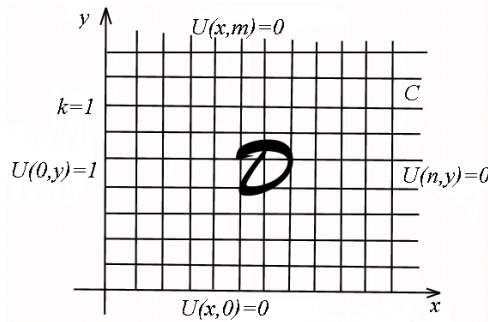


Fig 8.5 Rețeaua domeniului funcției soluție și condițiile la limită

### 8.3.1.1. Ecuația lui Laplace

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad (8.105)$$

Aplicând relațiile 8.101, ecuația lui Laplace se scrie funcție de diferențele finite astfel:

$$u(x, y) = \frac{1}{4} [u(x+h, y) + u(x-h, y) + u(x, y+h) + u(x, y-h)] \quad (8.106)$$



Considerăm rețeaua din fig.8.6 și notăm cu  $u_{11}, u_{12}, u_{13}, u_{21}, u_{22}, u_{23}, u_{31}, u_{32}, u_{33}$

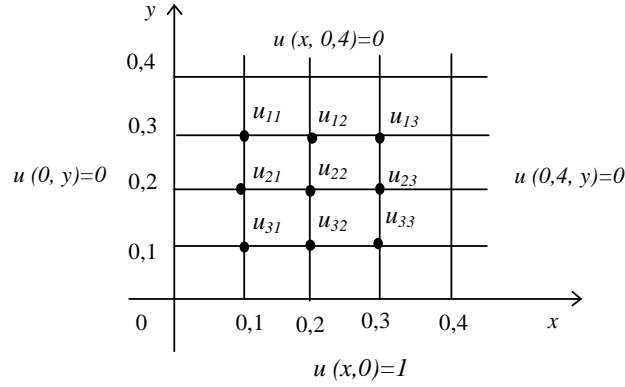


Fig.8.6 Rețeaua domeniului funcției și condițiile la limită pentru ecuația lui Laplace.

valorile funcției în punctele 1, 2, 3, 4, 5, 6, 7, 8, 9. Se obține sistemul (8.107) care este un sistem iterativ de tip Jacobi (Capitolul 3).

Prin rezolvarea recursivă a sistemului, rezultă valorile funcției soluție a ecuației diferențiale cu derivate parțiale Laplace în punctele rețelei. Rețeaua se poate realiza cu un pas mai mic ceea ce duce la un sistem cu multe ecuații și necunoscute, determinând valorile funcției soluție în punctele mai fine ale rețelei.

$$\left\{ \begin{array}{l} u_{11} = \frac{1}{4}(0 + u_{12} + 0 + 0 + 0 + 0 + 0 + 0 + 0 + 1) \\ u_{12} = \frac{1}{4}(u_{11} + 0 + u_{13} + 0 + u_{22} + 0 + 0 + 0 + 0 + 1) \\ u_{13} = \frac{1}{4}(0 + u_{12} + 0 + 0 + 0 + u_{23} + 0 + 0 + 0 + 1) \\ u_{21} = \frac{1}{4}(u_{11} + 0 + 0 + 0 + u_{22} + 0 + u_{31} + 0 + 0 + 0) \\ u_{22} = \frac{1}{4}(0 + u_{12} + 0 + u_{21} + 0 + u_{23} + 0 + u_{32} + 0) \\ u_{23} = \frac{1}{4}(0 + 0 + u_{13} + 0 + u_{22} + 0 + 0 + 0 + u_{33} + 0) \\ u_{31} = \frac{1}{4}(0 + 0 + 0 + u_{21} + 0 + 0 + 0 + u_{32} + 0 + 0) \\ u_{32} = \frac{1}{4}(0 + 0 + 0 + 0 + u_{22} + 0 + u_{31} + 0 + u_{33} + 0) \\ u_{33} = \frac{1}{4}(0 + 0 + 0 + 0 + u_{22} + 0 + u_{31} + 0 + u_{33} + 0) \\ u_{33} = \frac{1}{4}(0 + 0 + 0 + 0 + 0 + u_{23} + 0 + u_{32} + 0 + 0) \end{array} \right. \quad (8.107)$$

**8.3.1.1.1. Algoritmul 8.8. Ecuația Laplace**

```

{ Variabile
lsx:limita stângă a domeniului pe Ox, reală;
ldx:limita dreaptă a domeniului pe axa Ox, reală;
lsy:limita stângă a domeniului pe axa Oy, reală;
ldy:limita dreaptă a domeniului pe axa Oy, reală;
nrx:numărul de subintervale pe Ox, întreg;
nry:numărul de subintervale pe Oy, întreg;
h:pasul pe Ox, real;
k:pasul pe Oy, real;
{calculează h=(ldx-lsx)/nrx;
k=h;
pentru i=0 până la nrx
{
u(lsx + ih,0) = 1;
u(lsx + ih,ldy) = 0
}
pentru j=1 până la nrt
u(0,lsy + jk) = 0;
u(ldx,lsy + jk) = 0;
}
pentru i=1 până la nrx
pentru j=1 până la nrt
{
calculează

$$u(x_i, y_j) = \frac{1}{4}(u(lsx + (i+1)h, lsy + jh) + u(lsx + (i-1)h, lsy + jh) + u(lsx + ih, lsy + (j+1)h) + u(lsx + ih, lsy + (j-1)h))$$

}
construiește sistemul iterativ și -l rezolvă cu metoda Iacobi sau Gauss-Seidel
tipărește  $u_{ij}$  pentru i=0 până la nrx
j=0 până la nrt;
}

```

**8.3.1.1.2. Implementarea algoritmului 8.8**

```

/*
Funcția care implementează metoda de rezolvare a ecuațiilor
diferențiale de tip eliptic
*/
void ELIPTIC( int ord,

```

```

double Domeniu[][NrMax]
)
{
int i,j,nrp,lung;
double mat[NrMax][NrMax];
lung=ord-2;
for(nrp=1;nrp<=ord*ord-4*ord+4;nrp++)
{
for(i=1;i<=ord-2;i++)
for(j=1;j<=ord-2;j++)
{
if ( (nrp/lung==(i-1)) && (nrp%lung==j) )
{
mat[i][j]=-1;
mat[i-1][j]=mat[i+1][j]=mat[i][j+1]=mat[i][j-1]=0.25;
}
else mat[i][j]=0;
}
}
}
i=20;
}

```

### 8.3.1.2. Ecuații diferențiale cu derivate parțiale de tip parabolic

Vom considera ecuația căldurii într-o bară de lungime  $l$  ce reprezintă o ecuație cu derivate parțiale de tip parabolic:

$$\frac{1}{a^2} \frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0 \quad (8.108)$$

$u(x,t)$  reprezintă temperatura funcție de coordonata punctului și timp. Prin schimbarea de variabilă  $t = a^2 t'$  ecuația (8.108) se poate scrie:

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0 \quad (8.109)$$

Pentru rezolvarea ecuației se aplică metoda diferențelor finite pe rețeaua prezentată în figura (8.7) cu pașii  $h = \frac{l}{n}$  pe axa  $Ox$  și  $k = \eta h^2$  pe axa  $Ot$ .  $\eta=1/6$  reprezintă constanta pentru care eroarea este minimă dacă rezolvarea se face cu metoda diferențelor finite.

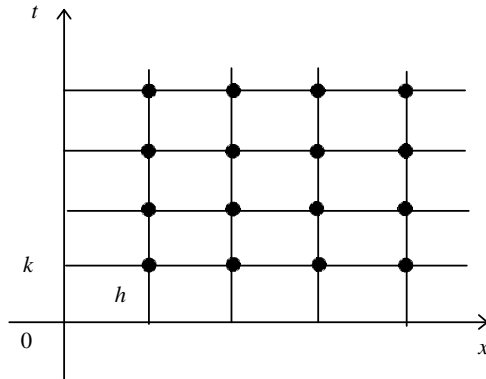


Fig.8.7.Rețeaua domeniului pe care se integrează ecuația diferențială cu derivate parțiale de tip parabolic.

Presupunem date condiția inițială  $u(x,0) = f(x)$  și condițiile la limită:

$$u(0,t) = s(t), \quad u(1,t) = z(t).$$

Ecuția (8.109), scrisă cu diferențe finite, devine:

$$\frac{u_{i,j+1} - u_{i,j}}{\eta \cdot h^2} = \frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{h^2} \quad (8.110)$$

Din ecuația (8.110) se deduce formula de calcul a valorilor funcției în punctele rețelei:

$$u_{ij+1} = \frac{1}{6}(u_{i-1,j} + 4u_{i,j} + u_{i+1,j}) \quad (8.111)$$

Pentru calculul valorilor funcției se ține seama de condițiile inițiale și de cele la limită.

### 8.3.1.2.1. Algoritm 8.9. Ecuații cu derivate parțiale de tip parabolic

*{Variabile*

*lsx:limita stângă a domeniului pe Ox, reală;*

*ldx:limita dreaptă a domeniului pe axa Ox, reală;*

*lst:limita stângă a domeniului pe axa Ot, reală;*

*ldt:limita dreaptă a domeniului pe axa Ot, reală;*

*nrx:numărul de subintervale pe Ox, întreg;*

*nrt:numărul de subintervale pe Ot, întreg;*

*h:pasul pe Ox, real;*

*k:pasul pe Ot, real;*

*{pentru i=0 până la nrx*

*calculează  $u_{i0} = f(lsx + i \cdot h)$ ;*

*calculează  $k = (1/6) \cdot h$ ;*

*pentru j=1 până la nrt*

*{*  
*calculează  $u_{0j} = s(lst + j \cdot k)$ ;*

*calculează  $u_{ij} = z(lst + j \cdot k)$ ;*

*}*

*pentru i=1 până la nrx*

```

    pentru j=1 până la nrt
        calculează  $u_{ij} = \frac{1}{6}(u_{i-1,j-1} + 4u_{i,j-1} + u_{i+1,j-1})$ ;
    tipărește  $u_{ij}$  pentru i=0 până la nrx
        j=0 până la nrt;
    }

```

### 8.3.1.2.2. Implementarea algoritmului 8.9

```

/* Funcția care implementează metoda de rezolvare a ecuațiilor
diferențiale
cu derivate parțiale de tip parabolic.
*/
void Parabolica( double (*F)(double),
                double (*S)(double),
                double (*Z)(double),
                double lims,
                double limd,
                int nrx,
                int nry,
                double sol[][NMax]
                )
{
    int i,j;
    double h,k;
    h=(limd-lims)/nrx;
    k=h*h/6;
    for(i=0;i<=nrx;i++)sol[i][0]=f(lims+i*h); /*pe orizontala de jos */
    for(j=1;j<=nry;j++)
    {
        sol[0][j]=s(j*k); /* verticala stanga */
        sol[nrx][j]=z(j*k); /*verticala dreapta */
    }
    for(j=1;j<=nry;j++)
        for(i=1;i<=nrx-1;i++)
            sol[i][j]=(sol[i-1][j-1]+4*sol[i][j-1]+sol[i+1][j-1])/6;
}

```

### 8.3.1.3. Ecuațiile cu derivate parțiale de tip hiperbolic

O ecuație de acest tip este ecuația oscilațiilor libere ale unei bare omogene finite:

$$\frac{\partial^2 u}{\partial t^2} - a^2 \frac{\partial^2 u}{\partial x^2} = 0 \quad (8.112)$$

care are condițiile la inițiale:  $u(x,0) = f(x)$   $u'_t(x,0) = g(x)$ ,  $0 \leq x \leq l$  și condițiile la limită:  $u(0,t) = s(t)$ ,  $u(1,t) = z(t)$ ,  $0 \leq t \leq \infty$ . Se poate lua același domeniu de integrare dat în figura 8.7. Ecuația cu derivate parțiale (8.112) poate fi transcrisă cu ajutorul diferențelor finite, aplicate pe o rețea dreptunghiulară de pas  $h=l/n$  pe axa  $Ox$  și de pas  $k$  întreg oarecare astfel:

$$\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{k^2} = a^2 \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \quad (8.113)$$

Dacă se alege  $h = ka$ , ecuația (8.113) se simplifică la forma:

$$u_{i,j+1} = u_{i+1,j} - u_{i,j-1} + u_{i-1,j}. \quad (8.114)$$

Cu ajutorul formulei de calcul (8.114) se obțin valorile funcției  $u(x,t)$  în punctele domeniului dat astfel ca funcția să verifice ecuația (8.112).

#### 8.3.1.3.1. Algoritmul 8.10. Ecuații cu derivate parțiale de tip hiperbolic

```
{Variabile
lsx:limita stângă a domeniului pe Ox, reală;
ldx:limita dreaptă a domeniului pe axa Ox, reală;
lst:limita stângă a domeniului pe axa Ot, reală;
ldt:limita dreaptă a domeniului pe axa Ot, reală;
nrx:numărul de subintervale pe Ox, întreg;
nrt:numărul de subintervale pe Ot, întreg;
h:pasul pe Ox, real;
k:pasul pe Ot, real;
{
  calculează  $h = \frac{l}{nrx}$ ;
  calculează  $k = \frac{h}{a}$ ;
  pentru  $i=0$  până la  $nrx$ 
  {
    calculează  $u_{i0} = f(lsx + i * h)$ ;
    calculează  $u_{i,-1} = -k * g(lsx + i * h) + u_{i0}$ 
  }
  pentru  $j=1$  până la  $nrt$ 
  {
    calculează  $u_{0j} = s(lst + j * k)$ ;
    calculează  $u_{jj} = z(ldt + j * k)$ ;
  }
  pentru  $i=1$  până la  $nrx$ 
```

---

```

    pentru j=1 până la nrt
        calculează  $u_{ij} = (u_{i+1,j-1} + u_{i,j-2} + u_{i+1,j-1})$ ;
    pentru i=0 până la nrx
        pentru j=0 până la nrt tipărește  $u_{ij}$ 
    }

```

### 8.3.1.3.2. Implementarea algoritmului 8.10

```

/*
  Funcția care implementează metoda de rezolvare a ecuațiilor
  diferențiale cu derivate parțiale de tip hiperbolic
*/

void Hiperbolica( double (*F)(double),
                  double (*S)(double),
                  double (*Z)(double),
                  double lims,
                  double limd,
                  int nrx,
                  int nry,
                  double a,
                  double sol[][NMax]
                )

{
    int i,j,k1;
    double h,k;
    h=(limd-lims)/nrx;
    k=k1;
    for(i=0;i<=nrx;i++)sol[i][0]=f(lims+i*h); /*pe orizontala de jos */
    for(j=1;j<=nry;j++)
    {
        sol[0][j]=s(j*k); /* verticala stânga */
        sol[nrx][j]=z(j*k); /*verticala dreapta */
    }

    for(i=1;i<=nrx-1;i++)
        sol[i][1]=sol[i+1][0]+sol[i-1][0]-sol[i][0]+k*g(lims+i*h);
    for(j=2;j<=nry;j++)
        for(i=1;i<=nrx-1;i++)
            sol[i][j]=(sol[i+1][j-1]-sol[i][j-2]+sol[i-1][j-1]);
}

```

## 8.4. APLICAȚIE

Se consideră un circuit  $R,L$  în serie alimentat de o sursă de curent alternativ  $e = 10\cos(100\pi t)$  V,  $f = 50\text{Hz}$ , prezentat în fig.(8.8).

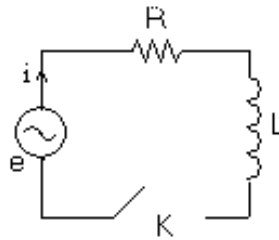


Fig.8.8. Circuitul  $R,L$

Cunoscând  $R=10\Omega$ ,  $L=10\text{mH}$  și că la  $t=0$   $i=0$ , să se calculeze valorile curentului la  $t=0.001$ ;  $0.002$ ;  $0.003$ ;  $0.004$ ;  $0.005$ ;  $0.006$ ;  $0.007$ ;  $0.008$ ;  $0.009$ ;  $0.01\text{s}$ .

Se aplică circuitului electric legea a doua a lui Kirchhoff și se obține ecuația diferențială

$$L \frac{di}{dt} + Ri = e$$

Rezultatele obținute prin metoda lui Euler modificată sunt prezentate în tabelul (8.1)

Tabelul 8.1

t[s]	i[A]
0.0000	0.00000
0.00100	0.48770
0.00200	0.65941
0.00300	0.63249
0.00400	0.447665
0.00500	0.24065
0.00600	- 0.03564
0.00700	-0.31683
0.00800	-0.57124
0.00900	-0.77188
0.01000	-0.89808



# 9

## REZOLVAREA NUMERICA A ECUAȚIILOR INTEGRALE\*

Ecuatiile integrale sunt ecuații în care funcția necunoscută se găsește sub semnul integralei. Ecuatiile integrale liniare se clasifică în funcție de tipul limitelor de integrare și modul de apariție în ecuație a funcției necunoscute astfel :

- 1 - ecuații integrale la care limitele de integrare sunt constante și se numesc ecuații de tip Fredholm ;
- 2 - ecuații integrale la care o limită de integrare este variabilă și se numesc ecuații de tip Volterra.

În ambele cazuri, dacă funcția necunoscută este numai sub integrală, spunem că ecuațiile sunt de speța întâia, iar dacă funcția necunoscută este și în afara integralei, spunem că ecuațiile sunt de speța a doua.

$$\begin{array}{l} \text{Ecuații} \\ \text{integrale} \\ \text{liniare} \end{array} \left\{ \begin{array}{l} \text{de tip Fredholm} \\ \text{de tip Volterra} \end{array} \right\} \begin{cases} \text{speța I} \\ \text{speța II} \\ \text{speța I} \\ \text{speța II} \end{cases}$$

### 9.1. INTEGRAREA ECUAȚIEI FREDHOLM NEOMOGENĂ DE SPEȚA A DOUA PRIN METODA APROXIMĂRIILOR SUCCESIVE

Se consideră ecuația:

$$\varphi(x) = f(x) + \lambda \int_a^b S(x, y)\varphi(y)dy \quad (9.1)$$

unde  $f$  și  $S$  sunt funcții date ,  $\varphi$  este funcția necunoscută pe care o determinăm și  $\lambda$  un parametru numeric suficient de mic, astfel că pentru început poate fi aproximată soluția :

$$\varphi_1(x) = f(x) \quad (9.2)$$

Se obține formula de recurență

---

\*Bibliografie [12], [17], [18], [22]

$$\varphi_k(x) = f(x) + \lambda \int_a^b S(x, y) \varphi_{k-1}(y) dy \quad (9.3)$$

Pentru convergența 0irului de soluții aproximative se scad funcțiile (9.1) și (9.3) obținându-se eroarea :

$$\varepsilon_k(x) = \varphi(x) - \varphi_k(x) = \lambda \int_a^b S(x, y) \varepsilon_{k-1}(y) dy \quad (9.4)$$

Dacă  $|S(x, y)| \leq M$  oricare ar fi  $(x, y) \in [a, b] \times [a, b]$  atunci

$$\lambda \int_a^b |S(x, y)| \varepsilon_{k-1}(y) dy \leq \lambda S(b-a) E_{k-1} \quad (9.5)$$

unde  $E_k$  este maximul lui  $|\varepsilon_k(x)|$  din  $[a, b]$ .

Convergența 0irului de soluții este satisfăcută dacă

$$\lambda M(b-a) < 1. \quad (9.6)$$

Utilizând pentru integrare o metodă de cuadratură cunoscută se obține din ecuația (9.3) ecuația :

$$\varphi_i = f_i + \lambda \int_a^b S(x, y) \varphi(y) dy \approx f_i + \lambda h \sum_{j=0}^n A_j S(x_i, y_j) \varphi(y_j) \quad (9.7)$$

$$\text{sau } \varphi_i = f_i + \lambda \sum_{j=0}^n C_{ij} \varphi_j \quad i = 0, 1, \dots, n$$

care reprezintă un sistem în necunoscutele  $\varphi_i$ . Sistemul se poate rezolva iterativ aplicând formula de iterație:

$$\varphi_i^{(k)} = f_i + \lambda \sum_{j=0}^n C_{ij} \varphi_j^{(k-1)} \quad i = 0, 1, 2, \dots, n \quad (9.8)$$

Procesul de iterație se continuă până când

$$\left| \varphi_i^{(k)} - \varphi_i^{(k+1)} \right| < \varepsilon \quad (9.9)$$

unde  $\varepsilon > 0$  reprezintă eroarea de calcul.

### 9.1.1. ALGORITMUL 9.1. METODA LUI FREDHOLM

{Variabile

$l$  : constanta ecuației ;

$x$  : vectorul punctelor de pe  $0x$ , real ;

$y$  : vectorul punctelor de pe  $0y$ , real ;

$M$  : matricea punctelor formată de vectorul  $x$  și  $y$  ;

$C_{ij}$  : valorile funcției cunoscute  $S(x, y)$  în punctele matricei  $M$  reale

înmulțită cu ponderile din metoda cuadraturii, real ;

$a$  : limita stângă de integrare, reală ;

$b$  : limita dreaptă de integrare, reală ;

$\varphi_0$  : valoarea inițială a funcției soluție, real ;  
 $f_i$  : valorile funcției cunoscute în variabilele vectorului  $x$ , real ;  
 $sum$  : suma parțială, real ;  
 $\varepsilon$  : eroarea, real ;  
 $n$  : numărul de puncte în care se calculează soluția, întreg ;  
 $\{ \varphi_0 = \alpha ;$   
 $i = 1;$   
 repetă  
 $\varphi_i^{(k)} = \varphi_0;$   
 $k = 1;$   
 repetă  
 $\{ \text{calculează } f_i = f(x_i);$   
 pentru  $j = 1 \dots n$   
 $\{ \text{calculează } sum = sum + \lambda * A_i * S(x_i, y_j) * \varphi^{(k-1)}(x_i);$   
 $\text{calculează } \varphi_i^{(k+1)} = f_i + sum;$   
 $\}$   
 $k = k + 1;$   
 până când  $|\varphi_i^{(k+1)} - \varphi_i^{(k)}| < \varepsilon;$   
 $i = i + 1;$   
 până când  $i = n;$   
 tipărește soluția  $\varphi_i^{(k+1)}$   $i = 1$  până la  $n;$   
 $\}$

### 9.1.2. IMPLEMENTAREA ALGORITMULUI 9.1. METODA LUI FREDHOLM

```

{void Fredholm(double (*)(double),
               double (*)(double,double),
               double lims,
               double limd,
               int np,
               double lam,
               double sola[])
{
    double h,sum;
    int i,j,sem,cont;
    int niter=10000;
    double eps=1e-15;
    static double mat[NMax][NMax];

```

```

static double tl[NMax];
static double solp[NMax];
h=(limd-lims)/np;
for (i=1;i<=np+1;i++)
{
    tl[i]=f(lims+(i-1)*h);
    sola[i]=0;
    for(j=1;j<=np+1;j++)
    {
        if ( (j==1)/(j==np+1) ) mat[i][j]=0.5*lam*h*S(lims+(i-1)*h,lims+
            (j-1)*h);
        else mat[i][j]=lam*h*S(lims+(i-1)*h,lims+(j-1)*h);
    }
}
cont=0;
do
{
    sem=1;
    for(i=1;i<=np+1;i++)solp[i]=sola[i];
    for(i=1;i<=np+1;i++)
    {
        sum=0;
        for(j=1;j<=np+1;j++)if(j!=i)sum+=mat[i][j]*solp[j];
        sola[i]=(tl[i]+sum)/(1-mat[i][i]);
    }
    for(i=1;i<=np+1;i++) if(fabs(sola[i]-solp[i])>eps)sem=0;
    cont++;
}
while ((sem==0)&&(cont<niter))
}
}

```

## 9.2. INTEGRAREA ECUAȚIEI DE TIP VOLTERRA NEOMOGENĂ DE SPEȚA A DOUA PRIN METODA APROXIMĂRIILOR SUCCESIVE

Fie ecuația

$$\varphi(x) = f(x) + \int_a^x S(x, y) \varphi(y) dy. \quad (9.10)$$

Diferența finită pentru ecuația (9.10) este dată de relația :

$$\Delta\varphi = \varphi(x) - \varphi(x-h) = f(x) - f(x-h) + \int_{x-h}^x S(s, y) \varphi(y) dy \quad (9.11)$$

Aplicând o formulă de cuadratură pentru integrală pe intervalul  $h$  rezultă :

$$\varphi_i - \varphi_{i-1} = f_i - f_{i-1} + \sum_{j=0}^i C_{ij} \varphi_j \quad (9.12)$$

Din această relație prin explicitarea lui  $\varphi_i$  se obține

$$\varphi_i = \frac{1}{1 - C_{ii}} \left[ f_i - f_{i-1} + (1 + C_{i,i-1}) \varphi_{i-1} + \sum_{j=0}^{i-2} C_{ij} \varphi_j \right] \quad (9.13)$$

Din ecuația inițială (9.10) rezultă că  $\varphi(a) = f(a)$  deci  $\varphi_0 = f_0$ . Ca urmare, din ultima relație (9.13) rezultă :

$$\varphi_1 = \frac{1}{1 - C_{11}} \left[ f_1 - f_0 + (1 - C_{10}) \varphi_0 \right] = \frac{f_1 + C_{10} f_0}{1 - C_{11}} \quad (9.14)$$

Această valoare este utilizată ca valoare de start pentru ecuația de recurență (9.13) .

### 9.2.1. ALGORITM 9.2. METODA LUI VOLTERRA

{Variabile

$n$  : numărul de puncte în care se calculează soluția , întreg ;

$x$  : vectorul punctelor de pe  $Ox$  , real ;

$y$  : vectorul punctelor de pe  $Oy$  , real ;

$M$  : matricea punctelor formată de vectorul  $x$  și  $y$  ;

$C_{ij}$  : valorile funcției cunoscute  $S(x, y)$  în punctele în care se calculează prin metoda cuadraturii înmulțită cu ponderile, real ;

$f_i$  : valorile funcției  $f(x_i)$  , real ;

$h$  : intervalul de integrare , real ;

$\varphi_i$  : valorile funcției soluție , real ;

$\varepsilon$  : eroarea de calcul , real ;

$sum$  : suma parțială , real ;

{

$sum = 0$  ;

calculează  $f_1, f_0$  ;

calculează  $\varphi_1 = \frac{f_1 + C_{10} f_0}{1 - C_{11}}$  ;

$i = 1$  ;

repetă

pentru  $j = 0$  la  $i - 2$

{calculează  $sum = sum + C_{ij} \varphi_j$  ;

calculează  $\varphi_i = \frac{1}{1 - C_{ii}} \left[ f_i - f_{i-1} + (1 + C_{i,i-1}) \varphi_{i-1} + sum \right]$  ;

}

$i = i + 1$  ;

până când  $i = n$  ;

Valorile funcției soluție sunt  $\varphi_i$  pentru  $i = 1, \dots, n$  ;

}

### 9.2.2. IMPLEMENTAREA ALGORITMULUI 9.2

```
{void Volterra(double (*f)(double),
               double (*S)(double,double),
               double lambda,
               double lims,
               double h,
               int np,
               double sola[])
{
    int i,j;
    double sum;
    sola[0]=f(lims);
    sola[1]=(f(lims+h)+0.5*f(lims)*S(lims,lims)*lambda)
    /(10.5*S(lims+h,lims+h)*lambda);
    for(i=2;i<=np;i++)
    {
        sum=0;
        for(j=0;j<=i-2;j++)sum+=sola[j]*S(lims+i*h,lims+j*h)*lambda;
        sum=sum-0.5*sola[0]*S(lims,lims)*lambda;
        sola[i]=(f(lims+i*h)-f(lims+(i-1)*h)+(1+S(lims+i*h,lims+
        (i-1)*h))*lambda*sola[i-1]+sum)/(10.5*lambda*S(lims+i*h,lims+i*h));
    }
}
```

### 9.3. APLICAȚIE

Să se rezolve ecuația integrală de tip Fredholm

$$\varphi(x) = x - 0.05 \cdot x^2 - 0.025 + 0.1 \int_0^1 (x^2 + y^2) \cdot y dy$$

pentru care lims=0,limd=1,lambda=0.1, nr=5.

Rezultatele obținute sunt :

$$\varphi(0.0) = 0.001038, \quad \varphi(0.2) = 0.201042, \quad \varphi(0.4) = 0.401055,$$

$$\varphi(0.6) = 0.601076 \quad \varphi(0.8) = 0.801106, \quad \varphi(1.0) = 1.001145.$$

Aceste rezultate sunt valorile funcției soluției în punctele de diviziune ale intervalului

[0 1].

# 10

## VECTORI ȘI VALORI PROPRII\*

Se consideră o matrice pătrată  $M_R^{n \times n}$  de ordinul  $n$

$$M_R = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ - & - & - & - \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \quad (10.1)$$

și un vector  $X^T = [x_1, x_2, x_3, \dots, x_n]$  (10.2)

Problema care se pune este să determinăm valorile  $\lambda$  și vectorii  $X$  pentru care transformarea făcută de matrice asupra vectorului  $X$  să ne dea un vector  $\lambda X$  adică un vector coliniar cu el. Ca urmare  $A \cdot X = \lambda \cdot X$  (10.3)

sau scrisă matriceal :

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ - & - & - & - \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ -- \\ x_n \end{pmatrix} = \lambda \begin{pmatrix} x_1 \\ x_2 \\ -- \\ x_n \end{pmatrix} \quad (10.4)$$

Relațiile (10.4) se mai pot scrie și sub forma :

$$\begin{pmatrix} a_{11} - \lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & \dots & a_{2n} \\ - & - & - & - \\ a_{n1} & a_{n2} & \dots & a_{nn} - \lambda \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ -- \\ x_n \end{pmatrix} = 0 \quad (10.5)$$

Sistemul obținut (10.5) este un sistem omogen care admite întotdeauna soluția banală

$$x_1 = x_2 = x_3 = \dots = x_n = 0.$$

Suntem interesați de soluțiile diferite de zero ale sistemului. Pentru ca sistemul (10.5) să admită soluții diferite de soluțiile banale este necesar și suficient ca determinantul sistemului să fie nul. Determinantul în necunoscuta  $\lambda$  prezintă un polinom de grad  $n$  și se numește prin definiție *polinom caracteristic*, iar egalat cu zero poartă numele de *ecuație caracteristică* a matricei  $A$ . Ecuația (10.6) reprezintă polinomul caracteristic iar  $P(\lambda) = 0$  ecuația caracteristică. Pentru fiecare valoare proprie determinată din ecuația caracteristică se pot determina valori proprii care verifică ecuația  $AX = \lambda X$ . Pentru o valoare proprie  $\lambda$ , există o infinitate de vectori proprii.

---

\*Bibliografie : [6], [10], [11], [22], [23]

$$P(\lambda) = \begin{pmatrix} a_{11} - \lambda & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} - \lambda & \dots & a_{2n} \\ - & - & - & - \\ a_{n1} & a_{n2} & \dots & a_{nn} - \lambda \end{pmatrix} \quad (10.6)$$

Vectorii și valorile proprii sunt utili pentru simplificarea operațiilor cu matrice ce se întâlnesc în rezolvarea diferitelor sisteme de ecuații diferențiale și în alte operații mai complicate .

## 10.1. TIPURI DE MATRICE

În continuare sunt date definițiile unor tipuri de matrice, mai des utilizate în cercetare și proiectare și unele proprietățile mai importante a lor.

### 10.1.1. MATRICE SIMILARE

Două matrice  $A$  și  $B \in M_{n \times n}(S)$  unde  $S \subset \mathbb{R}$  sau  $S \subset \mathbb{C}$  (complex ) se numesc *similare* dacă există o matrice nesingulară  $P \in M_{n \times n}(S)$  astfel încât

$$B = P A P^{-1} \quad (10.7)$$

**Teorema1:** Două matrice similare au aceleași valori proprii. Matricea vectorilor proprii :

$$V = \begin{pmatrix} x_1^1 & x_1^2 & \dots & x_1^n \\ x_2^1 & x_2^2 & \dots & x_2^n \\ -- & -- & -- & -- \\ x_n^1 & x_n^2 & \dots & x_n^n \end{pmatrix} \quad (10.8)$$

se numește *matricea modală* iar matricea valorilor proprii  $A$  se numește *matricea diagonală*:

$$A = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ -- & -- & -- & -- \\ 0 & 0 & \dots & \lambda_n \end{pmatrix} \quad (10.9)$$

Relația între cele două matrice este :

$$A V = V \lambda \quad (\text{ecuația modală}) \quad (10.10)$$

$$\text{sau} \quad A = V \lambda V^{-1} \quad (10.11)$$

Matricea modală a vectorilor proprii și matricea diagonală a valorilor proprii sunt similare .

### 10.1.2. MATRICE SIMETRICE

O matrice pătratică  $A$  este *simetrică* dacă

$$A^T = A \quad (10.12)$$

(adică transpusa este egală cu matricea ) .



**Teorema 2:** Valorile proprii ale unei matrice simetrice  $A \in M_R^{n \times n}$  sunt reale .

**Corolar 1:** Dacă  $A \in M_R^{n \times n}$  este simetrică, atunci există o transformare similară  $P A P^{-1} = \Lambda$  unde  $P \in M_R^{n \times n}$  este o matrice ortogonală și  $\Lambda \in M_\Gamma^{n \times n}$  este diagonală .

### 10.1.3. MATRICE ORTOGONALE

O matrice pătratică  $A$  se numește *ortogonală* dacă

$$A^T = A^{-1} \quad (10.13)$$

Aceste matrice sunt utile pentru transformările similare .

### 10.1.4. MATRICE SUPERIOR TRIUNGHIULARE

Sunt matricele cu toate elementele nule sub diagonală principală. Pentru aceste matrice valorile proprii sunt elementele de pe diagonală .

### 10.1.5. MATRICE INFERIOR TRIUNGHIULARE

Sunt matricele cu toate elementele nule deasupra diagonalei principale. Valorile proprii sunt elementele de pe diagonală .

### 10.1.6. MATRICEA DIAGONALĂ

Este matricea cu toate elementele nule deasupra și dedesubtul diagonalei principale. Valorile proprii sunt elementele de pe diagonală .

### 10.1.7. MATRICEA HERMITICĂ

Matricea  $A$  este *hermitică* dacă  $A = A^H$  unde

$$a_{ij} = a_{ji}^*,$$

iar pentru elementele de pe diagonală principală avem

$$\bar{a}_{ii} = a_{ii},$$

adică sunt reale .

Dacă  $A \in M_\Gamma^{n \times n}$  este o matrice hermitică triunghiulară, atunci pentru orice  $X \in C$  expresia  $X^H A X$  este reală .

Dacă  $A$  este matricea hermitică, atunci :

- este pozitiv definită dacă pentru  $X \neq \Phi$  rezultă  $X^H A X > 0$
- este semipozitiv definită dacă pentru  $X \neq \Phi$  rezultă  $X^H A X \geq 0$  .

### 10.1.8. MATRICEA SINGULARĂ

$A \in M_{\Gamma}^{n \times n}$  este o matrice *singulară* dacă și numai dacă admite o valoare proprie nulă. Prin definiție *urma* unei matrice  $A \in M_{\Gamma}^{n \times n}$  este suma elementelor de pe diagonala principală :

$$\text{tr } A = \sum_{i=1}^n a_{ii} \quad (10.14)$$

Prin definiție *spectrul radial*  $S(A)$  al unei matrice  $A \in M_{\Gamma}^{n \times n}$  este dat de valoarea  $|\lambda_i|$  unde  $\lambda_i$  este valoarea absolută maximă dintre valorile proprii .

## 10.2. LOCALIZAREA VALORILOR PROPRII

**Teorema 3** : Fiecare valoare proprie a unei matrice  $A \in M_{\Gamma}^{n \times n}$  se găsește în cel puțin un disc circular,  $L_i$  cu centrul în  $a_{ii}$  și rază  $\gamma_i$  unde

$$\gamma_i = l_i = \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}| \quad i=1,2,\dots,n \quad (10.15)$$

sau un disc circular  $C_j$  cu centrul în  $a_{jj}$  și rază  $c_j$

$$\gamma_j = c_j = \sum_{\substack{i=1 \\ i \neq j}}^n |a_{ij}| \quad j=1,2,\dots,n \quad (10.16)$$

Toate valorile proprii să se găsească în reuniunea cercurilor :

$$L_i = \{z, |z - a_{ii}| \leq l_i = \gamma_i\} \quad i=1,2,\dots,n \quad (10.17)$$

sau în reuniunea cercurilor :

$$C_j = \{z, |z - a_{jj}| \leq c_j = \gamma_j\} \quad j=1,2,\dots,n \quad (10.18)$$

## 10.3. METODE DE DETERMINARE A VALORILOR ȘI VECTORILOR PROPRII AI UNEI MATRICE

Valorile proprii ale unei matrice  $A$ , deci soluțiile ecuației ei caracteristice pot fi reale sau complexe. Din acest punct de vedere clasificăm matricele în :

a) Matrice hermitice care admit numai valori proprii reale, iar vectorii asociați sunt distincți și ortogonali. Aceste matrice pot fi diagonalizate cu matrice ortogonale .

b) Matrice nehermitice ale căror valori proprii nu sunt toate reale, iar vectorii asociați nu au proprietăți particulare .

### 10.3.1. METODE DE DETERMINARE A VALORILOR ȘI VECTORILOR PROPRII ALE MATRICELOR HERMITICE

Dintre aceste metode se prezintă metoda puterii, metoda lui Krîlov, metoda Householder, metoda RT și metoda LR.

#### 10.3.1. METODA PUTERII

Considerăm ecuația (10.3) în care  $\lambda$  reprezintă valoarea proprie iar  $X$  vectorul propriu al matricei  $A$ . Pentru rangul  $n$  al matricei  $A$  presupunem existența a  $n$  valori proprii distincte  $\lambda$  și a  $n$  vectori proprii  $X$ , independenți. Rezultă că un vector oarecare  $Y$  din spațiul  $n$  poate fi scris ca o combinație liniară de cei  $n$  vectori proprii ai matricei  $A$ :

$$Y = \alpha_1 x_1 + \alpha_2 x_2 + \dots + \alpha_n x_n \quad (10.19)$$

Înmulțim ecuația (10.19) cu matricea  $A$  și rezultă:

$$AY = A\alpha_1 x_1 + A\alpha_2 x_2 + \dots + A\alpha_n x_n = \alpha_1 \lambda_1 x_1 + \alpha_2 \lambda_2 x_2 + \dots + \alpha_n \lambda_n x_n \quad (10.20)$$

Continuăm înmulțirea ecuației (10.20) cu  $A$  până la un rang  $k$  când se obține următoarea ecuație:

$$A^k Y = A^k \alpha_1 x_1 + A^k \alpha_2 x_2 + \dots + A^k \alpha_n x_n = \alpha_1 \lambda_1^k x_1 + \alpha_2 \lambda_2^k x_2 + \dots + \alpha_n \lambda_n^k x_n \quad (10.21)$$

Ecuația (10.21) se mai poate scrie și astfel:

$$A^k Y = \lambda_1^k (\alpha_1 x_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k x_2 + \dots + \alpha_n \left(\frac{\lambda_n}{\lambda_1}\right)^k x_n) \approx \lambda_1^k \alpha_1 x_1 \quad (10.22)$$

dacă valoarea lui  $\lambda_1$  este cea mai mare valoare proprie și  $k$  este mare.

Se poate face aceeași aproximare și pentru ecuația:

$$A^{k-1} Y = \lambda_1^{k-1} (\alpha_1 x_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1}\right)^{k-1} x_2 + \dots + \alpha_n \left(\frac{\lambda_n}{\lambda_1}\right)^{k-1} x_n) \approx \lambda_1^{k-1} \alpha_1 x_1 \quad (10.23)$$

Din împărțirea ecuației (10.22) la (10.23) rezultă valoarea proprie maximă pentru matricea  $A$ . Vectorul  $Y$  este un vector coloană și prin împărțirea vectorilor corespunzători ecuațiilor (10.22) și (10.23) vom realiza împărțirea componentelor vectorilor a căror valoare este aproximativ egală cu valoarea proprie maximă.

Dacă matricea  $A$  este nesingulară atunci ecuația (10.3) mai poate fi scrisă și astfel:

$$A^{-1} X = \lambda^{-1} X \quad (10.24)$$

Calculând valoarea caracteristică maximă pentru ecuația (10.24) rezultă valoarea caracteristică minimă a matricei  $A$ . Cu această metodă se poate determina cea mai mare și cea mai mică valoare caracteristică a unei matrice date.

#### 10.3.1.1. Algoritm 10.1. Metoda puterii

```

{Variabile
n,k:rangul matricei,contor, întregi;
A:matrice;
y:vector;
p:vectorul produs;
 $\Lambda$ :vectorul valorilor proprii;
{
p[0]=y
pentru i=1,..., k
    /* presupunem valoarea maximă a valorilor proprii  $\lambda_1$  */
    calculează p[i]=A*p[i-1];
    calculează  $\lambda[1]=p[k]/p[k-1]$ ;
    /* împărțirea vectorilor constă în împărțirea componentelor */
}
}

```

### 10.3.1.2. Implementarea algoritmului 10.1. Metoda puterii

```

/* Funcția întoarce :
0 dacă s-a găsit valoarea proprie cu precizia dorită,
1 dacă s-a găsit o valoare proprie, algoritmul terminându-se
după numărul de iterații specificat.
*/
int PowerDirVP(int or_mat, /*ordinul matricei */
double MAT[][NrMax], /*matricea */
double VECS[NrMax], /*vectorul de start */
double eps, /* precizia */
int maxiter, /* numărul maxim de iterații */
double *valpr /* valorile propri */
)
{
int i,j,k,sem;
static double XN[NrMax],XN_1[NrMax];
k=0;
for(i=1;i<=or_mat;i++)XN[i]=VECS[i];
do
{sem=1;
k++;
for(i=1;i<=or_mat;i++)XN_1[i]=XN[i];
for(i=1;i<=or_mat;i++)
{
XN[i]=0;
for(j=1;j<=or_mat;j++)XN[i]+=MAT[i][j]*XN_1[j];
}
}
}

```

```

for(i=1;i<=or_mat;i++)if(XN_I[i]==0)sem=0;
if(sem!=0)for(i=2;i<=or_mat;i++)
    if( fabs( XN[1]/XN_I[1]-XN[i]/XN_I[i]) >eps)sem=0;
}while( (sem==0) && (k<maxiter) );
for(i=1;i<=or_mat;i++)VECS[i]=XN[i];
if(k>=maxiter)return 1;
*valpr=XN[1]/XN_I[1];
return 0;
}

```

### 10.3.2. METODA LUI KRÎLOV

Această metodă determină coeficienții polinomului caracteristic. Dacă se consideră polinomul caracteristic

$$p(\lambda) = \sum_{i=0}^n c_i \lambda^i \quad \text{cu} \quad c_n = 1 \quad (10.25)$$

și substituim  $\lambda=A$  obținem ecuația:  $P(A)=0$  (10.26)

Se alege un vector de start  $X_0$  cu care înmulțim ecuația obținută (10.26) și se obține

$$\text{ecuația:} \quad \sum_{i=0}^n c_i A^i X_0 = 0 \quad \text{cu condițiile} \quad c_n = 1; \quad A^0 = I \quad (10.27)$$

Notăm  $A^i X_0 = X_i$  unde componentele lui  $X_i$  sunt calculate iterativ din  $X_0$  astfel:  $X_1 = AX_0; X_2 = AX_1; \dots, X_i = AX_{i-1};$  (10.28)

și au formulele de calcul:

$$x_{1j} = \sum_{k=1}^n a_{jk} x_{0k} \quad ; \quad j = 1 \dots n; \quad x_{2j} = \sum_{k=1}^n a_{jk} x_{1k} \quad ; \quad j = 1 \dots n; \quad \dots, x_{ij} = \sum_{k=1}^n a_{jk} x_{i-1,k} \quad ; \quad j = 1 \dots n \quad (10.29)$$

$$\text{Rezultă sistemul:} \quad \sum_{k=0}^{n-1} x_{ki} c_k = -x_{ni}; \quad i = 1 \dots n. \quad (10.30)$$

sau scris matriceal

$$\begin{pmatrix} x_{01} & x_{11} & x_{21} & \dots & x_{n-1,1} \\ x_{02} & x_{12} & x_{22} & \dots & x_{n-1,2} \\ \dots & \dots & \dots & \dots & \dots \\ x_{0n} & x_{1n} & x_{2n} & \dots & x_{n-1,n} \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ c_2 \\ \dots \\ c_n \end{pmatrix} = \begin{pmatrix} x_{n1} \\ x_{n2} \\ \dots \\ x_{nn} \end{pmatrix} \quad (10.31)$$

unde  $C_i$  reprezintă coeficienții polinomului caracteristic al matricei  $A$ .

Din rezolvarea sistemului se obțin coeficienții polinomului caracteristic al matricei  $A$ .

#### 10.3.2.1. Algoritmul 10.2. Metoda lui Krîlov

{Variabile

```

n:ordinul matricei, întreg;
A:matrice;
V:vectorul de start ales;
C:vectorul coloană al coeficienților polinomului caracteristic;
X: vector, produs de matrice A cu vector;
i,j,k:contori, întregi;
{X[0]=V;
pentru i=1, ... , n
  calculează X[i]=A*X[i-1];
formează sistemul [X[0],X[1],...,X[n-1]]*C=-X[n];
Rezolvă sistemul;
tipărește soluțiile sistemului C[i]; i=1,...,n ;
}
}

```

### 10.3.2.2. Implementarea algoritmului 10.2. Metoda lui Krîlov

```

/*Funcția care implementează metoda lui Krilov pentru
determinarea coeficienților polinomului caracteristic
*/
int Krilov(int ord,
           double mat[][NrMax],
           double X[],
           double pol[])
{
  static double A[NrMax][NrMax];
  static double TL[NrMax];
  static double V[NrMax];
  int i,j,k;
  for(i=1;i<=ord;i++)A[i][ord]=X[i];
  for(i=1;i<=ord-1;i++)
  {
    for(j=1;j<=ord;j++)V[j]=X[j];
    for(k=1;k<=i;k++)ProdMatVect(ord,mat,V);
    for(j=1;j<=ord;j++)A[j][ord-i]=V[j];
  }
  for(i=1;i<=ord;i++)TL[i]=X[i];
  for(i=1;i<=ord;i++)ProdMatVect(ord,mat,TL);
  for(i=1;i<=ord;i++)TL[i]=-TL[i];
  for(i=1;i<=ord;i++)
  {
    for(j=1;j<=ord;j++)
      printf("%5.3lf ",A[i][j]);
    printf("\n");
  }
}

```

```

getche();
for(i=1;i<=ord;i++) printf("%5.3lf\n",TL[i]);
getche();
if(GAUSS(ord,A,TL,V)==0) return 0;
for(i=1;i<=ord;i++) pol[i]=V[i];
return 1;
}

```

### 10.3.3. METODA LUI HOUSEHOLDER

Această metodă determină valorile și vectorii proprii ai unei matrice simetrice  $A^T = A$ . Mai întâi matricea  $A$  este adusă cu ajutorul transformărilor ortogonale la o matrice tridiagonală similară cu  $A$ . Se determină valorile proprii ale matricei tridiagonale, care sunt identice cu cele ale matricei  $A$ , prin metoda șirului lui Șurm și prin metoda bisecției. Se trece la determinarea vectorilor proprii pentru matricea tridiagonală din care se determină vectorii proprii ai matricei  $A$ .

Tridiagonalizarea se realizează cu ajutorul transformărilor ortogonale de tipul  $B^{(k)} = E - 2W^{(k)} \cdot (W^{(k)})^T$  unde  $(W^{(k)})^T = (0, \dots, 0, w_{k+1}^{(k)}, \dots, w_n^{(k)})$ ;  $(W^{(k)})^T \cdot W^{(k)} = 1$  (10.32)

Matricea  $B^{(k)}$  este o matrice simetrică și ortogonală. Detaliat matricea se prezintă ca în expresia (10.33).

$$B^{(k)} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 & & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 & & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 - 2(w_{k+1}^{(k)})^2 & -2w_{k+1}^{(k)}w_{k+2}^{(k)} & -2w_{k+1}^{(k)}w_{k+3}^{(k)} & \dots & -2w_{k+1}^{(k)}w_n^{(k)} \\ 0 & 0 & 0 & \dots & -2w_{k+2}^{(k)}w_{k+1}^{(k)} & 1 - 2(w_{k+2}^{(k)})^2 & -2w_{k+2}^{(k)}w_{k+3}^{(k)} & \dots & -2w_{k+2}^{(k)}w_n^{(k)} \\ 0 & 0 & 0 & \dots & -2w_{k+3}^{(k)}w_{k+1}^{(k)} & -2w_{k+3}^{(k)}w_{k+2}^{(k)} & 1 - 2(w_{k+3}^{(k)})^2 & \dots & -2w_{k+3}^{(k)}w_n^{(k)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & -2w_n^{(k)}w_{k+1}^{(k)} & -2w_n^{(k)}w_{k+2}^{(k)} & -2w_n^{(k)}w_{k+3}^{(k)} & \dots & 1 - 2(w_n^{(k)})^2 \end{pmatrix}$$

$$k=1, 2, \dots, n-2. \quad (10.33)$$

Din produsul  $A^{(k-1)} \cdot B^{(k)}$ , punând condiția ca matricea pe linia  $k$  și coloana  $k$  să aibă numai primii doi termeni rezultă formulele de calcul ale elementelor  $w_i^{(k)}$ ;  $i = k, \dots, n$ . Se ține seama că suma pătratelor elementelor unei linii a unei matrice  $A$  este invariantă la transformarea similară ortogonală  $(B^{(k)})^T \cdot A \cdot B^{(k)}$ . În aceste condiții, se obțin următoarele formule de calcul:  
pentru  $k=1, 2, \dots, n-2$

$$\begin{aligned}
sum &= \sum_{j=k+1}^n a_{kj}^2, \quad (w_i^{(k)} = 0, \text{ pentru } i = 1, \dots, k); \\
w_{k+1}^{(k)} &= \sqrt{\frac{1}{2} \left( 1 + \frac{a_{k,k+1} \cdot \text{sign}(a_{k,k+1})}{\sqrt{sum}} \right)}; \\
w_i^{(k)} &= \frac{a_{ki} \cdot \text{sign}(a_{k,k+1})}{2 \cdot w_{k+1}^{(k)} \cdot \sqrt{sum}}; \quad i = k+2, \dots, n;
\end{aligned} \tag{10.34}$$

$$t_i = \sum_{j=k+1}^n a_{ij} \cdot w_j^{(k)}, \quad i = k+1, \dots, n;$$

$$s = \sum_{i=k+1}^n w_i^{(k)} \cdot t_i;$$

$$r_i = t_i - s \cdot w_i^{(k)}; \quad i = k+1, \dots, n;$$

$$\alpha_k = a_{kk};$$

$$\beta_k = -\text{sign}(a_{k,k+1}) \cdot \sqrt{sum};$$

Când se ajunge la pasul  $n-2$ , tridiagonalizarea ia sfârșit și se consideră:

$$a_{n-1} = a_{n-1,n-1}; \quad a_n = a_{nn}; \quad \beta_{n-1} = a_{n-1,n};$$

Matricea  $A^{(n-2)}$  este tridiagonală iar matricea  $W$  se prezintă astfel:

$$W = (W^{(1)}, W^{(2)}, \dots, W^{(n-2)}) = \begin{pmatrix} 0 & 0 & 0 & 0 & \dots & 0 \\ w_2^{(1)} & 0 & 0 & 0 & \dots & 0 \\ w_3^{(1)} & w_3^{(2)} & 0 & 0 & \dots & 0 \\ w_4^{(1)} & w_4^{(2)} & w_4^{(3)} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ w_n^{(1)} & w_n^{(2)} & w_n^{(3)} & w_n^{(4)} & \dots & w_n^{(n-2)} \end{pmatrix} \tag{10.35}$$

Valorile proprii ale matricei  $A$  se calculează ca soluțiile ecuației:

$$\begin{vmatrix} \lambda - \alpha_1 & -\beta_1 & 0 & 0 & \dots & 0 \\ -\beta_1 & \lambda - \alpha_2 & -\beta_2 & 0 & \dots & 0 \\ 0 & -\beta_3 & \lambda - \alpha_3 & -\beta_3 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & \lambda - \alpha_n \end{vmatrix} = 0 \tag{10.36}$$

Din ecuația (10.36) se pot calcula minorii principali de la ordinul 1 până la ordinul  $n$  cu formulele:

$$\begin{aligned}
f_1(\lambda) &= \lambda - \alpha_1; \\
f_2(\lambda) &= (\lambda - \alpha_2) \cdot f_1(\lambda) - \beta_1^2; \\
f_i(\lambda) &= (\lambda - \alpha_i) \cdot f_{i-1}(\lambda) - \beta_{i-1}^2 \cdot f_{i-2}(\lambda), \quad i = 3, \dots, n.
\end{aligned} \tag{10.37}$$

Expresiile minorilor principali (10.37) formează un șir Sturm asociat lui  $f_n(\lambda)$  ce reprezintă polinomul caracteristic al matricei tridiagonale și al matricei inițiale date.



Numărul de variații de semn ale șirului lui Șturm dă numărul de valori proprii reale ale matricei tridiagonale. Se calculează :

$$L = \max\{|\alpha_1| + |\beta_1|, |\beta_{i-1}| + |\alpha_i| + |\beta_i| \quad i = 2, 3, \dots, n-1, \quad |\beta_{n-1}| + |\alpha_n|\} \quad (10.38)$$

Valorile caracteristice sunt cuprinse în intervalul  $(-L, L)$ . Ca urmare pentru  $n$  valori caracteristice reale, numărul de variații de semn ale șirului lui Șturm în punctul  $-L$  este  $n$ , iar în  $L$  este zero. Dacă se consideră intervalul  $(I_i, S_i)$  în care este situată soluția  $\lambda_i$  se pornește cu  $I_i = -L$  ;  $S_i = l$  pentru toate soluțiile. Se aplică metoda biseecției până când  $|S_i - I_i| < \varepsilon$ ,  $\varepsilon$  fiind eroarea dată.

Vectorii proprii ai matricei tridiagonale se calculează prin rezolvarea sistemului  $A^{n-2} \cdot V = \lambda V$  unde  $\lambda$  ia valorile calculate.

$$\begin{aligned} v_1^{(j)} &= 1; \\ v_2^{(j)} &= \frac{1}{\beta_1}(\lambda_i - \alpha_1); \quad j = 1, \dots, n; \quad k = 2, \dots, n-1; \\ v_{k+1}^{(j)} &= \frac{1}{\beta_k}(\lambda_i - \alpha_k) \cdot v_k^{(j)} - \beta_{k-1} \cdot v_{k-1}^{(j)}; \end{aligned} \quad (10.39)$$

Pentru calculul vectorilor proprii ai matricei inițiale  $A$  se ține seama de relația de calcul a matricei tridiagonale:

$$A^{(n-2)} = B^{(n-2)} \dots B^{(2)} \cdot B^{(1)} \cdot A \cdot B^{(1)} \cdot B^{(2)} \dots B^{(n-2)} \quad (10.40)$$

Din ecuația

$$A^{(n-2)} \cdot V^{(j)} = B^{(n-2)} \dots B^{(2)} \cdot B^{(1)} \cdot A \cdot B^{(1)} \cdot B^{(2)} \dots B^{(n-2)} \cdot V^{(j)} = \lambda_j \cdot V^{(j)} \quad (10.41)$$

$$\text{rezultă:} \quad A \cdot B^{(1)} \cdot B^{(2)} \dots B^{(n-2)} \cdot V^{(j)} = \lambda_j \cdot B^{(1)} \cdot B^{(2)} \dots B^{(n-2)} \cdot V^{(j)} \quad (10.42)$$

$$\text{Ecuația (10.42) se reduce la ecuația : } A \cdot X^{(j)} = \lambda_j \cdot X^{(j)} \quad (10.43)$$

unde  $X^{(j)}$  reprezintă vectorii proprii ai matricei  $A$ .

Făcând produsul iterativ se obține:

$$V^{(j)} = B^{(n-2)} \cdot V^{(j)}; \quad V^{(j)} = B^{(n-3)} \cdot V^{(j)} \dots V^{(j)} = B^{(1)} \cdot V^{(j)} \quad (10.44)$$

Pentru prima ecuație din (10.44) rezultă:

$$\begin{pmatrix} v_1^{(j)} \\ v_2^{(j)} \\ - \\ v_{n-2}^{(j)} \\ v_{n-1}^{(j)} \\ v_n^{(j)} \end{pmatrix} = \begin{pmatrix} 1 \\ v_2^{(j)} \\ - \\ v_{n-2}^{(j)} \\ v_{n-1}^{(j)} \\ v_n^{(j)} \end{pmatrix} - 2 \begin{pmatrix} 0 \\ 0 \\ - \\ 0 \\ w_{n-1}^{(j)} \\ w_n^{(j)} \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & \dots & 0 & w_{n-1}^{(j)} & w_n^{(j)} \end{pmatrix} \begin{pmatrix} 1 \\ v_2^{(j)} \\ - \\ 0 \\ v_{n-1}^{(j)} \\ v_n^{(j)} \end{pmatrix}. \quad (10.45)$$

Dacă notăm  $\text{suma} = \sum_{i=k+1}^n w_i^{(k)} \cdot v_i^{(j)}$ , atunci vectorii proprii sunt:

$$v_i^{(j)} = v_i^{(j)} - 2 \cdot \text{suma} \cdot w_i^{(k)} \text{ pentru } k=n-2, \dots, 1; i=k+1, \dots, n; j=1, \dots, n; \quad (10.46)$$

### 10.3.3.1. Algoritmul 10.3. Metoda lui Householder

*{Variabile*  
*n:ordinul matricei, întreg;*  
*A:matrice simetrică;*  
*V:vectorul ;*  
*X:vectorii proprii;*  
*W:matrice cu coloane vectori;*  
*i,j,k:contori, întregi;*  
*λ:vectorul valorilor proprii;*  
*α:vectorul elementelor de pe diagonala principală a matricei*  
*tridiagonale;*  
*β:vectorul elementelor de pe prima și a doua diagonală;*  
*t,r:vectori;*  
*p,sum,suma:variabile, reale;*  
*ε:eroare de calcul, reală;*  
*{*  
*pentru k=1,..., n-2*  
*{calculează*  $sum = \sum_{j=k+1}^n a_{kj}^2$  *, (* $w_i^{(k)} = 0$ *, pentru*  $i = 1, \dots, k$ *);*  
  
*calculează*  $w_{k+1}^{(k)} = \sqrt{\frac{1}{2} \left( 1 + \frac{a_{k,k+1} \cdot \text{sign}(a_{k,k+1})}{\sqrt{sum}} \right)}$ ;  
*pentru i=k+2, ... , l*  
  
*{  calculează*  $w_i^{(k)} = \frac{a_{ki} \cdot \text{sign}(a_{k,k+1})}{2 \cdot w_{k+1}^{(k)} \cdot \sqrt{sum}}$  *;*  
  
*calculează*  $t_i = \sum_{j=k+1}^n a_{ij} \cdot w_j^{(k)}$  *,*  
  
*calculează*  $s = \sum_{i=k+1}^n w_i^{(k)} \cdot t_i$ ;  
  
*calculează*  $r_i = t_i - s \cdot w_i^{(k)}$  *;*  
*}*  
  
*calculează*  $a_k = a_{kk}$  *;*  
  
*calculează*  $b_k = -\text{sign}(a_{k,k+1}) \cdot \sqrt{sum}$  *;*  
*}*  
*calculează*  $\alpha_{n-1} = a_{n-1,n-1}$  *;*

calculează  $\alpha_n = a_{nn}$  ;

calculează  $\beta_{n-1} = a_{n-1,n}$  ;

pentru  $i = 2, \dots, n-1$

calculează

$$f_1(l) = l - a_1 ;$$

$$f_2(l) = (l - a_2) \cdot f_1(l) - b_1^2 ;$$

$$f_i(l) = (l - a_i) \cdot f_{i-1}(l) - b_{i-1}^2 \cdot f_{i-2}(l);$$

pentru  $i=2, \dots, n-1$

calculează

$$L = \max(|a_1| + |b_1|, |b_{i-1}| + |a_1| + |b_i|, \dots, |b_{n-1}| + |a_n|)$$

pentru  $i=1, \dots, n$

{

$$I_i = -L;$$

$$S_i = L;$$

}

determină numărul de variații de semn al șirului;

pentru  $k=i$  până la  $m$  compară  $k$  cu  $nv$  și modifică

$I_k$  sau  $S_k$

Aplică metoda biseției și calculează  $l_i, i=1, \dots, n$ ;

pentru  $j=1, \dots, n$

{

calculează

$$v_1^{(j)} = 1;$$

$$v_2^{(j)} = \frac{1}{b_1} (l_1 - a_1);$$

$$v_{k+1}^{(j)} = \frac{1}{b_k} ((l_i - a_k) \cdot v_k^j - b_{k-1} \cdot v_{k-1}^{(j)});$$

}

pentru  $k=n-2, \dots, 1$

pentru  $j=1, \dots, n$

{

calculează

$$suma = \sum_{i=k+1}^n w_i^{(k)} \cdot v_i^{(j)} ;$$

$$v_i^{(j)} = v_i^{(j)} - 2 \cdot suma \cdot w_i^{(k)} ;$$

}

}

}

}

**10.3.3.2. Implementarea algoritmului 10.3**

```

Funcția semnătură
*/
int SGN(double x)
{
    if(x>0) return 1;
    if(x<0) return -1;
    return 0;
}
/*
    Funcția care întoarce numărul de variații de semn al polinoamelor
    Șturm într-un punct dat
*/
int Nr_var(int ord, /* ordinul matricei*/
           double valfa[],
           double vbeta[],
           /* Coeficienții alfa și beta ai matricei tridiagonale */
           double Med)
{
    int i,nv;
    double q;
    double eps=0.00001;
    nv=0;
    q=valfa[1]-Med;
    if(q<=0)nv++;
    for(i=2;i<=ord;i++)
    {
        if(q!=0)q=valfa[i]-Med-vbeta[i-1]*vbeta[i-1]/q;
        else q=valfa[i]-Med-fabs(vbeta[i-1])/eps;
        if(q<0) nv++;
    }
    return nv;
}
/*
    Funcția care implementează metoda Householder pentru aflarea
    valorilor proprii.
*/
void HOW( int n,
          double mat[][NrMax],
          /* matricea simetrică */
          double Lambda[])
    /* vectorul valorilor proprii */
{
    int i,j,k,VB1,VB2,nv,r;

```

```

double spat,c;
double static W[NrMax][NrMax];
static double p[NrMax];
static double q[NrMax];
static double alfa[NrMax];
static double beta[NrMax];
double N0,T,M,LAM,suma;
static double S[NrMax];
static double D[NrMax];
double eps=0.000001;
static double v[NrMax][NrMax];
for(k=1;k<=n-2;k++) /* ciclul mare */
{
    spat=0;
    for(j=k+1;j<=n;j++)spat=spat+mat[k][j]*mat[k][j];
    for(i=1;i<=n;i++)W[i][k]=0;
    W[k+1][k]=sqrt( 0.5*(1+mat[k][k+1]*SGN(mat[k][k+1])/sqrt(spat) )
);

    for(i=k+2;i<=n;i++)
    W[i][k]=mat[k][i]*SGN(mat[k][k+1])/(2*W[k+1][k]*sqrt(spat));
    for(i=k+1;i<=n;i++)
    {
        p[i]=0;
        for(j=k+1;j<=n;j++)p[i]+=mat[i][j]*W[j][k];
    }
    c=0;
    for(i=k+1;i<=n;i++)c=c+W[i][k]*p[i];
    for(i=k+1;i<=n;i++)q[i]=p[i]-c*W[i][k];
    alfa[k]=mat[k][k];
    beta[k]=-SGN(mat[k][k+1])*sqrt(spat);
    mat[k][k+1]=mat[k+1][k]=beta[k];
    for(j=k+2;j<=n;j++)
    { mat[k][j]=0;
      mat[j][k]=0;
    }
    for(i=k+1;i<=n;i++)
    for(j=k+1;j<=n;j++)
        mat[i][j]=mat[i][j]-2*W[i][k]*q[j]-2*q[i]*W[j][k];
}
alfa[n-1]=mat[n-1][n-1];
alfa[n]=mat[n][n];
beta[n-1]=mat[n-1][n];
/* ..... */
/* Se calculează intervalul (-N0 , N0) în care se află valorile proprii */
N0=fabs(alfa[1])+fabs(beta[1]);
for(i=2;i<=n-1;i++)

```

```

{
    T=fabs(beta[i-1])+fabs(alfa[i])+fabs(beta[i]);
    if(N0<T) N0=T;
}
T=fabs(beta[n-1])+fabs(alfa[n]);
if(N0<T) N0=T;
/*.....*/
/*Inițializarea limitei stângă si dreaptă a intervalului în care se
găsesc valorile proprii */
for(i=1;i<=n;i++)
{
    S[i]=-N0;
    D[i]=N0;
}
/* M mijlocul intervalului - Se va aplica biseția */
for(i=1;i<=n;i++)
{ while( fabs(D[i]-S[i])>eps )
    { M=(D[i]+S[i])/2;
      nv=Nr_var(n,alfa,beta,M);
      if(nv>=i) D[i]=M;
      else S[i]=M;
    }
}
for(j=1;j<=n;j++) Lambda[j]=D[j];
}

```

### 10.3.4. METODA RT

Această metodă utilizează tridiagonalizarea matricei simetrice prin metoda Householder, calculează coeficienții polinomului caracteristic și rezolvă ecuația caracteristică cu metoda Birge -Vieta, determinând astfel valorile proprii ale matricei date. Din ecuația (10.36) se calculează coeficienții polinomului caracteristic în mod iterativ după următoarele formule:

$$\begin{aligned}
 1; \quad a_1 &= -\alpha_1; \\
 1; \quad a_2 &= a_1 - \alpha_2; \quad b_2 = -\alpha_2 a_1 - \beta_1^2; \\
 1; \quad a_3 &= a_2 - \alpha_3; \quad b_3 = -\alpha_3 a_2 + b_2 - \beta_2^2; \quad c_3 = -\alpha_3 b_2 - \beta_2^2 a_1; \\
 1; \quad a_4 &= a_3 - \alpha_4; \quad b_4 = -\alpha_4 a_3 + b_3 - \beta_3^2; \quad c_4 = -\alpha_4 b_3 + c_3 - \beta_3^2 a_2; \quad d_4 = -\alpha_4 c_3 - \beta_3^2 b_2;
 \end{aligned}$$

-----  
(10.47)

#### 10.3.4.1. Algoritmul 10.4. Metoda RT

```

{Variabile
n:ordinul matricei, întreg;
A:matricea simetrică;
α:vectorul elementelor de pe diagonala principală, tridiagonală;
β:vectorul elementelor de pe diagonalele simetrice, tridiagonale;
i,j:contori, întregi;
C:matricea de calcul a coeficienților polinomului caracteristic;
{pentru i=1, ... , n
    Ci0 = 1;
    pentru j=1, ... , n
        Cij = 0;

        C11 = α1;
        C22 = α2·α1 - β12;
        pentru i=3, ... , n
            calculează Cii = -αi·Ci-1,i-1 - βi-12·Ci-2,i-2;
        pentru i=2, ... , n
            calculează Ci1 = Ci-1,1 - αi;
        pentru i=3, ... , n
            calculează Ci2 = -αi·Ci-1,1 + Ci-1,2 - βi-12;
        pentru j=3, ... , n
            pentru i=j+1, ... , n
                calculează
                    Cij = -αi·Ci-1,j-1 + Ci-1,j - βi-12·Ci-2,j-2;
        coeficienții polinomului caracteristic sunt

        θi = Cn,n-i; i = 1,2,...,n;
    }
}

```

#### 10.3.4.2. Implementarea algoritmului 10.4

```

/*
  Funcția care implementează metoda Householder pentru
  aflarea vectorilor și valorilor proprii ale unei matrice simetrice
*/
void HOW2( int n,
           double mat[][NrMax],
           double Lambda[])
{

```

```

    int i,j,k,VB1,VB2,nv,r;
    double spat,c;
    double static W[NrMax][NrMax];
    static double p[NrMax];
    static double q[NrMax];
    static double alfa[NrMax];
    static double beta[NrMax];
    double N0,T,M,LAM,suma;
    static double S[NrMax];
    static double D[NrMax];
    double eps=0.000001;
    static double v[NrMax][NrMax];
    static double Coef[NrMax][NrMax];
    static double Pol[NrMax];
    for(k=1;k<=n-2;k++) /* ciclul mare */
    {
        spat=0;
        for(j=k+1;j<=n;j++) spat=spat+mat[k][j]*mat[k][j];
        for(i=1;i<=n;i++) W[i][k]=0;
        W[k+1][k]=sqrt( 0.5*(1+mat[k][k+1]*SGN(mat[k][k+1])/sqrt(spat) )
    );

        for(i=k+2;i<=n;i++)
        W[i][k]=mat[k][i]*SGN(mat[k][k+1])/(2*W[k+1][k]*sqrt(spat));
        for(i=k+1;i<=n;i++)
        {
            p[i]=0;
            for(j=k+1;j<=n;j++) p[i]+=mat[i][j]*W[j][k];
        }
        c=0;
        for(i=k+1;i<=n;i++) c=c+W[i][k]*p[i];
        for(i=k+1;i<=n;i++) q[i]=p[i]-c*W[i][k];
        alfa[k]=mat[k][k];
        beta[k]=-SGN(mat[k][k+1])*sqrt(spat);
        mat[k][k+1]=mat[k+1][k]=beta[k];
        for(j=k+2;j<=n;j++)
        { mat[k][j]=0;
          mat[j][k]=0;
        }
        for(i=k+1;i<=n;i++)
        for(j=k+1;j<=n;j++)
            mat[i][j]=mat[i][j]-2*W[i][k]*q[j]-2*q[i]*W[j][k];
    }
    alfa[n-1]=mat[n-1][n-1];
    alfa[n]=mat[n][n];
    beta[n-1]=mat[n-1][n];
    for(i=1;i<=n;i++)

```



```

        for(j=1;j<=n;j++) printf("%5.3lf ",mat[i][j]);
        printf("\n");
    }
    getch();
    for(i=1;i<=n;i++)
    {
        Coef[i][0]=1;
        for(j=1;j<=n;j++) Coef[i][j]=0;
    }
    Coef[1][1]=-alfa[1];
    Coef[2][2]=alfa[2]*alfa[1]-beta[1]*beta[1];
    for(i=3;i<=n;i++)
        Coef[i][i]=-alfa[i]*Coef[i-1][i-1]-beta[i-1]*beta[i-1]*Coef[i-2][i-2];
    for(i=2;i<=n;i++)
        Coef[i][1]=Coef[i-1][1]-alfa[i];
    for(i=3;i<=n;i++)
        Coef[i][2]=-alfa[i]*Coef[i-1][1]+Coef[i-1][2]-beta[i-1]*beta[i-1];
    for(j=3;j<=n;j++)
        for(i=j+1;i<=n;i++)
            Coef[i][j]=-alfa[i]*Coef[i-1][j-1]+Coef[i-1][j]
                -beta[i-1]*beta[i-1]*Coef[i-2][j-2];
    for(i=0;i<=n;i++)
    {
        Pol[i]=Coef[n][n-i];
        printf("%5.2lf\n",Pol[i]);
    }
    getch();
    Birge_Vieta(n,Pol,0,1000,0.00001,S);
    for(i=1;i<=n;i++)
    {
        Lambda[i]=S[i];
        printf("%5.2lf\n",S[i]);
    }
    getch();
    for(j=1;j<=n;j++)
    {
        LAM=S[j];
        v[1][j]=1;
        for(k=2;k<=n-1;k++)
            v[k+1][j]=((LAM-alfa[k])*v[k][j]-beta[k-1]*v[k-1][j])/beta[k];
    }
    for(j=1;j<=n;j++)
        for(r=1;r<=n-2;r++)
        {
            k=n-1-r;
            suma=0;

```

```

for(i=k+1;i<=n;i++)
    suma=suma+W[i][k]*v[i][j];
for(i=k+1;i<=n;i++)
    v[i][j]=v[i][j]-2*suma*W[i][k];
}
for(i=1;i<=n;i++)
{
    for(j=1;j<=n;j++) printf("%5.3lf ",v[i][j]);
    printf("\n");
}
getche();
}

```

### 10.3.5. METODA LR (LEFT-RIGHT)

Această metodă se bazează pe descompunerea matricei  $A_1$  în două matrice una inferior triunghiulară cu elementele diagonalei principale egale cu unitatea  $L_1$  și una superior triunghiulară  $R_1$ .

$$A_1 = L_1 \cdot R_1; \quad (10.48)$$

Se formează o nouă matrice din matricele triunghiulare obținute:

$$A_2 = R_1 \cdot L_1 \quad (10.49)$$

Matricea  $A_2$  se descompune iarăși în două matrice inferior și superior triunghiulare  $L_2$  și respectiv  $R_2$  :

$$A_2 = L_2 \cdot R_2 \quad (10.50)$$

Se formează o nouă matrice  $A_2$

$$A_2 = R_2 \cdot L_2 \quad (10.51)$$

Procedeul este iterativ și se obține șirul:

$$A_1, A_2, \dots, A_n \quad (10.52)$$

Din ecuația (10.48) rezultă:

$$L_1 = A_1 \cdot R_1^{-1} \quad \text{și} \quad R_1 = L_1^{-1} \cdot A_1 \quad (10.53)$$

Ținând cont de relațiile (10.53) avem:

$$A_2 = R_1 \cdot A_1 \cdot R_1^{-1} \quad \text{sau} \quad A_2 = L_1^{-1} \cdot A_1 \cdot L_1 \quad (10.54)$$

Deci un element oarecare al șirului (10.52) se poate scrie sub forma:

$$A_i = (R_{i-1} \cdot R_{i-2} \cdot \dots \cdot R_2 \cdot R_1) \cdot A_1 \cdot (R_{i-1} \cdot \dots \cdot R_2 \cdot R_1)^{-1} = (L_1 \cdot L_2 \cdot \dots \cdot L_{i-1})^{-1} \cdot A_1 \cdot (L_1 \cdot L_2 \cdot \dots \cdot L_{i-1}) \quad (10.55)$$

ca urmare toate matricele șirului (10.52) au aceleași valori caracteristice.

Produsul matricelor inferior triunghiulare  $L$  este tot o matrice inferior triunghiulară cu elementele diagonalei principale egale cu unitatea:

$$L = \prod_{i=1}^n L_i \quad (10.56)$$

iar produsul matricelor superior triunghiulare  $R$ , este o matrice superior triunghiulară

$$R = \prod_{i=1}^n R_i \quad (10.57)$$

Dacă în șirul matricelor  $A_n$ ,  $n \rightarrow \infty$ , elementele diagonale ale matricei  $R_n$  tind la valorile proprii ale matricei  $A_1$ .

În cadrul metodei se pune problema descompunerii matricei  $A_i$  în matricele  $L_i$  și  $R_i$  și calculul produsului  $R_i \cdot L_i$ .

Calculul elementelor matricelor  $L_i$  și  $R_i$  se realizează după expresiile (10.58) iar elementele produsului  $R_i L_i$  după expresiile (10.59).

$$\begin{aligned} l_{ii} &= 1; \quad i = 1, 2, \dots, n; \\ l_{ij} &= 0; \quad \text{pentru } i < j; \\ l_{i1} &= \frac{a_{i1}}{a_{11}}; \quad i = 2, 3, \dots, n; \quad a_{11} \neq 0; \\ r_{1i} &= a_{1i}; \quad i = 1, 2, \dots, n; \\ r_{ij} &= 0; \quad i > j; \end{aligned} \quad (10.58)$$

$$\begin{aligned} r_{ij} &= a_{ij} - \sum_{k=1}^{i-1} l_{ik} r_{kj}; \quad i \leq j; \\ l_{ij} &= \frac{1}{r_{ii}} (a_{ij} - \sum_{k=1}^{i-1} l_{ik} r_{kj}); \quad r_{ii} \neq 0; \quad i > j; \\ a_{ij} &= r_{ij} + \sum_{k=j+1}^n r_{ik} \cdot l_{kj}; \quad i \leq j; \quad j < n; \\ a_{ij} &= r_{ij}; \quad i \leq j; \quad j = n; \\ a_{ij} &= \sum_{k=i}^n r_{ik} \cdot l_{kj}; \quad i > j; \end{aligned} \quad (10.59)$$

Valorile proprii ale matricei  $A_1$  sunt elementele diagonale ale matricei  $R_{n-1}$  dacă se îndeplinește condiția

$$|a_{ii}^{(n)} - a_{ii}^{(n-1)}| < \varepsilon; \quad i = 1, 2, \dots, n; \quad (10.60)$$

unde  $\varepsilon$  este eroarea de calcul.

**10.3.5.1. Algoritmul 10.5. Metoda LR**

```

{Variabile
A:matrice;
L:matrice inferior triunghiulară;
R:matrice superior triunghiulară;
i,j,k:contori, întregi;
λ:vectorul valorilor proprii;
ε:eroarea de calcul, real;
sum:sumă parțială, real;

{pentru i=1, ... , n
  pentru j=1, ... , n
     $b_{ij} = a_{ij}$ ;
  repetă
     $a_{ij} = b_{ij}$ ;
  pentru i=1, ... , n
    pentru j=1, ... , n
      {dacă  $i > j$  atunci  $r_{ij} = 0$ ;
        dacă  $i < j$  atunci  $l_{ij} = 0$ ;
      }
    pentru i=1, ... , n
      pentru j=i, ... , n
        {
          sum=0;
          pentru k=1, ... , i-1
            calculează  $sum = sum + l_{ik} \cdot r_{kj}$ ;
            calculează  $r_{ij} = a_{ij} - sum$ ;
          }

        dacă  $r_{ii} = 0$  atunci
          stop problema este nerezolvabilă;

        pentru j=i+1, ... , n
          {
            sum=0;
            pentru k=1, ... , i-1
              calculează  $sum = sum + l_{jk} + r_{ki}$ ;
              calculează  $l_{ji} = \frac{a_{ji} - sum}{r_{ii}}$  ;
            }

           $l_{ii} = 1$ ;

```

```

    }
    pentru i=1, ... , n
        pentru j=1, ... , n
            dacă ((i<j) și (j<n)) atunci
                {
                    sum=0;
                    pentru k=j+1, ... , n
                        calculează  $sum = sum + r_{ik} \cdot l_{kj}$ ;
                        calculează  $b_{ij} = r_{ij} + sum$ ;
                    }
            dacă ((i<j) și (j=1)) atunci  $b_{ij} = r_{ij}$ ;
            dacă (i>j) atunci
                {  $b_{ij} = 0$ ;
                  pentru k=1, ... , n
                      calculează  $b_{ij} = a_{ij} + r_{ik} \cdot l_{kj}$ ;
                  }
        până când  $|b_{ii} - a_{ii}| < e$ ;
        scrie valorile proprii sunt  $r_{ii}$  pentru i=1, ... , n;
    }
}

```

### 10.3.5.2. Implementarea algoritmului 10.5. Metoda LR

```

int LR(int n,
        double A[][NrMax],
        double L[][NrMax],
        double R[][NrMax])
{
    int i,j,k;
    double sum;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            {
                if(i>j) R[i][j]=0;
                if(i<j) L[i][j]=0;
            }
    for(i=1;i<=n;i++)
        {
            for(j=i;j<=n;j++)
                {
                    {
                        sum=0;

```

```

        for(k=1;k<=i-1;k++)sum+=L[i][k]*R[k][j];
        R[i][j]=A[i][j]-sum;
    }
}
if (R[i][i]==0) return 0;
for(j=i+1;j<=n;j++)
{
    sum=0;
    for(k=1;k<=i-1;k++)sum+=L[j][k]*R[k][i];
    L[j][i]=(A[j][i]-sum)/R[i][i];
}

L[i][i]=1;
}
return 1;
}
void ProdRL(int n,
            double R[][NrMax],
            double L[][NrMax],
            double A[][NrMax])
{
    int i,j,k;
    double sum;
    for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
    {
        if ( ( i<=j ) && ( j<n ) )
        {
            sum=0;
            for(k=j+1;k<=n;k++) sum+=R[i][k]*L[k][j];
            A[i][j]=R[i][j]+sum;
        }
        if ( ( i<=j ) && ( j==n ) ) A[i][j]=R[i][j];
        if ( i>j )
        {
            A[i][j]=0;
            for(k=i;k<=n;k++) A[i][j]+=R[i][k]*L[k][j];
        }
    }
}
/* Funcția care implementează metoda descompunerii LR
   pentru aflarea valorilor proprii
*/
int VPLR(int n,
        double A[][NrMax],
        double VP[])
{

```

```

double static
r[NrMax][NrMax],l[NrMax][NrMax],A_1[NrMax][NrMax];
int i,j,sem;
double eps=0.00000001;
do
{
sem=1;
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
A_1[i][j]=A[i][j];
if( LR(n,A,l,r)==0) return 0;
ProdRL(n,r,l,A);
for(i=1;i<=n;i++)if( fabs(A[i][i]-A_1[i][i])>eps)sem=0;
}
while(sem==0);
for(i=1;i<=n;i++)VP[i]=A[i][i];
return 1;
}

```

## 10.4. APLICAȚII

1. Se consideră matricea:

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

Să se determine valorile și vectorii proprii ai matricei date.

Prin metoda puterii se obține cea mai mare valoare caracteristică:

$$\lambda_{\max} = 7.225565;$$

Prin metoda lui Krîlov se obține următorul polinom caracteristic:

$$P(\lambda) = \lambda^4 - 5\lambda^3 - 28\lambda^2 + 50\lambda + 261$$

Rădăcinile polinomului sunt determinate cu metoda lui Bairstow și se obțin următoarele valori caracteristice:

$$\lambda_1 = 7.225565; \quad \lambda_2 = 3.680183; \quad \lambda_{3,4} = 2.952874 \pm i1.046774;$$

2. Se dă matricea simetrică:

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

Se cer valorile și vectorii proprii ai matricei  $A$ .

Prin metoda Householder se obțin valorile proprii:

$$l_1 = -6.429979; \quad l_2 = 3.741197; \quad l_3 = 6.029197; \quad l_4 = 9.659586;$$

și vectorii proprii:

$$x_1 = (1.000, 2.011, 2.573, 0.924); \quad x_2 = (1.000, 1.016, -1.719, -1.797);$$

$$x_3 = (1.000, 1.626, 0.914, -0.128); \quad x_4 = (1.000, 3.720, 9.950, 5.602);$$

Prin metoda puterii se obține următoarea valoare caracteristică maximă

$$\lambda_{\max} = 9.659586;$$

Prin metoda lui Krîlov se obține următorul polinom caracteristic:

$$P(\lambda) = \lambda^4 - 13\lambda^3 - 8\lambda^2 + 534\lambda - 1401;$$

Valorile caracteristice sunt:

$$\lambda_1 = -6.429979; \quad \lambda_2 = 3.741197; \quad \lambda_3 = 6.029197; \quad \lambda_4 = 9.659586;$$

iar vectorii proprii sunt identici cu cei de la metoda Householder.

Prin metoda LR și TR s-au obținut aceleași valori proprii ca și în cazul metodei lui Householder.



# 11

## FUNCȚII SPECIALE \*

Acest capitol cuprinde câteva funcții utilizate mai mult în aplicații, și ale căror valori este necesar să le cunoaștem în diferite puncte ale domeniului de definiție. Ca funcții speciale considerăm funcțiile transcendente (funcții definite prin integrale improprii) sau funcții care reprezintă soluții pentru diferite ecuații diferențiale de ordinul doi. Deoarece în calculul acestor funcții se utilizează funcții mai simple, se prezintă și aceste funcții.

### 11.1. FUNCȚIA GAMMA

Funcția gamma este definită de integrala improprie:

$$\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt \quad (11.1)$$

Pentru  $x \in \mathbf{N}$  funcția gamma se calculează cu ajutorul formulei:

$$\Gamma(n+1) = n! \quad (11.2)$$

Ca urmare a acestei relații, funcția factorial se poate calcula și cu ajutorul funcției gamma. Argumentul funcției gamma poate fi și complex, deci  $x = z \in \mathbf{C}$  (mulțimea numerelor complexe). Funcția gamma satisface următoarea relație de recurență:

$$\Gamma(z+1) = z \cdot \Gamma(z) \quad (11.3)$$

Dacă funcția gamma este cunoscută pentru  $\operatorname{Re}(z) > 1$ , ea poate fi calculată și pentru  $\operatorname{Re}(z) < 1$  cu ajutorul expresiei :

$$\Gamma(1-z) = \frac{\pi}{\Gamma(z) \cdot \sin \pi \cdot z} = \frac{\pi \cdot z}{\Gamma(1+z) \cdot \sin \pi \cdot z} \quad (11.4)$$

Calculul numeric al funcției gamma se poate face cu o formulă aproximativă, funcție de constantele întregi  $r, k$  și de un șir de constante  $c_1, c_2, \dots, c_k$ , cu condiția  $z > 0$ .

$$\Gamma(z+1) = \left(z+r+\frac{1}{2}\right)^{\left(z+\frac{1}{2}\right)} e^{-\left(z+r+\frac{1}{2}\right)} \sqrt{2\pi} \left(c_0 + \frac{c_1}{z+1} + \frac{c_2}{z+2} + \dots + \frac{c_k}{z+k} + \varepsilon\right) \quad (11.5)$$

Pentru  $r = 5$  și  $k = 6$  se determină setul de coeficienți astfel ca eroarea  $|\varepsilon| < 2 \cdot 10^{-10}$ .

---

\*Bibliografie [20], [22], [24]

Pentru calculul valorii gamma se realizează calculul  $\ln(\Gamma(x))$  pentru  $x \in \mathbb{R}^+$ . Se ține seama de formula (11.3) din care se calculează  $\Gamma(x) = \frac{1}{x} \Gamma(x+1)$  și de (11.5).

### 11.1.1. ALGORITMUL 11.1. FUNCȚIA GAMMA

```
{Variabile
  x: punctul în care se calculează funcția, real ;
  coef: vectorul coeficienților ;
  j: contor, întreg;
  t,s: variabile pentru sume, real;
  z,y: variabile pentru punctul de calcul, reale;
  gamma: valoarea funcției gamma în x, real;
  {coef[0]= 76.18009172947146;
  coef[1]=-86.50532032941677;
  coef[2]=24.01409824083091;
  coef[3]=-1.231739572450155;
  coef[4]=0.1208650973866179e-2;
  coef[5]=-0.5395239384953e-5
  y:=x;
  z:=x;
  t=x+5.5;
  t:=(x+0.2)*log(t)-t;
  s:=1.000000000190015;
    pentru j=1 până la 5
      {
        calculează s:=s+coef[j]/(y+j+1);
      }
    calculează gamma=exp(-t+log(2.5066282746310005*s/x));
  }
}
```

### 11.2.2. IMPLEMENTAREA ALGORITMULUI 11.1

```
#include <math.h>
/* Funcția întoarce valoarea funcției gamma în punctul xx>0 */
float gammln( float xx)
{
  double x,y,tmp,ser;
  static double coef[6]={ 76.18009172947146,
                          -86.50532032941677,
                          24.01409824083091,
```

```

-1.231739572450155,
0.1208650973866179e-2,
-0.5395239384953e-5};

int j;
float valoare;
y=x=xx;
tmp=x+5.5;
tmp=(x+0.5)*log(tmp);
ser=1.000000000190015;
for(j=0;j<=5;j++)ser+=coef[j]/++y;
valoare= - tmp+log(2.5066282746310005*ser/x);
return valoare;
}
int main(void)
{
float punct;
clrscr();
printf("Punctul de calcul : ");
scanf("%f",&punct);
printf(" Valoare funcției Gamma în punctul %.5f este
%.5f",punct,exp(gammln(punct)));
getche();
return 0;
}

```

## 11.2. FUNCȚIA FACTORIAL

Această funcție este definită pe mulțimea numerelor naturale **N**. Pentru  $n=0$  ia valoarea 1. Funcția factorial reprezintă o funcție numerică calculabilă cu expresia:

$$n! = 1 \cdot 2 \cdot 3 \cdots n \quad (11.6)$$

Calculul valorii acestei funcții într-un punct al domeniului de definiție se poate realiza cu ajutorul funcției gamma, expresia (11.2).

### 11.2.1. ALGORITMUL 11.2. FUNCȚIA FACTORIAL

*{Variabile*

*x: punctul în care se calculează funcția gamma, real ;*

*coef:vectorul coeficienților ;*

*j:contor, întreg;*

*t,s:variabile pentru sume, real;*

*z,y:variabile pentru punctul de calcul, real;*

*n:punctul în care se calculează funcția factorial, întreg;*

*fact:valoarea factorialului, întreg;*

```

{
  gamma(x): funcția gamma
}
{f[1]=1.0;
 f[2]=2.0;
 f[3]=6.0;
 f[4]=24.0;
  dacă n<0 atunci scrie eroare Stop;
  dacă (n>32) atunci f[n] = exp(gammln(n+1.0);
  atâta timp cât (ntop<n )
  {
    j=ntop+1;
    f[ntop]=f[j]*ntop;
  }
  fact=a[n];
}
}

```

#### 11.2.2. IMPLEMENTAREA ALGORITMULUI 11.2

```

{Variabile
#include <math.h>
/* Funcția întoarce valoarea funcției gamma în punctul xx>0 */
float gammln( float xx)
{
  double x,y,tmp,ser;
  static double coef[6]={ 76.18009172947146,
                          -86.50532032941677,
                          24.01409824083091,
                          -1.231739572450155,
                          0.1208650973866179e-2,
                          -0.5395239384953e-5};

  int j;
  float valoare;
  y=x=xx;
  tmp=x+5.5;
  tmp-=(x+0.5)*log(tmp);
  ser=1.000000000190015;
  for(j=0;j<=5;j++)ser+=coef[j]/++y;
  valoare= - tmp+log(2.5066282746310005*ser/x);
  return valoare;
}
/* Întoarce factorialul unui întreg pozitiv utilizând funcția Gamma */
float factrl(int n)
{

```

```

static int ntop=4;
static float a[33]={1.0,1.0,2.0,6.0,24.0};
int j;
if(n<0){
    printf("Eroare. Argument negativ");
    getche();
    return -1;
}
if (n>32) return exp(gammln(n+1.0));
while(ntop<n)
{
    j=ntop++;
    a[ntop]=a[j]*ntop;
}
return a[n];
}
int main(void)
{
    int punct;
    clrscr();
    printf("Punctul de calcul : ");
    scanf("%d",&punct);
    printf(" Valoarea funcției factorial în punctul %d este\n",punct,factrl(punct));
    getche();
    return 0;
}

```

### 11.3. COEFICIENȚII BINOMIALI

Coeficienții binomiali sunt definiți astfel:

$$C(k,n) = \frac{n!}{k!(n-k)!} ; \quad 0 \leq k \leq n ; \quad (11.7)$$

Valoarea coeficienților binomiali se calculează cu ajutorul funcției factorial, ei fiind definiți tot pe mulțimea numerelor naturale  $\mathbf{N}$ . Pentru calculul valorilor coeficienților binomiali se logaritmează formula coeficienților binomiali și se utilizează funcțiile:

$$\text{fact}(n) = \exp(\text{gammaln}(n+1,0)) \text{ și } \text{factln}(n) = \text{gammaln}(n+1.0) \quad (11.8)$$

De aceste expresii se ține seama în realizarea algoritmului metodei de calcul.

#### 11.3.1. ALGORITMUL 11.3. COEFICIENȚII BINOMIALI

{Variabile \_\_\_\_\_  
*x: punctul în care se calculează funcția gamma ;*

```

coef:vectorul coeficienților , real;
j:contor, întreg;
t,s:variabile pentru sume, real;
z,y:variabile pentru punctul de calcul, real;
n:punctul în care se calculează funcția factorial, întreg;
fact:valoarea factorialului, întreg;
{
gamma(x): funcția gama;
}
calculează factln(n);
calculează floor(0.5+exp(factln(n)-factln(k)-factln(n-k)));
scrie coeficienții binomiali;
}
}

```

### 11.3.2. Implementarea algoritmului 11.3

```

#include <math.h>
/* Funcția întoarce valoarea funcției gamma în punctul xx>0 */
float gammln( float xx)
{
double x,y ,tmp,ser;
static double coef[6]={ 76.18009172947146,
                        -86.50532032941677,
                        24.01409824083091,
                        -1.231739572450155,
                        0.1208650973866179e-2,
                        -0.5395239384953e-5};

int j;
float valoare;
y=x=xx;
tmp=x+5.5;
tmp=(x+0.5)*log(tmp);
ser=1.000000000190015;
for(j=0;j<=5;j++)ser+=coef[j]/++y;
valoare= - tmp+log(2.5066282746310005*ser/x);
return valoare;
}
float factln(int n)
{
static float a[101];
if( n<0 ) {
printf("Eroare. Argument negativ");
return -1;
}
}

```

```

        getche();
    }
    if(n<=1) return 0;
    else return gammaln(n+1.0);
}
/* Funcția întoarce coeficienții binomial Combinări de n luate câte k */
float coef_bin(int n, int k)
{
    return floor(0.5+exp(factln(n)-factln(k)-factln(n-k)));
}
int main(void)
{
    int nn, kk;
    clrscr();
    printf("n : ");scanf("%d",&nn);
    printf("k : ");scanf("%d",&kk);
    printf(" Valoare funcției coef_bin de %d luate câte %d este\n",nn,kk,coef_bin(nn,kk));
    getche();
    return 0;
}

```

## 11.4. FUNCȚIA BETA

Funcția beta este definită de următoarea integrală:

$$B(z, w) = B(w, z) = \int_0^1 t^{z-1} (1-t)^{w-1} dt \quad (11.9)$$

și se calculează cu ajutorul funcției gamma după următoarea formulă:

$$B(z, w) = \frac{\Gamma(z) \cdot \Gamma(w)}{\Gamma(z + w)} \quad (11.10)$$

### 11.4.1. Algoritmul 11.4. Funcția beta

```

{ Variabile
z:argumentul funcției, real;
w:argumentul funcției, real;
sol:valoarea funcției beta, real;
{calculează gamma(z);
calculează gamma(w);
    calculează gamma(z+w);
calculează sol=gamma(z)*gamma(w)/gamma(z+w);
}
scrie valoarea funcției sol;

```

---

 }

### 11.4.2. Implementarea algoritmului 11.4

```

#include <math.h>
/* Funcția întoarce valoarea funcției gamma în punctul xx>0 */
float gammln( float xx)
{
    double x,y,tmp,ser;
    static double coef[6]={ 76.18009172947146,
                           -86.50532032941677,
                           24.01409824083091,
                           -1.231739572450155,
                           0.1208650973866179e-2,
                           -0.5395239384953e-5};

    int j;
    float valoare;
    y=x=xx;
    tmp=x+5.5;
    tmp-=(x+0.5)*log(tmp);
    ser=1.000000000190015;
    for(j=0;j<=5;j++)ser+=coef[j]/++y;
    valoare= - tmp+log(2.5066282746310005*ser/x);
    return valoare;
}
/* Funcția întoarce valoarea funcției beta(z,w) */
float beta( float z, float w)
{
    return exp(gammln(z)+gammln(w)-gammln(z+w));
}
int main(void)
{
    float zz,ww;
    clrscr();
    printf("z : ");
    scanf("%f",&zz);
    printf("w : ");
    scanf("%f",&ww);
    printf(" Valoarea funcției Beta( %.5f,%.5f) este\n",
           %.5f,zz,ww,beta(zz,ww));
    getche();
    return 0;
}

```



### 11.5. FUNCȚIILE BESSEL

Prin definiție, se numește ecuația lui Bessel ecuația diferențială:

$$x^2 y'' + xy' + (x^2 - \nu^2)y = 0 \quad (11.11)$$

unde  $\nu$  este un parametru real sau complex, iar soluțiile ecuației se numesc funcții Bessel sau funcții cilindrice. Soluția ecuației (1) se caută sub forma:

$$y(x) = x^r \sum_{k=0}^{\infty} C_k x^k, \quad x \in \mathbb{C} \quad (11.12)$$

în care  $r$  și  $C_k$  se vor calcula. Prin înlocuirea expresiei (11.12) în ecuația (11.11) se ajunge la soluția :

$$y(x) = \sum_{k=0}^{\infty} \frac{(-1)^k}{k! \Gamma(\nu + k + 1)} \left(\frac{x}{2}\right)^{\nu+2k} \quad (11.13)$$

Notăția  $y$  a funcției este înlocuită cu notația  $J$  consacrată pentru funcțiile lui Bessel de speța întâi. Astfel, funcția

$$J_\nu(x) = \left(\frac{x}{2}\right)^\nu \sum_{k=0}^{\infty} \frac{\left(-\frac{1}{4}x^2\right)^k}{k! \Gamma(\nu + k + 1)} \quad (11.14)$$

pentru  $\nu \neq 0$ ,  $\arg(\nu) \in [0, \pi]$  are următoarele proprietăți:

- 1) pentru  $\nu = n \in \mathbb{N}$ ,  $J_\nu$  este o funcție întreagă;
- 2) pentru  $\nu \notin \mathbb{N}$ ,  $J_\nu$  este olomorvă pe  $\mathbb{D} \setminus T$  unde  $T$  este o semidreaptă cu originea în  $O$ .

Dacă  $\nu$  nu este întreg se poate obține o nouă soluție a ecuației Bessel ca o combinație de funcții Bessel  $J_\nu$ . Astfel, funcția

$$Y_\nu(x) = \frac{J_\nu(x) \cos n\pi - J_{-\nu}(x)}{\sin n\pi} \quad \dots \quad (11.15)$$

reprezintă soluții ale ecuației lui Bessel (11.11) și se numesc funcții Bessel de speța a doua. Și aceste funcții la limită pot fi date pentru  $\nu$  întreg. Pentru  $0 < x < \nu$  se fac aproximările:

$$\begin{aligned} J_\nu(x) &\approx \frac{1}{\Gamma(\nu+1)} \left(\frac{x}{2}\right)^\nu; \\ Y_0(x) &\approx \frac{2}{\pi} \ln x; \\ Y_\nu(x) &\approx \frac{\Gamma(x)}{\pi} \left(\frac{x}{2}\right)^{-\nu}; \end{aligned} \quad (11.16)$$

Pentru  $x \gg \nu$  se fac aproximările:

$$\begin{aligned} J_\nu &\approx \sqrt{\frac{2}{\pi x}} \cos\left(x - \frac{1}{2}\nu\pi - \frac{1}{4}\pi\right) \\ Y_\nu &\approx \sqrt{\frac{2}{\pi x}} \sin\left(x - \frac{1}{2}\nu\pi - \frac{1}{4}\pi\right) \end{aligned} \quad (11.17)$$

Pentru  $x \approx \nu$  se fac aproximațiile:

$$\begin{aligned} J_\nu &\approx \frac{\sqrt[3]{2}}{\sqrt[3]{9}\Gamma\left(\frac{2}{3}\right)} \cdot \frac{1}{\sqrt[3]{\nu}} \approx \frac{0.4473}{\sqrt[3]{\nu}} \\ Y_\nu &\approx -\frac{\sqrt[3]{2}}{\sqrt[3]{9}\Gamma\left(\frac{2}{3}\right)} \cdot \frac{1}{\sqrt[3]{\nu}} \approx \frac{0.7748}{\sqrt[3]{\nu}} \end{aligned} \quad (11.18)$$

Relațiile de recurență pentru calculul funcțiilor Bessel sunt:

$$\begin{aligned} J_{n+1}(x) &= \frac{2n}{x} J_n(x) - J_{n-1}(x) \\ Y_{n+1}(x) &= \frac{2n}{x} Y_n(x) - Y_{n-1}(x) \end{aligned} \quad (11.19)$$

Pentru  $0 < x < 8$  se aproximează  $J_0(x)$  și  $J_1(x)$  prin funcții raționale în  $x$ .

Aceste aproximații pentru  $Y_0(x)$  și  $Y_1(x)$  sunt:

$$Y_0(x) \cong \frac{2}{\pi} J_0(x) \ln x \quad \text{și} \quad Y_1(x) \cong \frac{2}{\pi} \left( J_1(x) \ln x - \frac{1}{x} \right)$$

Pentru  $8 < x < \infty$  se utilizează aproximațiile ( $n=0,1$ ):

$$\begin{aligned} J_n(x) &= \sqrt{\frac{2}{\pi x}} \left( P_n\left(\frac{8}{x}\right) \cos X_n - Q_n\left(\frac{8}{x}\right) \sin X_n \right) \\ Y_n(x) &= \sqrt{\frac{2}{\pi x}} \left( P_n\left(\frac{8}{x}\right) \sin X_n + Q_n\left(\frac{8}{x}\right) \cos X_n \right) \end{aligned} \quad (11.20)$$

unde

$$X_n = x - \frac{2n+1}{4} \cdot \pi; \quad 0 < \frac{8}{x} < 1;$$

Calculul funcțiilor Bessel întregi începe cu calculul funcțiilor  $J_0, J_1, Y_0, Y_1$ , după care se aplică formulele de recurență.

Coeficienții polinoamelor  $P_n\left(\frac{1}{x}\right), Q_n\left(\frac{1}{x}\right)$ , [24], sunt dați în cadrul programelor.

În continuare, se dau integral programele de calcul pentru funcțiile Bessel de speța întâi și de ordinul zero, de speța a doua și de ordinul zero, de speța întâi și de ordinul întâi, și de speța a doua și ordinul întâi notate respectiv  $J_0, Y_0, J_1, Y_1$ .

### 11.5.1. PROGRAMUL PENTRU FUNCȚIA BESSEL $J_0$

```

#include <math.h>
/* Funcția întoarce valoarea lui Jo(x) pentru x real */
float bessj0(float x)
{
    float ax,z;
    double xx,y, ans,ans1,ans2;
    if ((ax=fabs(x)) <8.0
        {
            y=x*x;
            ans1=57568490574.0+y*(-13362590354.0+y*(651619640.7
                +y*(-11214424.18+y*(77392.33017+y*(-184.9052456)))));
            ans2=57568490411.0+y*(-1029532985.0+y*(9494680.718
                +y*(59272.64853+y*(267.8532712+y*(1.0)))));
            ans=ans1/ans2;
        }
        else
        {
            z=8.0/ax;
            y=z*z;
            xx=ax-0.785398164;
            ans1=1.0+y*(-0.1098628627e-2+y*(0.2734510407e-4+
                y*(-0.2073370639e-5+y*0.2093887211e-6)));
            ans2=-0.1562499995e-1+y*(0.1430488765e-3+
                y*(0.6911147651e-5+y*0.7621095161e-6-
                y*0.934935152e-7)));
            ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-z*sin(xx)*ans2);
        }
        return ans;
    }
int main (void)
{
    float punct;
    clrscr();
    printf("Dați punctul : ");
    scanf("%f",&punct);
    printf("Rezultat : %.5f",bessj0(punct));
    getch();
    return 0;
}

```

### 11.5.2. PROGRAMUL PENTRU FUNCȚIA BESSEL $Y_0$

```

/* Întoarce  $Y_0(x)$  pentru  $x>0$  */

```

```

float bessy0( float x)
{
    float z;
    double xx,y,ans,ans1,ans2;
    if (x<8.0)
    {
        y=x*x;
        ans1=-2957821389.0+y*(7062834065.0+y*(-512359803.6+
            y*(10879881.29+y*(-86327.92757+y*228.4622733))));
        ans2=40076544269.0+y*(745249964.8+y*(7189466.438+
            y*(47447.26470+y*(226.1030244+y*1.0))));
        ans=(ans1/ans2)+0.636619772*bessj0(x)*log(x);
    }
    else
    {
        z=8.0/x;
        y=z*z;
        xx=x-0.785398164;
        ans1=1.0+y*(-0.1098628627e-2+y*(0.2734510407e-4+
            y*(-0.2073370639e-5+y*0.2093887211e-6)));
        ans2=-0.1562499995e-1+y*(0.1430448765e-3+
            y*(-0.6911147651e-5+ y*(0.7621095161e-6+
            y*(-0.934945152e-7))));
        ans=sqrt(0.636619772/x)*(sin(xx)*ans1+z*cos(xx)*ans2);
        return ans;
    }
}

int main (void)
{
    float punct;
    clrscr();
    printf("Dați punctul : ");
    scanf("%f",&punct);
    printf("Rezultat : %.5f",bessy0(punct));
    getch();
    return 0;
}

```

### 11.5.3. PROGRAMUL PENTRU FUNCȚIA BESSEL $J_1$

```

#include <math.h>
/* Funcția întoarce valoarea lui  $J_1(x)$  pentru  $x$  real */
float bessj1(float x)
{
    float ax,z;

```

```

double xx,y, ans,ans1,ans2;
if ((ax=fabs(x)) <8.0 )
{
    y=x*x;
    ans1=x*(72362614232.0+y*(-7895059235.0+y*(242396853.1
+y*(-2972611.439+y*(15704.48260+y*(- 30.16036606))))));
    ans2=144725228442.0+y*(-2300535178.0+y*(18583304.74
+y*(99447.43394+y*(376.9991397+y*(1.0)))));
    ans=ans1/ans2;
}
else
{
    z=8.0/ax;
    y=z*z;
    xx=ax-2.356194491;
    ans1=1.0+y*(-0.183105e-2+y*(0.3516396496e-4+
y*(-0.2457520174e-5+y*(-0.240337019e-6)))));
    ans2=0.04687499995+y*(-0.2002690873e-3+y*(0.8449199096e-5+
y*(-0.88228987e-6+y*0.105787412e-6)));
    ans=sqrt(0.636619772/ax)*(cos(xx)*ans1-z*sin(xx)*ans2);
    if (x<0.0) ans=-ans;
}
return ans;
}
int main (void)
{
    float punct;
    clrscr();
    printf("Dați punctul : ");
    scanf("%f",&punct);
    printf("Rezultat : %.5f",bessj1(punct));
    getch();
    return 0;
}

```

#### 11.5.4. PROGRAMUL PENTRU FUNCȚIA BESSEL Y<sub>1</sub>

```

#include <math.h>
/* Funcția întoarce valoarea lui J1(x) pentru x real */
float bessj1(float x)
{
    float ax,z;
    double xx,y, ansj,ans1,ans2;
    if ((ax=fabs(x)) <8.0 )
    {y=x*x;

```

```

    ans1=x*(72362614232.0+y*(-7895059235.0+y*(242396853.1
        +y*(-2972611.439+y*(15704.48260+y*(-30.16036606))))));
    ans2=144725228442.0+y*(-2300535178.0+y*(18583304.74
        +y*(99447.43394+y*(376.9991397+y*(1.0)))));
    ansj=ans1/ans2;
}
else
{
    z=8.0/ax;
    y=z*z;
    xx=ax-2.356194491;
    ans1=1.0+y*(-0.183105e-2+y*(0.3516396496e-4+
        y*(-0.2457520174e-5+y*(-0.240337019e-6)))));
    ans2=0.04687499995+y*(-0.2002690873e-3+y*(0.8449199096e-5+
        y*(-0.88228987e-6+y*0.105787412e-6)));
    ansj=sqrt(0.636619772/ax)*(cos(xx)*ans1-z*sin(xx)*ans2);
    if (x<0.0) ansj=-ansj;
}
return ansj;
}
/* Funcția întoarce valoarea lui Y1(x) pentru x pozitiv */
float bessy1(float x)
{
    float z;
    double xx,y, ans,ans1,ans2,ansj;
    if (x < 8.0 )
    {
        y=x*x;
        ans1=x*(-0.4900604943e13+y*(0.1275274390e13
            +y*(-0.5153438139e11+y*(0.7349264551e9+y*(-0.4237922726e7
            +y*0.8511937935e4)))));
        ans2=0.2499580570e14+y*(0.4244419664e12+y*(0.3733650367e10
            +y*(0.2245904002e8+y*(0.1020426050e6+y*(0.3549632885e3
            +y)))));
        ans=(ans1/ans2)+0.636619772*(ansj*log(x)-1.0/x);
    }
    else
    {
        z=8.0/x;
        y=z*z;
        xx=x-2.356194491;
        ans1=1.0+y*(0.183105e-2+y*(0.3516396496e-4
            +y*(-0.2457520174e-5+y*(-0.240337019e-6)))));
        ans2=0.04687499995+y*(-0.2002690873e-3+y*(0.8449199096e-5
            +y*(-0.88228987e-6-y*0.105787412e-6)));
    }
}

```

```

    ans=sqrt(0.636619772/x)*(sin(xx)*ans1+z*cos(xx)*ans2);
}
return ans;
}
int main (void)
float punct;
clrscr();
printf("Dați punctul : ");
scanf("%f",&punct);
printf("Rezultat : %.5f",bessyl(punct));
getche();
return 0;
}

```

## 11.6. APLICAȚII

În tabelul (11.1) sunt prezentate valorile funcțiilor  $J_0, Y_0, J_1, Y_1$  pentru diferite valori ale argumentului  $x$ .

Tabelul 11.1

$x$	$J_0$	$J_1$	$Y_0$	$Y_1$
0	1.00000	0.00000	-1.144e+308	-1.245+340
0.1	0.99750	0.04995	-1.53424	-6.38575
0.2	0.99002	0.09963	-1.08111	-3.22188
0.5	0.93847	0.24420	-0.44452	-1.36457
0.9	0.80752	0.41654	0.00563	-0.84590
1.0	0.76520	0.45426	0.08826	-0.78121
1.2	0.67113	0.52163	0.22808	-0.67897
1.5	0.51183	0.59930	0.38245	-0.55633
1.9	0.28182	0.65179	0.49682	-0.40188
2.0	0.22389	0.65487	0.51038	-0.36152
2.4	0.00251	0.62454	0.51041	-0.18943
3.0	-0.26005	0.45084	0.37685	0.08754
3.5	-0.38013	0.20216	0.18902	0.30063
4.0	-0.39715	-0.10903	-0.01694	0.45621
4.5	-0.32054	-0.43292	-0.19471	0.52224
5.0	-0.17760	-0.70237	-0.30852	0.48350
5.5	-0.00684	-0.84045	-0.33948	0.34680
6.0	0.15065	-0.77852	-0.28819	0.14059
6.5	0.26009	-0.48785	-0.17324	-0.09077
7.0	0.30008	-0.01631	-0.02595	-0.29687
7.5	0.26634	0.49928	0.11731	-0.43262

8.0	0.17165	0.23382	0.22352	-0.15806
8.5	0.04194	0.27226	0.27021	-0.02617
9.0	-0.09033	0.24460	0.24994	0.10431

Calculând mai multe valori ale funcțiilor date în intervalul (0,10) s-au obținut graficele funcțiilor Bessel din fig.11.1.

Funcția Bessel

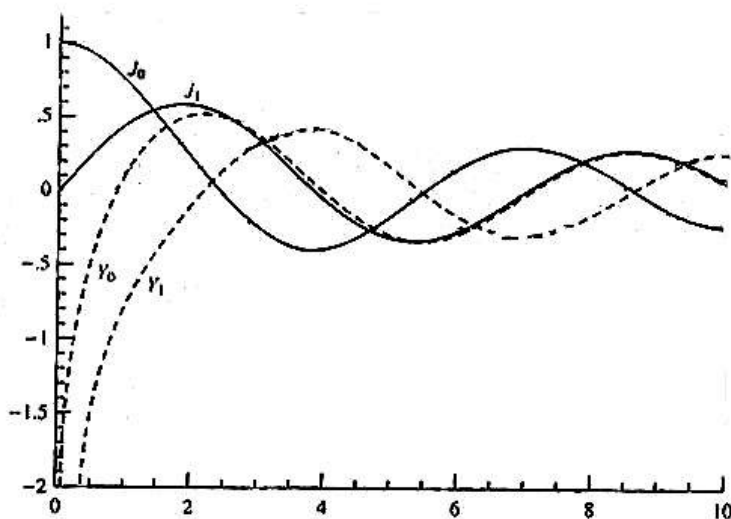


Fig.11.1. Graficele funcțiilor Bessel

Cu ajutorul programelor se poate determina valoarea funcțiilor Bessel în orice punct din domeniul de definiție al funcțiilor.



# 12

## TRANSFORMATĂ FOURIER DISCRETĂ\*

Un semnal discret este definit prin valorile acestuia la momente discrete de timp. În general, timpul  $t$  este discretizat uniform dacă  $t=nT$  cu  $n \in \mathbb{Z}$ . Semnalele în timp discret sunt reprezentate matematic prin secvențe de numere notate astfel:

$$\begin{aligned} \{x[nT]\}, \quad n \in [N_1, N_2] \\ \{x[n]\}, \quad n \in [N_1, N_2] \\ x[n], \quad n \in [N_1, N_2] \end{aligned} \quad (12.1)$$

Toate aceste reprezentări sunt identice dacă în prima relație se ia  $T=1$ .

Se va nota semnalul discret  $x[n]$  iar limitele intervalului în care este definit pot lua valori întregi, depinzând de suportul pe care este definit semnalul.

În studiul semnalelor și sistemelor în timp discret se utilizează câteva secvențe de bază cum ar fi :

1. Secvența impuls unitate, prezentată în figura (12.1), definită prin relația (12.2) și cu o importanță mare ca și funcția Dirac  $\delta(t)$  pentru semnalele analogice.

$$\delta[n] = \begin{cases} 0, & \text{pentru } n \neq 0; \\ 1, & \text{pentru } n = 0; \end{cases} \quad (12.2)$$

2. Secvența treaptă unitate, prezentată în figura (12.2) și definită prin relația (12.3).

$$u[n] = \begin{cases} 1, & \text{pentru } n \geq 0; \\ 0 & \text{pentru } n < 0; \end{cases} \quad (12.3)$$

Legătura între secvența unitate și secvența impuls este dată de relația:

$$u[n] = \sum_{k=0}^n \delta[k] \quad \text{sau} \quad u[n] = \sum_{k=0}^{\infty} \delta[n-k] \quad (12.4)$$

Legătura inversă, între secvența impuls și secvența treaptă, este dată de relația cu diferențe finite (12.5):

$$\delta[n] = u[n] - u[n-1] \quad (12.5)$$

---

\*)Bibliografie: [5],[16],[24]

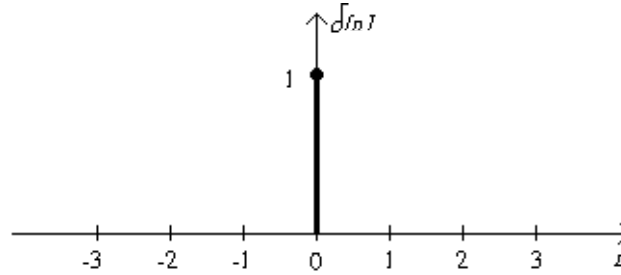


Fig.12.1. Impulsul unitate

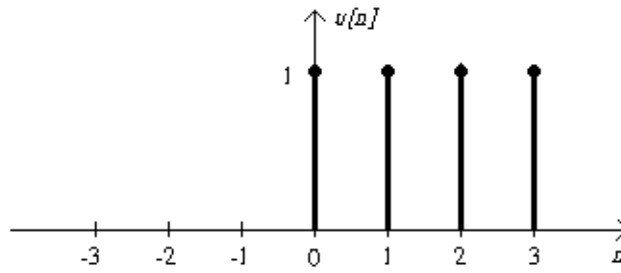


Fig.12.2. Impulsul treaptă unitate

3. Secvența exponențială este definită prin relația următoare:

$$x[n] = \begin{cases} A \cdot \alpha^n, & \text{pentru } \forall n \in \mathbb{N}, \alpha \in (0,1) \\ 0, & \text{pentru } n < 0; \end{cases} \quad (12.6)$$

iar secvența sinusoidală prin expresia :

$$x[n] = A \cdot \cos(\omega_0 n + \varphi), \forall n \in \mathbb{Z} \quad (12.7)$$

Datorită faptului că  $n=t / T$  este adimensional, reprezentând numărul eșantionului, pulsația trebuie luată în radiani / eșantion sau frecvența în Hertz/eșantion, iar calculul sau analiza semnalului se face pentru  $\omega_0 \in [0, 2\pi]$  sau  $\omega_0 \in [-\pi, \pi]$ .

Periodicitatea în timp discret este definită prin relația:

$$x[n] = x[n \pm N] \quad \forall n \in \mathbb{Z} \quad (12.8)$$

unde  $N$  este întreg. Testând condiția de periodicitate, trebuie să avem:

$$x[n] = A e^{j\omega_0 n} = A e^{j\omega_0 (n+N)} \quad (12.9)$$

deci

$$\omega_0 N = 2k\pi \quad (12.10)$$

Din relația (12.10) rezultă că numai pentru anumite valori ale pulsației,  $N$  și  $k$  sunt întregi. Pulsațiile din intervalul  $[0, 2\pi]$  sunt date de setul de valori:

$$\omega_{0k} = \frac{2k\pi}{N}, \quad k = 0, 1, 2, \dots, N; \quad (12.11)$$

Sistemele care transferă o secvență de intrare  $x[n]$  într-o altă secvență de ieșire  $y[n]$  se numesc sisteme în timp discret. Notând  $S[.]$  operatorul sistemului, acesta este descris matematic prin relația:

$$y[n] = S\{x[n]\} \quad (12.12)$$

Pentru sistemele discrete în timp sunt importante următoarele proprietăți:

1. Liniaritatea în timp, ce presupune satisfacerea condiției:

$$S\{c_1x_1[n] + c_2x_2[n]\} = c_1y_1[n] + c_2y_2[n] \quad (12.13)$$

2. Invarianța în timp, care implică conservarea translației semnalului la trecerea prin sistem. Dacă  $x_1[n] = x[n - n_0]$ , atunci răspunsul este:

$$y_1[n] = y[n - n_0] \quad (12.14)$$

3. Cauzalitatea presupune un răspuns neanticipativ la orice secvență de intrare.

Răspunsul la  $n = n_0$  depinde numai de intrările la  $n \leq n_0$ .

4. Stabilitatea definită printr-un răspuns finit la o intrare finită.

5. Sisteme fără memorie sunt sistemele la care răspunsul depinde de intrare pentru același număr  $n$  de eșantioane.

## 12.1. ANALIZA ÎN FRECVENȚĂ A SEMNALELOR ÎN TIMP DISCRET

Prelucrarea numerică a semnalelor își mărește domeniul de aplicabilitate deși evoluția proceselor și în particular a semnalelor este continuă. Evoluția rapidă a componentelor electronice digitale și a performanțelor circuitelor de conversie analog/numerică au contribuit decisiv la trecerea spre prelucrarea numerică a semnalelor. În ceea ce privește eșantionarea semnalului continuu, pentru a nu se produce interferarea lobilor secundari cu cel principal în spectrul semnalului eșantionat, este necesar ca frecvența maximă a spectrului să fie mai mică sau cel mult egală cu frecvența lui Nyquist, care reprezintă jumătate din frecvența de eșantionare. La fel ca și pentru semnalele continue este necesar să se cunoască relația dintre semnalul numeric și spectrul semnalului numeric, relație dată de transformata Fourier discretă.

### 2.1.1. REPREZENTAREA SECVENȚELOR CU TRANSFORMATĂ FOURIER

În acest paragraf frecvența și pulsația sunt normate și nu apar apar relații cu mărimi nenormate. Intervalele de studiu sunt  $-0.5 < f < 0.5$  și  $-\pi < \omega < \pi$ .

Majoritatea secvențelor utilizate în tehnică pot fi reprezentate în domeniul frecvență cu ajutorul transformatei Fourier (12.15):

$$F\{x[n]\} = X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \quad (12.15)$$

unde  $X(e^{j\omega})$  poartă numele de funcție densitate spectrală sau spectrul secvenței  $x[n]$ . Relația (12.15) se referă la transformata Fourier a semnalelor în timp discret (notată și DTFT= *Discrete Time Fourier Transform*).

Reprezentarea secvențelor prin formula (12.15) necesită existența transformatei deci:

$$|X(e^{j\omega})| < \infty \text{ pentru } \forall \omega \in \mathbb{R} \quad (12.16)$$

Această condiție se poate simplifica astfel:

$$|X(e^{j\omega})| = \left| \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \right| \leq \sum_{n=-\infty}^{\infty} |x[n]| \cdot |e^{j\omega n}| \quad (12.17)$$

unde  $|e^{j\omega n}| = 1$ . Ca urmare, condiția devine  $\sum_{n=-\infty}^{\infty} |x[n]| < \infty$  (12.18)

Dacă secvența  $x[n]$  este absolut sumabilă, transformata Fourier există și, de aici rezultă că intrările și răspunsurile sistemelor stabile în timp discret sunt totdeauna reprezentabile în domeniul frecvență cu ajutorul transformatei Fourier. Condiția (12.16) se mai poate exprima și prin următoarea limită:

$$\lim_{M \rightarrow \infty} |X(e^{j\omega}) - X_M(e^{j\omega})| = 0 \quad \text{unde} \quad X_M(e^{j\omega}) = \sum_{n=-M}^M x[n]e^{-j\omega n} \quad (12.19)$$

Dacă  $x[n]$  este absolut sumabil, convergența seriei (12.15) este asigurată pentru orice  $\omega$ . Teoria dezvoltării în serie exponențială [MA] asigură convergența în sensul erorii pătratice minime pentru secvențe de pătrat sumabil (de energie finită), adică satisfac condiția:

$$\sum_{n=-\infty}^{\infty} |x[n]|^2 < \infty \quad (12.20)$$

Mai general, relația (12.19) poate fi scrisă sub forma (12.21):

$$\lim_{M \rightarrow \infty} \int_{-\pi}^{\pi} |x(e^{j\omega}) - X_M(e^{j\omega})|^2 d\omega = 0$$

Transformatele Fourier sunt funcții continue de  $\omega$  și periodice cu perioada  $2\pi$ . Dacă este cunoscută transformarea Fourier  $X(e^{j\omega})$ , se poate calcula  $x[n]$  cu ajutorul transformării inverse:

$$x[n] = F^{-1}\{X(e^{j\omega})\} \text{ sau } x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n} d\omega \quad (12.21)$$

În caz general, transformata Fourier  $X(e^{j\omega})$  este o funcție complexă de  $\omega$ . În analiza semnalului interesează părțile reală și imaginară, respectiv modulul și faza:

$$X(e^{j\omega}) = X_R(e^{j\omega}) + jX_I(e^{j\omega}) \quad (12.22)$$

sau  $X(e^{j\omega}) = |X(e^{j\omega})|e^{j\varphi(\omega)} \quad (12.23)$

Funcția  $X(e^{j\omega})$  reprezintă spectrul în frecvență al secvenței  $x[n]$ ,  $|X(e^{j\omega})|$  este spectrul de amplitudine, iar  $\varphi(\omega) = \arg(X(e^{j\omega}))$  este spectrul de fază. Funcția spectrului de fază nu este univoc determinată. Reprezentările se fac pe o perioadă.

### 12.1.1.1. Algoritmi de calcul al transformatei Fourier rapidă FFT (Fast Fourier Transform )

Pentru calculul transformatei Fourier discrete (DTFT):

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{nk}, \quad k = 0, 1, \dots, N-1, \quad W_N = e^{-j\frac{2\pi}{N}} \quad (12.24)$$

sunt necesare un număr de  $N^2$  înmulțiri complexe și  $N(N-1)$  adunări complexe. Dacă o înmulțire complexă se efectuează în patru înmulțiri și două adunări reale, rezultă un număr de  $4 \cdot N^2$  înmulțiri reale și  $4 \cdot N^2 - 2 \cdot N$  adunări reale.

Având în vedere proprietățile coeficienților  $W_N$ :

$$W_N^{kN} = 1; \quad W_N^{(2k+1) \cdot \frac{N}{2}} = W_N^{\frac{N}{2}} = e^{-j\pi} = -1; \quad W_N^{nk} = W_N^{\frac{n}{2}}; \quad (12.25)$$

există algoritmi care permit efectuarea DTFT cu un număr mai mic de operații. Acești algoritmi pleacă de la descompunerea transformatei de ordinul  $N$  în transformate de ordin mai mic. Componentele pot fi prime între ele sau pot avea divizori comuni. Un caz important este acela în care  $N = R^P$ , unde  $R$  este numită bază. Pentru explicarea unei largi categorii de algoritmi se poate porni de la reprezentarea indicilor sub forma:

$$n = k_1 n_1 + k_2 n_2 \pmod{N}; \quad k = k_3 k_1 + k_4 k_2 \pmod{N} \quad (12.26)$$

#### 12.1.1.1.1. Algoritmul în bază 2 cu decimare în timp

Algoritmul este ales pentru secvențe reale. Secvențele  $x[n]$  reale au proprietățile:

$$X[k] = X^*[N-k], \text{ sau } \operatorname{Re} X[k] = \operatorname{Re} X[N-k], \operatorname{Im} X[k] = \operatorname{Im} X[N-k] \quad (12.27)$$

iar  $X[0]$  [și  $X[N/2]$ ] au valori reale. Ca urmare, este suficient să se calculeze următoarele  $N$  valori:

$$X[0], \operatorname{Re} X[1], \operatorname{Re} X[2], \dots, \operatorname{Re} X[N/2-1], \\ X[N/2], \operatorname{Im} X[1], \operatorname{Im} X[2], \dots, \operatorname{Im} X[N/2-1] \quad \text{---}$$

Algoritmul este:

$$X[k_1 + \frac{N}{2} k_2] = \sum_{n_1=0}^{\frac{N}{2}-1} x[2n_1] \cdot W_N^{n_1 k_1} + W_N^k \cdot W_N^{\frac{N}{2} k_2} \cdot \sum_{n_1=0}^{\frac{N}{2}-1} x[2n_1+1] \cdot W_N^{n_1 k_1}. \quad (12.28)$$

Pentru  $k_2 = 0$  rezultă:

$$X[k_1] = X'[k_1] + W_N^{k_1} \cdot X''[k_1] \quad (12.29)$$

unde  $X'[k_1]$  și  $X''[k_1]$  reprezintă cele două DTFT de ordinul doi. Pentru a pune în evidență simetria circulară pară a acestor transformate:

$$X'[k_1] = X^* \cdot [\frac{N}{2} - k_1]; \quad X''[k_1] = X^{**} \cdot [\frac{N}{2} - k_1] \quad (12.30)$$

și evaluând avem:

$$X[\frac{N}{2} - k_1] = X'[k_1] - W_N^{-k_1} \cdot X''[k_1] \quad (12.31)$$

Părțile reale și imaginare sunt:

$$\begin{aligned}
\operatorname{Re} X[k_1] &= \operatorname{Re} X'[k_1] + \cos \theta \operatorname{Re} X''[k_1] + \sin \theta \operatorname{Re} X'''[k_1] \\
\operatorname{Im} X[k_1] &= \operatorname{Im} X'[k_1] - \sin \theta \operatorname{Re} X''[k_1] + \cos \theta \operatorname{Im} X''[k_1] \\
\operatorname{Re} X\left[\frac{N}{2} - k_1\right] &= \operatorname{Re} X'[k_1] - \cos \theta \operatorname{Re} X''[k_1] - \sin \theta \operatorname{Re} X'''[k_1] \quad (12.32) \\
\operatorname{Im} X\left[\frac{N}{2} - k_1\right] &= -\operatorname{Im} X'[k_1] - \sin \theta \operatorname{Im} X''[k_1] + \cos \theta \operatorname{Re} X''[k_1] \\
\theta &= k_1 \frac{2\pi}{N}
\end{aligned}$$

Se utilizează primele două relații pentru eșantioanele de ordin  $0, \dots, N/4$ , luând  $k_1 = 0, \dots, N/4$ , apoi eșantioanele  $N/4, \dots, N/2$ , luând  $k_1 = 0, \dots, N/4$ . Rezultă fluturile de calcul cu structura din figura 12.3.

Structuri mai simple se obțin pentru  $k_1 = 0, N/4, N/8$ .

Pentru evaluarea complexității calculului aritmetic trebuie avut în vedere că într-un etaj există un fluture de tipul  $k_1 = 0$  caracterizat prin două adunări, un fluture de tipul  $k_1 = N/4$  care nu necesită operații, un fluture de tipul  $k_1 = N/8$  care necesită două înmulțiri și șase adunări și  $N/4 - 2$  fluturi complecși care necesită fiecare patru înmulțiri și șase adunări. În final rezultă:

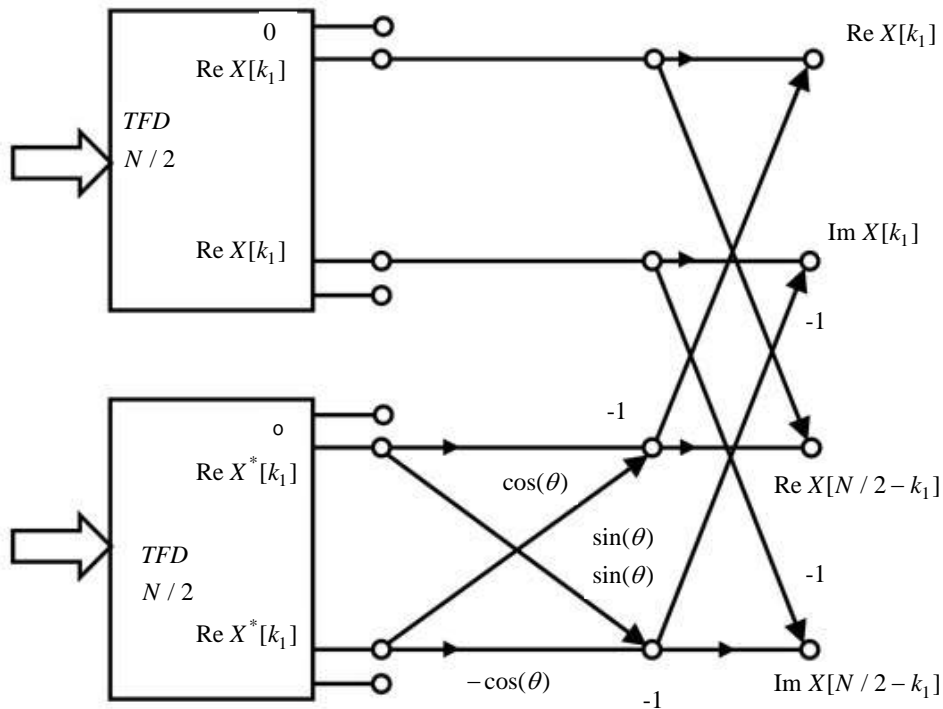


Fig.1.2.3. Structura fluturelui de calcul în algoritmul cu decimare în timp

$$\text{numărul de înmulțiri reale (NMR), } NMR[N] = N \log_2 N - \frac{13}{4} N + 4 \quad (12.33)$$

$$\text{numărul de adunări reale (NAR), } NAR[N] = \frac{3}{2} N \log_2 N - \frac{5}{2} N + 4 \quad (12.34)$$

unde  $N=2^n$  reprezintă numărul total de eșantioane. Numărul de operații se micșorează, prin acest algoritm rezultând o reducere a memoriei necesare ca urmare a utilizării unor numere reale în loc de numere complexe.

#### 12.1.1.1.2. Programul pentru FFT cu decimare în timp

```
#include <math.h>
#include <graphics.h>
#define PI 3.141592653
/* Transformata Fourier rapidă cu decimare în timp
   pentru secvențe de tipul 2^m
   m ordinul secvenței
   x vector cu dimensiunea 2^m care conține la intrare
   partea reală a semnalului în timp, iar la ieșire
   partea reală a semnalului în frecvență
   y vector cu dimensiunea 2^m care conține la intrare
   partea imaginară a semnalului în timp, iar la ieșire
   partea imaginară a semnalului în frecvență
*/
void FFT_DT(int m, double x[], double y[])
{
    int i, j, k, l, n1, n2, n;
    double a, c, s, xt, yt, w;
    n = pow(2, m);
    j = 1;
    for(i = 1; i <= n - 1; i++)
    {
        if(i < j)
        {
            xt = x[j];
            x[j] = x[i];
            x[i] = xt;
            yt = y[j];
            y[j] = y[i];
            y[i] = yt;
        }
        k = n / 2;
        while (k < j)
        {
            j -= k;
```

```

                                k=k/2;
                                }
                        j+=k;
                }
        n1=1;
        for(k=1;k<=m;k++)
        {
                n2=n1;
                n1=n2*2;
                w=PI/n2;
                a=0;
                for(j=1;j<=n2;j++)
                {
                        c=cos(a);
                        s=sin(a);
                        a=j*w;
                        for(i=j;i<=n;i+=n1)
                        {
                                l=i+n2;
                                xt=c*x[l]+s*y[l];
                                yt=c*y[l]-s*x[l];
                                x[l]=x[i]-xt;
                                x[i]=x[i]+xt;
                                y[l]=y[i]-yt;
                                y[i]=y[i]+yt;
                        }
                }
        }
}

void main(void)
{
        int i;
        int gdriver = DETECT, gmode, errorcode;
        double re[128],im[128], modul[128];
        for(i=0;i<=127;i++)
        {
                re[i]=0;
                im[i]=0;
        }

        /* Sunt date funcțiile impuls și triunghiulară */

        /* impuls */
        for(i=0;i<=10;i++) re[i]=1;
        /*sinus */
        /* for(i=0;i<=127;i++) re[i]=0.2*sin(PI/4*i)+0.1*sin(PI/8*i);*/

```



```

/* triunghi
for(i=0;i<10;i++) re[i]=i;
for(i=10;i<20;i++) re[i]=20-i; */
FFT_DT(7,re,im);
for(i=0;i<=127;i++)modul[i]=sqrt( pow(re[i],2)+pow(im[i],2) );

/* inițializare mod grafice */
initgraph(&gdriver, &gmode, "");
errorcode = graphresult();
if (errorcode != grOk)
{
    printf("Eroare grafica: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    getch();
    exit(1);
}
putpixel(4,300-30*modul[1],YELLOW);
for(i=2;i<=127;i++)
{
    lineto(4*i,300-30*modul[i]);
}
getche();
closegraph();
}

```

#### 12.1.1.1.3. Algoritmul în baza 2 cu decimare în frecvență

În acest caz se prezintă indicii sub forma:

$$\begin{aligned}
 n &= n_1 + \frac{N}{2}n_2, \quad n_1 = 0,1,\dots,\frac{N}{2}-1, \quad n_2 = 0,1; \\
 k &= 2k_1 + k_2, \quad k_1 = 0,1,\dots,\frac{N}{2}-1, \quad k_2 = 0,1;
 \end{aligned}
 \tag{12.35}$$

deci

$$X[2k_1 + k_2] = \sum_{n_1=0}^{\frac{N}{2}} \sum_{n_2=0}^1 x[n_1 + \frac{N}{2}n_2] \cdot W_N^{(2k_1+k_2)(n_1+\frac{N}{2}n_2)}
 \tag{12.36}$$

din care rezultă pentru  $k_2 = 0$  și respectiv  $k_2 = 1$

$$\begin{aligned}
 X[2k_1] &= \sum_{n_1=0}^{\frac{N}{2}-1} \{x[n_1] + x[n_1 + \frac{N}{2}]\} W_N^{\frac{n_1 k_1}{2}} = TFD_{\frac{N}{2}} \{x[n_1] + x[n_1 + \frac{N}{2}]\} \\
 X[2k_1 + 1] &= \sum_{n_1=0}^{\frac{N}{2}-1} \{x[n_1] - x[n_1 + \frac{N}{2}]\} W_N^{\frac{n_1}{2}} W_N^{\frac{n_1 k_1}{2}} = TFD_{\frac{N}{2}} \{(x[n_1] - x[n_1 + \frac{N}{2}]) W_n^{\frac{n_1}{2}}\}
 \end{aligned}
 \quad (12.37)$$

Transformata de ordin  $N$  se descompune în două transformate de ordin  $N/2$ . Fiecare din cele două grupuri de câte  $N/2$  semnale ce constituie intrările transformatelor în  $N/2$  puncte se divide în două după același procedeu (primele  $N/4$  și următoarele  $N/4$ ). Vor rezulta un număr de  $\log_2 N$  etaje, în fiecare realizându-se  $N/2$  operații de tip “fluture” structura fluturului fiind dată în figura 12.4.

Complexitatea calculelor aritmetice este dat de :

$$NMR[N] = 2 \log_2 N, \quad NAR[N] = 3N \log_2 N \quad (12.38)$$

în varianta 4/2, ca și la decimarea în timp, și

$$NMR[N] = (3/2)N \log_2 N, \quad NAR[N] = (7/2)N \log_2 N \quad (12.39)$$

pentru varianta 3/3.

Pentru acest algoritm eșantioanele semnalului se iau în ordinea naturală, iar ale ieșirii rezultă în ordinea inversă a biților.

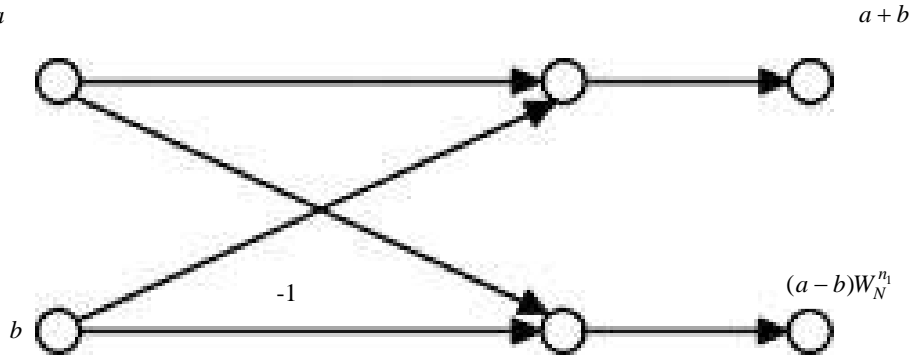


Fig.12.4. Structura fluturului de calcul în algoritmul cu decimare în frecvență

#### 12.1.1.1.4. Programul pentru FFT cu decimare în frecvență

```

#include <math.h>
#include <graphics.h>
#define PI 3.141592653
/* Transformata Fourier rapidă cu decimare în frecvență
   pentru secvențe de tipul 2^m

```

```

    m ordinul secvenței
    x vector cu dimensiunea 2^m care conține la intrare
        partea reală a semnalului în timp, iar la ieșire
        partea reală a semnalului în frecvență
    y vector cu dimensiunea 2^m care conține la intrare
        partea imaginară a semnalului în timp, iar la ieșire
        partea imaginară a semnalului în frecvență
*/
void FFT_DF(int m, double x[], double y[])
{
    int i, j, k, l, n1, n2, n;
    double a, c, s, xt, yt, w;
    n = pow(2, m);
    n2 = n;
    for(k=1; k<=m; k++)
    {
        n1 = n2;
        n2 = n2/2;
        w = PI/n2;
        a = 0;
        for(j=1; j<=n2; j++)
        {
            c = cos(a);
            s = sin(a);
            a = j*w;
            for(i=j; i<=n; i+=n1)
            {
                l = i+n2;
                xt = x[i]-x[l];
                x[i] = x[i]+x[l];
                yt = y[i]-y[l];
                y[i] = y[i]+y[l];
                x[l] = c*xt+s*yt;
                y[l] = c*yt-s*xt;
            }
        }
    }
    j = 1;
    for(i=1; i<=n-1; i++)
    {
        if(i<j)
        {
            xt = x[j];
            x[j] = x[i];
            x[i] = xt;
            xt = y[j];

```

```

        y[j]=y[i];
        y[i]=xt;
    }
    k=n/2;
    while(k<j)
    {
        j-=k;
        k=k/2;
    }
    j+=k;
}
}
void main(void)
{
    int i;
    int gdriver = DETECT, gmode, errorcode;
    double re[128],im[128], modul[128];
    for(i=0;i<=127;i++)
    {
        re[i]=0;
        im[i]=0;
    }

    /* impuls */
    for(i=0;i<=10;i++) re[i]=1;
    /*sinus */
    /* for(i=0;i<=127;i++) re[i]=0.2*sin(PI/4*i)+0.1*sin(PI/8*i); */

    /* triunghi
    for(i=0;i<10;i++) re[i]=i;
    for(i=10;i<20;i++) re[i]=20-i; */
    FFT_DF(7,re,im);
    for(i=0;i<=127;i++)modul[i]=sqrt( pow(re[i],2)+pow(im[i],2) );

    /* inițializare mod grafice */
    initgraph(&gdriver, &gmode, "");
    errorcode = graphresult();
    if (errorcode != grOk)
    {
        printf("Eroare grafică: %s\n", grapherrormsg(errorcode));
        printf("Press any key to halt:");
        getch();
        exit(1);
    }
    putpixel(1,300-30*modul[1],YELLOW);
    for(i=2;i<=127;i++)

```

```

{
    lineto(8*i,300-30*modul[i]);
}
getche();
closegraph();
}

```

### 12.1.2. ALGORITMUL GOERTZEL

Acest algoritm are viteza de calcul inferioară algoritmilor FFT prezentați, dar superioară vitezei de calcul a transformatei DTFT. Codul algoritmului este simplu, ceea ce-l face util în unele aplicații.

Utilizând egalitatea  $W_N^{Nk} = 1$ , expresia transformatei Fourier discrete (12.23) devine:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{-k(N-n)} \quad (12.40)$$

Relația (12.40) poate fi interpretată ca o convoluție discretă a secvenței  $x[n]$  cu secvența  $W_N^{-kn}$ , care reprezintă funcția de transfer a unui filtru. Notând răspunsul filtrului cu  $y_k[n]$  pentru intrarea  $x[n]$  se obține:

$$y_k[n] = \sum_{m=0}^{N-1} x[m] W_N^{-k(n-m)} \quad (12.41)$$

$$\text{Răspunsul filtrului la impulsul treaptă } u[n] \text{ este } h_f[n] = W_N^{-kn} u[n] \quad (12.42)$$

Aplicând transformata z relației (12.42) se obține funcția de transfer a filtrului:

$$H_k(z) = \sum_{n=0}^{\infty} W_N^{-kn} u(n) \cdot z^{-n} \quad (12.43)$$

Ținând cont de valorile lui  $u[n]$  și că suma este o serie geometrică infinită, rezultă:

$$H(z) = \frac{1}{1 - W_N^{-k} \cdot z^{-1}} \quad (12.44)$$

Făcând prelucrări asupra funcției de transfer (12.44), obținem:

$$H_k(z) = \frac{1}{1 - W_N^{-k} \cdot z^{-1}} \cdot \frac{1 - W_N^k \cdot z^{-1}}{1 - W_N^k \cdot z^{-1}} = \frac{1 - W_N^k \cdot z^{-1}}{1 - 2 \cdot \cos\left(\frac{2 \cdot \pi \cdot k}{N}\right) \cdot z^{-1} + z^{-2}} \quad (12.45)$$

Considerând

$$H_k(z) = \frac{Y_k(z) \cdot V_k(z)}{X_k(z) \cdot V_k(z)} \quad \text{unde} \quad \frac{Y_k(z)}{V_k(z)} = 1 - W_N^k \cdot z^{-1} \quad (12.46)$$

rezultă:

$$\frac{V_k(z)}{X_k(z)} = \frac{1}{1 - 2 \cdot \cos\left(\frac{2 \cdot \pi \cdot k}{N}\right) \cdot z^{-1} + z^{-2}} \quad (12.47)$$

Aplicând transformata z inversă relațiilor (12.46) și (12.47) se obțin ecuațiile cu diferențe finite:

$$v_k[n] = 2 \cos\left(\frac{2\pi \cdot k}{N}\right) \cdot v_k[n-1] - v_k[n-2] + x[n]; \quad (12.48)$$

$$y_k[n] = v_k[n] - W_N^k v_k[n-1];$$

Secvența  $y_k[n]$  reprezintă spectrul semnalului  $x[n]$ . Ecuațiile (12.48) pot fi implementate prin schema filtrului numeric de ordinul doi din figura (12.5)

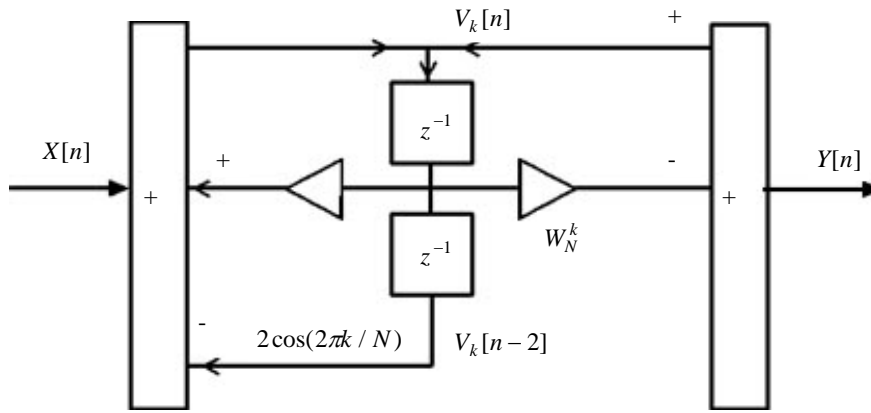


Fig.12.5. Structura filtrului care implementează algoritmul lui Goertzel

### 12.1.2.1. Implementarea algoritmului

```

Void GOERTZEL (double R[],double I[],int N,int K)
{
    double vr1,vr2,vi1,vi2,temp,yfr,yfi;
    double c,s,phi,wn;
    static double REAL[K],IMAG[K],MAG[K];
    int j,k;
    for(k=1;k<=K;k++)
    {
        vr1=vr2=0;
        vi1=vi2=0;
        phi=2*pi*k/N;
        c=cos(phi);
        s=sin(phi);
        for(j=1;j<=N;j++)
        {
            temp=vr1;
            vr1=2*c*vr1-vr2+R[j];
            vr2=temp;
            temp=vi1;

```

```

vi1=2*c*vi1-vi2+I[j];
vi2=temp;
}
yfr=c*vr1-vr2-s*vi1;
yfi=c*vi1-vi2+s*vr1;
REAL[k]=yfr;
IMAG[k]=yfi;
}
//Calculează magnitudinea semnalului de ieșire
for(k=1;k<=K,k++)MAG[k]=sqrt(real[k]*REAL[k]+IMAG[k]*IMAG[k]);
AXE1(k,Mag);//reprezentarea grafică a magnitudinii
getche();
}

```

### 12.1.2.2. Erori de cuantizare în calculul transformatei Fourier discrete

Pentru transformata Fourier discretă trebuie să calculăm suma :

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot W_N^{nk} \quad n = 0, 1, \dots, N-1 \quad (12.49)$$

în care  $\{x[n]\} \subset \mathbb{C}$  și se consideră numerele reprezentate în virgulă fixă, în complement față de doi. Dacă  $x[n]$  este prezentat pe  $B+1$  biți și este utilizat același format pentru reprezentarea coeficienților  $W_N^{nk}$ , după fiecare înmulțire efectuată exact ar trebui mărit formatul cu încă 8 biți. Din economie de memorie nu se procedează astfel, ci se face o rotunjire după fiecare înmulțire la  $B+1$  biți, ceea ce duce la o eroare echivalentă cu un zgomot. Dacă asociem câte o sursă de eroare pentru fiecare înmulțire reală și ținem seama că numerele sunt complexe, rezultă următoarea valoare cuantizată a unui produs:

$$\begin{aligned}
Q(x[n]W_N^{nk}) &= \operatorname{Re}\{x[n]\} \cos \frac{2\pi nk}{N} + e_{n,1}[k] + \operatorname{Im}\{x[n]\} \sin \frac{2\pi nk}{N} + j\{\operatorname{Im}\{x[n]\} \cos \frac{2\pi nk}{N} \\
&+ e_{n,3}[k] - \operatorname{Re}\{x[n]\} \sin \frac{2\pi nk}{N} - e_{n,4}[k]\}
\end{aligned} \quad (12.50)$$

iar eroarea corespunzătoare este:

$$e_n[k] = e_{n,1}[k] + e_{n,2}[k] + j\{e_{n,3}[k] - e_{n,4}[k]\} = e_{n,r}[k] + je_{n,i}[k] \quad (12.51)$$

Dacă referitor la erorile de cuantizare rezultate din înmulțirile reale, se fac ipotezele uzuale pentru zgomotul de rotunjire :

- densitate de probabilitate uniformă între  $-2^{-(B-1)}$  și  $2^{-(B-1)}$ ;
- erorile mai provin de la operații diferite care sunt necorelate;
- erorile sunt necorelate cu semnalul de la intrare

rezultă valorile medii:

$$\begin{aligned}
E\{e_{n,i}[k]\} &= 0 ; E\{e_{n,i}^2[k]\} = \frac{1}{12} \cdot 2^{-2B} = \sigma_{n,i}^2 ; E\{e_n[k]\} = 0 ; \\
E\{|e_n[k]|^2\} &= E\{e_{n,1}^2[k]\} + E\{e_{n,2}^2[k]\} + E\{e_{n,3}^2[k]\} + E\{e_{n,4}^2[k]\} + 2E\{e_{n,1}[k]e_{n,2}[k]\} + \\
&\quad + E\{e_{n,3}[k]e_{n,4}[k]\} = \frac{1}{3} 2^{-2B} = \Delta\sigma_B^2 ,
\end{aligned}$$

ultima expresie reprezentând varianța (dispersia) erorii dintr-un nod oarecare.  
Eroarea totală este caracterizată prin:

$$\begin{aligned}
F[k] &= \sum_{n=0}^{N-1} e_n[k], \quad E\{F[k]\} = 0, \\
E\{|F[k]|^2\} &= E\left\{\left|\sum_{n=0}^{N-1} e_{n,k}[k]\right|^2\right\} = E\left\{\sum_{n=0}^{N-1} e_{n,R}^2[n] + \sum_{n=0}^{N-1} e_{n,I}^2[n]\right\} = \sum_{n=0}^{N-1} E\{|e_n[k]|^2\} = \frac{N}{3} 2^{-2N}
\end{aligned}$$

Lucrându-se în virgulă fixă, trebuie avută în vedere problema scalării.

Ținând cont de relația  $|X[k]| \leq \sum_{n=0}^{N-1} |x[n]| \leq N$  se pot evita depășirile prin scalarea semnalelor de intrare cu  $1/N$ .

## 12.2. APLICAȚII

1. Se consideră semnalul EEG din figura 12.6 și eșantioanele următoare :

119,128,126,131,131,126,136,128,128,129,117,125,125,125,129,122,  
119,119,122,120,131,129,134,140,132,136,132,126,131,120,122,121,  
131,129,128,134,123,129,131,126,131,126,130,133,128,134,128,127,  
129,121,128,124,123,130,121,128,127,126,133,125,128,129,126,132,  
123,127,131,124,131,128,131,134,127,134,129,128,133,125,132,130,  
126,133,125,130,129,120,126,122,128,132,123,129,123,124,131,124,  
130,125,124,133,126,131,131,128,133,126,129,131,124,130,127,130,  
134,125,131,129,127,135,129,132,126,118,126,121,130,133,129,135

luate cu frecvența  $f=128\text{Hz}$  și măsurate în  $\mu\text{V}$ . Se cere analiza în frecvență a semnalului EEG.



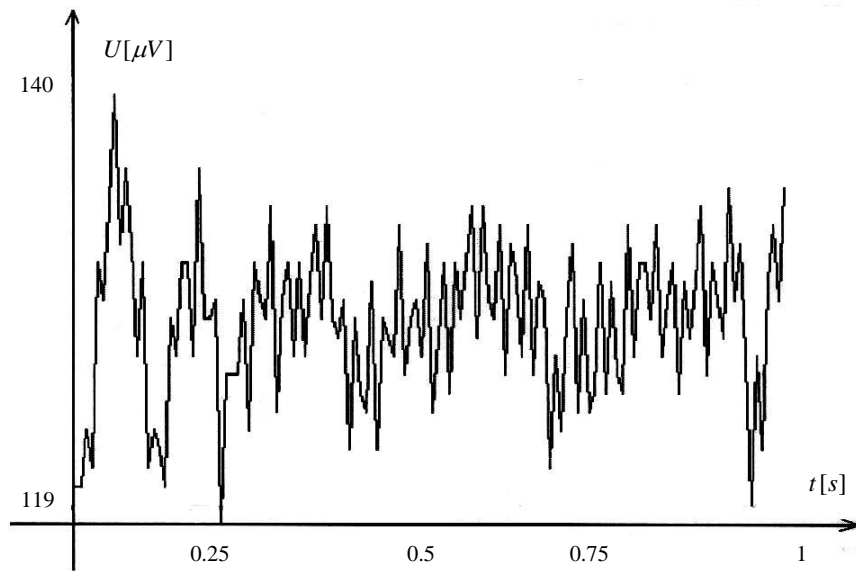


Fig.12.6. Semnal EEG pentru un subiect masculin

Prin algoritmul decimării în timp s-a obținut rezultatul din figura (12.7) iar cu algoritmul Goertzel rezultatul din figura (12.8)

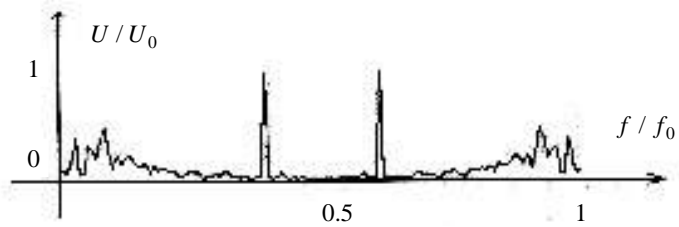


Fig.12.7. Modulul semnalului EEG în frecvență normală prin algoritmul decimării în timp.

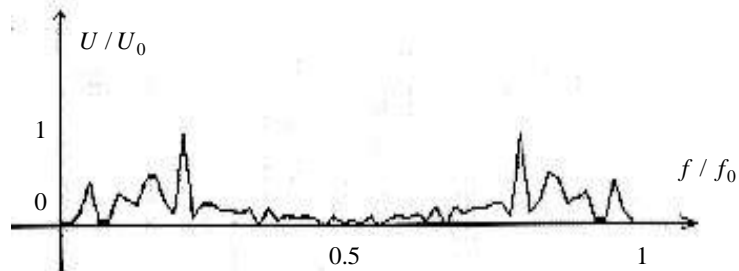


Fig. 12.8. Modulul semnalului EEG în frecvență normală prin algoritmul Goert

2. Să se determine transformata Fourier discretă pentru impulsul treaptă din figura(12.9).

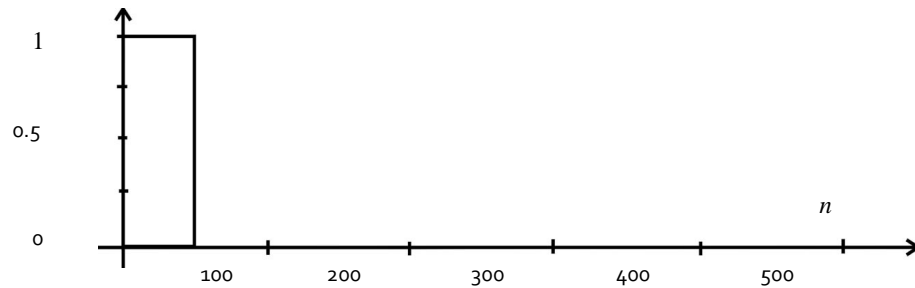


Fig.12.9. Impulsul treaptă.

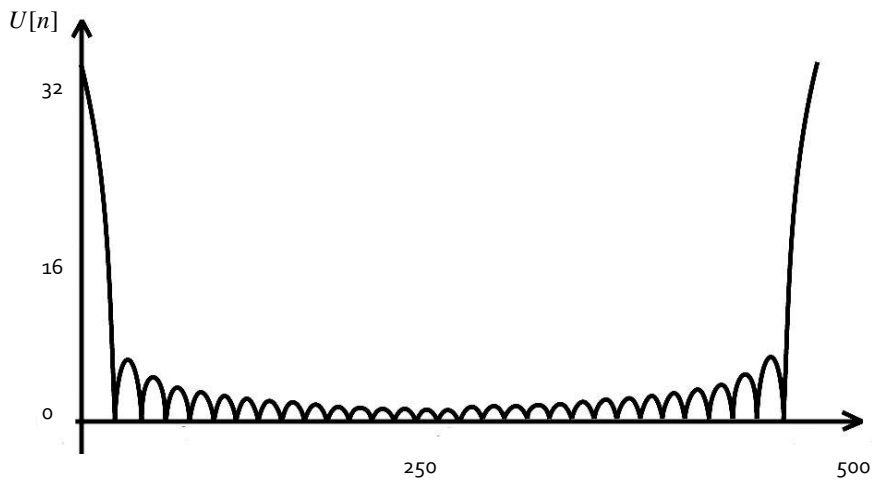


Fig.12.10. Transformata Fourier a impulsului treaptă

Prin analiza figurilor, transformatelor Fourier, a semnalelor se pot trage concluzii asupra comportării în frecvență a semnalelor, ceea ce în medicină înseamnă diagnosticarea unor afecțiuni, iar în tehnică realizarea unor sisteme cu diferite proprietăți.

## ANEXA 1

1. Programul pentru calculul valorii unui polinom într-un punct dat.

### VALPOL

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
#define NrMax 10
```

*/\* Funcția întoarce valoarea polinomului într-un punct dat \*/*

```
double VALPOL(int grad,
               double Coef[NrMax],
               double point)
```

```
{int i;
 double aux;
 aux=Coef[grad];
 for (i=grad-1; i>=0;i-- )
  aux=Coef[i]+point*aux;
 return aux;
}

int main(void)
{
 int n,i;
 double A[NrMax],pct;
 clrscr();
 printf( " Dați gradul polinomului");
 scanf( " %d",&n);
 printf( " Dați coeficienții polinomului \n");
 for (i=n;i>=0;i--)
  {printf( "A[%d]= ",i );
   scanf( " %lf",&A[i]);
  }
 printf( "Dați punctul de calcul");
 scanf( " %lf",&pct);
 clrscr();
 printf( "Valoarea polinomului este
 %6,5lf", VALPOL(n,A,pct));
 getch();
 return 0;
}
```

2. Programul pentru calculul valorii derivatei unui polinom într-un punct dat.

### DERPOL

```
#include <conio.h>
#include <stdio.h>
#include <math.h>
#define NrMax 10
```

*/\* Funcția întoarce valoarea polinomului într-un punct dat \*/*

```
double DERPOL(int grad,
               double Coef[NrMax],
               double point)
```

```
{int i;
 double aux;
 static double B[NrMax];
 B[grad]=aux=Coef[grad];
 for (i=grad-1; i>=0;i-- )
 {B[i]=Coef[i]+point*auxB[i+1];
  aux=B[i]+point*aux;
 }
 return aux;
}
```

```
int main(void)
{
 int n,i;
 double A[NrMax],pct;
 clrscr();
 printf( " Dați gradul polinomului");
 scanf( " %d",&n);
 printf( " Dați coeficienții polinomului \n");
 for (i=n;i>=0;i--)
 {printf( "A[%d]= ",i );
  scanf( "%lf",&A[i]);
 }
 printf( "Dați punctul de calcul");
 scanf( "%lf",&pct);
 clrscr();
 printf( "Valoarea derivatei polinomului este
 %6,5lf",DERPOL(n,A,pct));
 getch();
 return 0;
}
```