# L1. Interpolarea

## *1.1. Polinomul de interpolare Lagrange*

### 1.1.1. Aspecte teoretice

Una dintre cele mai vechi şi generale formule de interpolare este cea datorată lui Lagrange. Mai mult decât în utilitatea practică directă, importanţa ei constă în consecinţele teoretice. O întreagă clasă de formule de integrare numerică are la bază aproximarea funcţiilor prin polinomul de interpolare al lui Lagrange. Pe de altă parte, plecând de la acest interpolant pot fi construite scheme de derivare numerică cu diferite ordine de precizie.

Sa presupunem că, pentru funcţia $f(x)$ care trebuie aproximată, sunt cunoscute $n$ valori corespunzătoare argumentelor $x_1, x_2, \ldots, x_n$ din intervalul $[\alpha, \beta]$:

$$f(x_i) = y_i, \qquad i = 1, 2, \ldots, n.$$

Ne propunem construirea unui polinom $P_m(x)$ care să ia în punctele de tabelare $x_i$ aceleaşi valori ca şi funcţia $f(x)$,

$$P_m(x_i) = y_i, \qquad i=1,2,\ldots,n. \tag{1.1.1}$$

In acest scop să construim mai întâi o familie de polinoame $p_i(x)$, pentru care are loc

$$p_i(x_j) = \delta_{ij} = \begin{cases} 1, j = i \\ 0, j \neq i \end{cases} \tag{1.1.2}$$

unde $\delta_{ij}$ este simbolul lui Kronecker. Deoarece $p_i(x)$ trebuie să se anuleze în toate cele $n$ puncte de tabelare cu excepţia punctului $x_i$, el poate fi scris sub forma unui produs de factori de forma $(x - x_j)$, din care, în particular, lipseşte factorul $(x - x_i)$:

$$p_i(x) = C_i \prod_{j \neq i} (x - x_j).$$

Astfel definit, $p_i(x)$ este un polinom de ordinul $(n-1)$. Luând $x = x_i$ şi având în vedere că $p_i(x_i) = 1$, se găseşte coeficientul constant $C_i = \dfrac{1}{\prod_{j \neq i} (x_i - x_j)}$. Cu acestea, pentru polinomul $p_i(x)$ rezultă expresia

$$p_i(x) = \frac{\prod_{j \neq i} (x - x_j)}{\prod_{j \neq i} (x_i - x_j)}. \tag{1.1.3}$$

Revenind acum la problema iniţială, polinomul $P_m(x)$ care satisface condiţiile de interpolare (1.1.1) poate fi scris ca o combinaţie liniară a polinoamelor $p_i(x)$, pornind de la faptul

că fiecare dintre acestea din urmă se anulează în toate punctele de tabelare cu excepția câte unuia singur - $x_i$. In plus, deoarece polinoamele $p_i(x)$ sunt de ordinul $(n - 1)$, $P_n(x)$ este el însuşi de ordinul $(n - 1)$ şi avem

$$P_{n-1}(x)=\sum_{i=1}^{n} p_i(x)y_i \ . \qquad (1.1.4)$$

Utilizând proprietățile (1.1.2), se poate arăta uşor că sunt satisfăcute condițiile de interpolare (1.1.1):

$$P_{n-1}(x_j)=\sum_{i=1}^{n} p_i(x_j)y_i = \sum_{i=1}^{n} \delta_{ij} y_i = y_j \ , \quad j=1,2,\ldots,n.$$

Inlocuind expresia (1.1.3) a polinoamelor $p_i(x)$ in (1.1.4) rezultă polinomul de interpolare Lagrange:

$$P_{n-1}(x)=\sum_{i=1}^{n} \frac{\prod_{j\neq i}^{n}(x-x_j)}{\prod_{j\neq i}^{n}(x_i-x_j)} y_i \ . \qquad (1.1.5)$$

In general, numărul de puncte utilizate într-o schemă de interpolare minus unu este numit ordinul interpolării şi în cazul de față acesta este egal cu ordinul polinomului de interpolare.

Pentru a demonstra unicitatea polinomului de interpolare Lagrange vom presupune contrariul, adică faptul că există un polinom $\overline{P}_{n-1}(x)$ diferit de $P_{n-1}(x)$, care satisface de asemenea condițiile de interpolare

$$\overline{P}_{n-1}(x_i)=y_i, \quad i=1,2,\ldots,n.$$

In consecință, polinomul

$$Q_{n-1}(x)=\overline{P}_{n-1}(x)-P_{n-1}(x)$$

se anulează în toate cele $n$ puncte de interpolare $x_i$. Fiind însă de ordinul $(n - 1)$, nu are decât $(n - 1)$ zerouri, şi deci rezultă a fi identic nul. Prin urmare, $\overline{P}_{n-1}(x) \equiv P_{n-1}(x)$

In cazul particular $n = 2$ formula lui Lagrange se reduce la

$$P_1(x)=\frac{x-x_2}{x_1-x_2}y_1+\frac{x-x_1}{x_2-x_1}y_2 \qquad (1.1.6)$$

şi descrie dreapta care trece prin cele două puncte de tabelare. Pentru $n = 3$ interpolantul Lagrange corespunde parabolei definite de cele trei puncte de tabelare:

$$P_2(x)=\frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)}y_1+\frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)}y_2+\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}y_3. \qquad (1.1.7)$$

Următoarea funcție implementează formula (1.1.5) a polinomului de interpolare Lagrange.

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

double Lagrange(double xi[], double yi[], int ni, double x)
/*-------------------------------------------------------------------
Evalueaza polinomul de interpolare Lagrange
xi[] - abscisele punctelor interpolate
yi[] - ordonatele punctelor interpolate
ni - numarul punctelor interpolate
x - argumentul polinomului
-------------------------------------------------------------------*/
{
        double p, y;
        int i, j;
        y = 0.0;
        for (i = 0; i < ni; i++)
        {
                p = 1.0;
                for (j = 0; j < ni; j++)
                        if (j != i)
                                p *= (x - xi[j]) / (xi[i] - xi[j]);
                y += p * yi[i];
        }
        return y;
}
void main()
{
        double x[8] = {0.15, 0.2, 0.3, 0.5, 0.8, 1.1, 1.4, 1.7};
        double y[8];
        for (int i = 0; i < 8; i++)
                y[i] = 1 / x[i];
        printf("Valoarea interpolantului in punctul 1.3 este %lf\n",Lagrange(x, y, 8, 1.3));
}
```

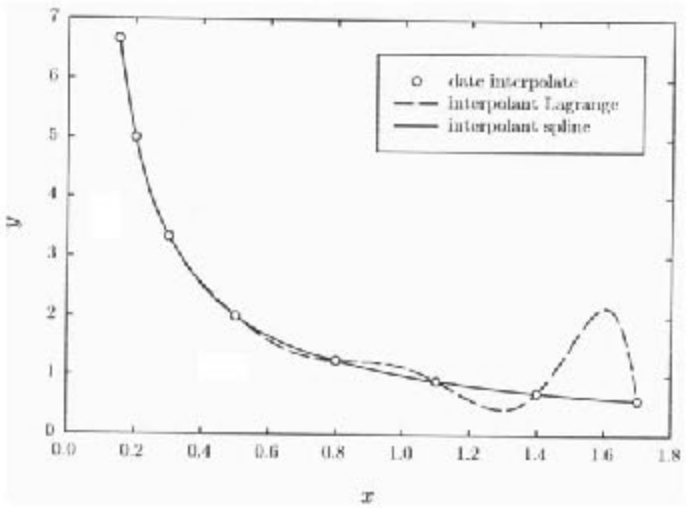


FIGURA 1.1.1

Exemplu în care interpolarea cu ajutorul polinomului lui Lagrange conduce la oscilaţii. Datele interpolate provin din eşantionarea funcţiei $f(x)=1/x$

Relatia (1.1.5) evidenţiază faptul că aproximarea realizată cu ajutorul polinomului de interpolare Lagrange este esenţialmente *nelocală,* în sensul că la evaluarea interpolantului contribuie informaţii din *toate* punctele tabelate, nu numai din cele din vecinătatea argumentului considerat. Nelocalitatea garantează pe de o parte "netezimea" aproximării prin continuitatea derivatelor, însă, pe de alte parte, poate induce oscilaţii complet străine de comportarea reala a funcţiei aproximate.

Una dintre cauzele apariţiei oscilaţiilor polinomului Lagrange o constituie dispunerea neadecvată a punctelor de tabelare. Pentru ilustrarea acestui fenomen a fost considerată funcţia $f(x) = 1/x$, pentru care, aşa cum se vede din figura 1.1.1, au fost generate puncte de interpolare corespunzătoare argumentelor 0.15, 0.2, 0.3, 0.5, 0.8, 1.1, 1.4 şi 1.7. Distanţa relativ mare dintre abscisele ultimelor puncte face ca interpolantul Lagrange să nu poată reproduce funcţia modelată în acest domeniu şi să prezinte oscilaţii din ce in ce mai ample. Distribuind însă în mod echidistant abscisele, oscilaţiile se reduc până aproape de dispariţie. Mai mult, adăugând încă un singur punct de tabelare, tot in condiţiile dispunerii echidistante a absciselor, oscilaţiile pot fi eliminate complet în acest caz. Trebuie avut în vedere totuşi că, neînsoţite de o dispunere judicioasă a absciselor, creşterea numărului punctelor de tabelare nu îmbunătăţeşte întotdeauna precizia interpolării, deoarece, în general, prin creşterea corespunzătoare a ordinului polinomului de interpolare apar zerouri suplimentare şi deci oscilaţii suplimentare ale interpolantului.

Prin constrast cu comportamentul interpolantului Lagrange în exemplul considerat mai sus, din figura 1.1.1 se poate constata că aproximarea realizată cu ajutorul funcţiilor spline cubice, nu prezintă oscilaţii. Calitativ, acest comportament diferit poate fi explicat prin faptul ca restricţiile interpolantului spline între punctele de tabelare sunt polinoame de ordin scăzut, cu număr mic de zerouri şi deci tendinţă redusă de oscilaţie.

Pe baza formulei de interpolare Lagrange se poate rezolva şi *problema interpolării inverse,* adică găsirea argumentului pentru care polinomul Lagrange are o anumită valoare. Pentru aceasta este suficient sa se considere $y$ ca variabilă independentă şi să se scrie, prin analogie cu relaţia (1.1.5), o formulă exprimând $x$ ca funcţie de $y$:

$$x = \sum_{i=1}^{n} \frac{\prod_{j \neq i} (y - y_i)}{\prod_{j \neq i} (y_i - y_j)} y_i . \qquad (1.1.8)$$

Cu ajutorul acestei formule se poate găsi o rădăcină aproximativă a ecuaţiei $f(x) = 0$. In acest scop se calculează un tablou de valori $y_i$ pentru $n$ argumente $x_i$ apropiate de rădăcină. Punând apoi $y = 0$ în relaţia (1.1.8), se găseşte rădăcina respectivă. Dacă $f(x) = P_{n-1}(x)$ este un polinom de ordinul $n$ - 1, rădăcina determinată prin această metodă este exactă.

### 1.1.2. Desfăşurarea laboratorului

1. să se scrie algoritmul de calcul al polinomului de interpolare Lagrange pentru funcţia f(x) = 1/x şi să se afişeze valoarea interpolantului în trei puncte la alegere. Corectitudinea valorilor obţinute se va aprecia prin comparare cu valoarea calculată a funcţiei în punctele respective. Se vor considera 8 puncte de interpolare distribuite aleator în intervalul (0,2].
2. să se scrie algoritmul de calcul al polinomului de interpolare Lagrange pentru funcţia f(x) = (1-x)/5 şi să se afişeze valoarea interpolantului în trei puncte la alegere. Corectitudinea valorilor obţinute se va aprecia prin comparare cu valoarea calculată a funcţiei în punctele respective. Se vor considera 10 puncte de interpolare distribuite aleator în intervalul (0,2].
3. să se scrie algoritmul de calcul al polinomului de interpolare Lagrange pentru funcţia f(x) = 1/(x-3) şi să se afişeze valoarea interpolantului în trei puncte la alegere. Se vor considera 8 apoi 15 puncte de interpolare distribuite uniform în intervalul (0,2] şi se va face comparaţie între valorile obţinute în cele două cazuri.

## 1.2. *Metoda Neville*

### 1.2.1. Aspecte teoretice

Metoda Neville nu constituie o abordare distinctă a problemei interpolării, ci reprezintă mai degrabă un algoritm optim de construire a polinomului de interpolare unic printr-un număr dat de puncte, care este polinomul Lagrange. Un dezavantaj al utilizării directe a formulei (1.1.5) a polinomului Lagrange este acela că nu oferă o estimare a erorii, decât prin compararea valorii interpolanților pentru seturi de puncte de tabelare care se subîntind. Acest mod de lucru nu permit însă refolosirea rezultatelor obținute pentru un anumit interpolant în evaluarea interpolanților de ordin superior.

Metoda Neville implică în esență calculul iterativ al interpolanților de ordine consecutive, construiți relativ la subseturi crescânde de puncte de interpolare, cu refolosirea în cel mai înalt grad a rezultatelor anterioare și furnizând în mod natural o estimare a erorii aproximării.

Fie funcția $f(x)$ specificată prin cele $n$ valori corespunzătoare șirului de argumente $x_1, x_2, \ldots, x_n$:

$$f(x_i) = y_i, \quad i = 1,2,\ldots,n.$$

Pentru a construi eficient polinomul Lagrange care interpolează toate cele $n$ puncte de tabelare $(x_i, y_i)$, și modelează funcția $f(x)$, definim mai întâi o familie de polinoame de interpolare, astfel ca $P_{i,j}(x)$ să fie polinomul unic care interpolează succesiunea de puncte cu abscisele cuprinse intre $x_i$ și $x_j$, adică:

$$P_{i,j}(x_k) = y_k, \quad i \le k \le j \tag{1.2.1}$$

Desigur, interpolând $j-i+1$ puncte, polinomul $P_{i,j}(x)$ este de ordinul $j-i$. In particular, $P_{i,i}(x)$ este polinomul de ordin zero care interpolează numai punctul $(x_i,y_i)$, adică funcția constantă care ia pentru orice argument valoarea $y_i$. Pentru ansamblul polinoamelor $P_{i,i}(x)$ avem atunci:

$$P_{i,i}(x) = y_i, \quad i = 1,2,\ldots,n. \tag{1.2.2}$$

Evident, polinomul Lagrange căutat poate fi identificat cu $P_{1,n}(x)$, care este polinomul de interpolare unic în raport cu întregul set de date:

$$P_{n-1}(x) = P_{1,n}(x). \tag{1.2.3}$$

Evaluarea polinomului $P_{1,n}(x)$ pentru un anumit argument $x$ poate fi realizată în mod optim aranjând valorile $P_{i,j}(x)$ potrivit următorului tablou cu "predecesori" și "descendenți", pe fiecare coloană aflându-se polinoamele de același ordin $m$ (egal cu diferența $j-i$ dintre cei doi indici):

| | $m=0$ | 1 | 2 | ... | $n-2$ | $n-1$ |
|---|---|---|---|---|---|---|
| $x_1$ | $P_{1,1}$ | $P_{1,2}$ | $P_{1,3}$ | ... | $P_{1,n-1}$ | $P_{1,n}$ |
| $x_2$ | $P_{2,2}$ | $P_{2,3}$ | ... | $P_{2,n-1}$ | $P_{2,n}$ | |
| $x_3$ | $P_{3,3}$ | ... | ... | ... | | |
| ... | ... | $P_{n-2,n-1}$ | $P_{n-2,n}$ | | | |
| $x_{n-1}$ | $P_{n-1,n-1}$ | $P_{n-1,n}$ | | | | |
| $x_n$ | $P_{n,n}$ | | | | | |

Metoda Neville este propriu-zis un algoritm de completare recursivă a coloanelor tabloului - de la stânga la dreapta, pornind de la polinoamele de ordin zero $P_{i,i}(x)$ şi conducând pe ultima coloană la un singur descendent, care este valoarea căutată $P_{1,n}(x)$. Propagarea valorilor se poate efectua pe baza relaţiei dintre un anumit descendent $P_{i,j}(x)$ şi cei doi predecesori ai săi de ordin inferior, $P_{i,j-1}(x)$ şi $P_{i+1,j}$ (caracterizaţi prin aceeaşi diferenţă $j-i-1$ între indici), situaţi unul sub altul pe coloana anterioară:

$$P_{i,j}(x)=\frac{(x-x_j)P_{i,j-1}(x)+(x_i-x)P_{i+1,j}(x)}{x_i-x_j}. \qquad (1.2.4)$$

Intr-adevăr, pentru orice $x_k$ cu $i<k<j$ sunt satisfăcute condiţiile de interpolare, deoarece predecesorii $P_{i,j-1}(x)$ şi $P_{i+1,i}(x)$ satisfac ei înşişi aceste condiţii pentru subsetul comun de puncte de interpolare:

$$P_{i,j}(x_k)=\frac{(x_k-x_j)y_k+(x_i-x_k)y_k}{x_i-x_j}=y_k.$$

In particular, pentru punctele extreme $x_i$ şi $x_j$ are loc

$$P_{i,j}(x_i)=\frac{(x_i-x_j)y_i+(x_i-x_i)P_{i+1,j}(x_i)}{x_i-x_j}=y_i,$$

$$P_{i,j}(x_j)=\frac{(x_j-x_j)P_{i,j-1}(x_j)+(x_i-x_j)y_j}{x_i-x_j}=y_j.$$

Prin urmare, polinomul $P_{i,j}(x)$ definit de relaţia de recurenţă (1.2.4) interpolează toate cele $j-i+1$ puncte $(x_i,y_i),\ldots,(x_j,y_j)$ şi, fiind un polinom de ordinul $j-i$, reprezintă într-adevăr polinomul unic de interpolare, adică polinomul Lagrange.

Având în vedere că metoda Neville implică parcurgerea *pe coloane* a tabloului de valori $P_{i,j}(x)$ şi că indicii polinoamelor de pe o anumită coloană $m$ sunt legaţi prin relaţia $j=m+i$, formula de recurenţă (1.2.4) poate fi rescrisă sub o formă mai adecvată pentru implementări prin introducerea notaţiilor $P_i^{(m)}(x)=P_{i,j}(x)$ şi $x_{m+i}=x_j$, care pun în evidenţă indicele de coloană şi, implicit, ordinul polinoamelor:

$$P_i^{(m)}(x) = \frac{(x - x_{m+i})P_i^{m-1}(x) + (x_i - x)P_{i+1}^{(m-1)}(x)}{x_i - x_{m+i}}, \qquad (1.2.5)$$

*m=1,2,...,n-1, i=1,2,...,n-m.*

Indicele de coloană *m* poate fi parcurs în mod independent, iar pentru fiecare coloană indicele de linie *i* ia *n-m* valori, cu una mai puțin decât pentru coloana anterioară. In implementări concrete este suficientă utilizarea unui tablou unidimensional, în ale cărui componente sunt memorate valorile polinoamelor $P_i^{(m)}(x)$ de pe coloana curentă.

O estimare utilă pentru eroarea aproximării funcției tabelate prin polinomul Lagrange $P_{n-1}(x) = P_1^{(n-1)}(x)$ este furnizată de diferența ultimilor doi predecesori:

$$\Delta = \left| P_1^{(n-2)}(x) - P_2^{(n-2)}(x) \right|.$$

Semnificația argumentelor următoarei implementări a metodei Neville este în esență aceeași cu cea din cazul rutinei Lagrange, prezentată în secțiunea anterioară: $x_i$ și $y_i$ - tablouri care conțin perechile de coordonate ale punctelor de interpolare, *ni* - numărul punctelor de interpolare și *x* - argumentul pentru care se evaluează polinomul de interpolare. Parametrul **err* returnează, în plus, eroarea aproximării.

```
#include "stdio.h"
#include "stdlib.h"
#include "math.h"

double Neville(double xi[], double yi[], int ni, double x, double *err)
/*----------------------------------------------------------------------------
Evalueaza polinomul de interpolare Lagrange prin metoda Neville si returneaza o estimare a erorii absolute
xi[] - abscisele punctelor interpolate
yi[] - ordonatele punctelor interpolate
ni - numarul punctelor interpolate
x - argumentul polinomului
*err - estimarea erorii absolute a valorii polinomului (iesire)
----------------------------------------------------------------------------*/
{
        double *p, y;
        int i, m;
        p = (double*)malloc(ni * sizeof(double));

        for (i = 0; i < ni; i++)            /* initializeaza tabloul cu */
                p[i] = yi[i];               /* polinoamele de ordin 0   */
        for (m = 1; m < ni - 1; m++)        /* parcurge coloanele tabloului */
```

```
            for (i = 0; i < ni - m; i++)
                p[i] = ((x - xi[m + i]) * p[i] + (xi[i] - x) * p[i + 1]) / (xi[i] - xi[m + i]);
        y = p[1];                       /* valoarea polinomului */
        *err = fabs(p[1] - p[2]);       /* estimarea erorii */
        free(p);
        return y;
}

void main()
{
        double x[8] = {0.15, 0.2, 0.3, 0.5, 0.8, 1.1, 1.4, 1.7};
        double y[8];
        double err=0;

        for (int i = 0; i < 8; i++)
            y[i] = 1 / x[i];
        printf("Valoarea interpolantului in punctul 1.55 este %lf,", Neville(x, y, 8, 1.55, &err));
        printf(" cu o valoare a erorii de %.5lf\n", err);
}
```

### 1.2.2. Desfășurarea laboratorului

1. să se scrie algoritmul metodei Neville pentru funcția f(x) = 1/x și să se afișeze valoarea interpolantului și a erorii în trei puncte la alegere. Se vor considera 8 puncte de interpolare distribuite aleator în intervalul (0,2].
2. să se scrie algoritmul metodei Neville pentru funcția f(x) = (7-x)/23 și să se afișeze valoarea interpolantului și a erorii în trei puncte la alegere. Se vor considera 10 puncte de interpolare distribuite aleator în intervalul (0,2].
3. să se scrie algoritmul metodei Neville pentru funcția f(x) = 1/(x-3) și să se afișeze valoarea interpolantului și a erorii în trei puncte la alegere. Se vor considera 15 apoi 20 puncte de interpolare distribuite uniform în intervalul (0,2] și se vor compara valorile erorilor obținute în cele două cazuri.

# L2. Calculul integralelor unidimensionale

## *2.1. Formulele de cuadratură Newton-Cotes*

### 2.1.1. Aspecte teoretice

Să presupunem că pentru o funcţie y = f(x) se cere calculul integralei definite

$$I = \int_a^b f(x)dx. \tag{2.1.1}$$

Impărţim intervalul [a,b] în (n-1) subintervale egale, de lungime

$$h = (b-a)/(n-1) \tag{2.1.2}$$

prin punctele

$$x_i = a + (i-1)h, i = 1,2,...,n \tag{2.1.3}$$

şi presupunând că sunt cunoscute valorile $f_i \equiv f(x_i)$ în nodurile $x_i$, vom aproxima funcţia $f(x)$ prin polinomul de interpolare Lagrange

$$P_{n-1}(x) = \sum_{i=1}^{n} \frac{\prod_{j\neq i}^{n}(x - x_j)}{\prod_{j\neq i}^{n}(x_i - x_j)} f_i.$$

Introducând variabila adimensională

$$q = (x-a)/h, \quad q \in [0, n-1],$$

argumentele funcţiilor pot fi exprimate sub forma $x = a + qh$ şi produsele din expresia polinomului de interpolare Lagrange devin

$$\prod_{j\neq i}^{n}(x - x_j) = h^{n-1}\prod_{j\neq i}^{n}[q - (j-1)]$$

$$\prod_{j\neq i}^{n}(x_i - x_j) = h^{n-1}\prod_{j\neq i}^{n}(i-j) = (-1)^{n-1}h^{n-1}\prod_{j=1}^{i-1}(i-j)\prod_{j=i+1}^{n}(j-i) = (-1)^{n-i}h^{n-1}(i-1)!(n-i)!$$

Cu acestea, polinomul Lagrange se scrie

$$P_{n-1}(x) = \sum_{i=1}^{n} \frac{\prod_{j\neq i}^{n}[q - (j-1)]}{(-1)^{n-i}(i-1)!(n-i)!} f_i \tag{2.1.4}$$

şi obţinem următoarea aproximaţie a integralei căutate

$$\int_a^b f(x)dx \approx \int_a^b P_{n-1}(x)dx = \sum_{i=1}^{n} A_i f_i \tag{2.1.5}$$

cu coeficienţi $A_i$ daţi de

$$A_i = \int_a^b \frac{\prod_{j\neq i}^{n}[q - (j-1)]dx}{(-1)^{n-i}(i-1)!(n-i)!} = \frac{h\int_0^{n-1}\prod_{j\neq i}^{n}[q - (j-1)]}{(-1)^{n-i}(i-1)!(n-i)!} \tag{2.1.6}$$

Punând aceşti coeficienţi sub forma $A_i = (b-a)H_i$, rezultă din (2.1.5) *formula de cuadratură Newton-Cotes,*

$$\int_a^b f(x)dx \approx (b-a)\sum_{i=1}^{n} H_i f_i \qquad (2.1.7)$$

în care coeficienţii $H_i$, numiţi *coeficienţi Cotes,* au expresiile

$$H_i = \frac{\int_0^{n-1} \prod_{j\neq i}^{n} [q-(j-1)]dq}{(-1)^{n-1}(i-1)!(n-i)!(n-1)}, \quad i = 1,2,\ldots,n. \qquad (2.1.8)$$

Se remarcă faptul că, potrivit definiţiei de mai sus, coeficienţii Cotes sunt independenţi atât de funcţia integrată cât şi de intervalul de integrare. Trebuie menţionat, de asemenea, că au loc proprietăţile:

$$\sum_{i=1}^{n} H_i = 1, \quad H_i = H_{n-i+1}. \qquad (2.1.9)$$

Prima dintre aceste relaţii poate fi demonstrată imediat punând în formula de cuadratură (2.1.7) $f(x) \equiv 1$, iar cea de a doua proprietate rezultă chiar din expresia (2.1.8) a coeficienţilor.

## *2.2. Formula trapezelor*

### 2.2.1. Aspecte teoretice

Particularizând relaţiile (2.1.8) pentru cazul n=2, obţinem coeficienţii Cotes

$$H_1 = -\int_0^1 (q-1)dq = \frac{1}{2} \qquad (2.2.1)$$

$$H_2 = \int_0^1 q dq = \frac{1}{2}, \qquad (2.2.2)$$

şi înlocuindu-i în formula de cuadratură (2.1.7), rezultă *formula trapezului:*

$$\int_{x_1}^{x_2} f(x)dx \approx \frac{h}{2}(f_1 + f_2) \qquad (2.2.3)$$

Numele formulei se datorează faptului că, aşa cum se vede în figura de mai jos, ea poate fi obţinută prin înlocuirea funcţiei $f(x)$ cu segmentul care uneşte punctele de coordonate $(x_1, f_1)$ şi $(x_1, f_2)$, valoarea integralei fiind aproximată de aria trapezului determinat prin proiectarea acestui segment pe axa absciselor.

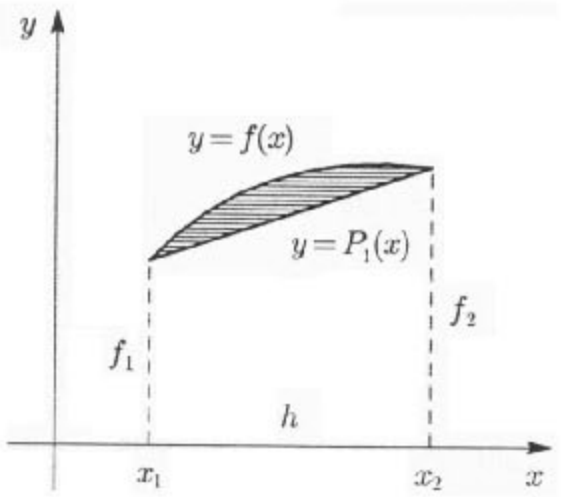


Fig. 2.2.1 Formula trapezului aproximează funcţia prin dreapta care trece prin punctele $(x_1,f_1)$ şi $(x_2,f_2)$.

Restul (eroarea) formulei de cuadratură (2.2.3), figurat haşurat în figura 2.2.1, poate fi dedus presupunând că $f(x)$ este continuă împreună cu primele două derivate pe $[a,b]$ $\left(f(x)dx \in C^{(2)}[a,b]\right)$. Astfel, exprimând $R$ ca funcţie de $h$ avem:

$$R(h)=\int_{x_1}^{x_1+h} f(x)dx-\frac{h}{2}[f(x_1)+f(x_1+h)]$$

Derivând această relaţie de două ori în raport cu $h$, obţinem:

$$R'(h)=\frac{1}{2}[f(x_1+h)-f(x_1)]-\frac{h}{2}f'(x_1+h)$$

$$R''(h)=-\frac{h}{2}f''(x_1+h).$$

Integrăm acum $R''(h)$ de două ori în raport cu $h$, observând că R(0) = R'(0) = 0 şi utilizând *teorema mediei*. Rezultă succesiv:

$$R'(h)=R'(0)+\int_0^h R''(t)dt=-\frac{1}{2}f''(\xi_1)\int_0^h tdt=-\frac{h^2}{4}f''(\xi_1)$$

$$R(h)=R(0)+\int_0^h R'(t)dt=-\frac{1}{4}f''(\xi)\int_0^h t^2dt=-\frac{h^3}{12}f''(\xi),$$

unde $\xi,\xi_1\in(x_1,x_1+1)$. Astfel, avem pentru restul formulei de cuadratură a trapezului

$$R=-\frac{h^3}{12}f''(\xi),\quad \xi\in(x_1,x_2). \qquad (2.2.4)$$

Se constată că restul are semn opus derivatei secunde din intervalul $(x_1,x_2)$ şi, prin urmare, formula trapezului supraestimează valoarea integralei dacă $f''>0$ şi o subestimează în caz contrar.

Formula trapezului (2.2.3) nu prezintă interes ca atare, deoarece, implicând doar două valori ale integrandului, oferă precizie satisfăcătoare numai pentru integrarea funcţiilor liniare. Utilizând însă proprietatea de *aditivitate* a integralelor faţă de intervalul de integrare, vom generaliza acest rezultat în cele ce urmează pentru a obţine o formulă de cuadratură de interes practic.

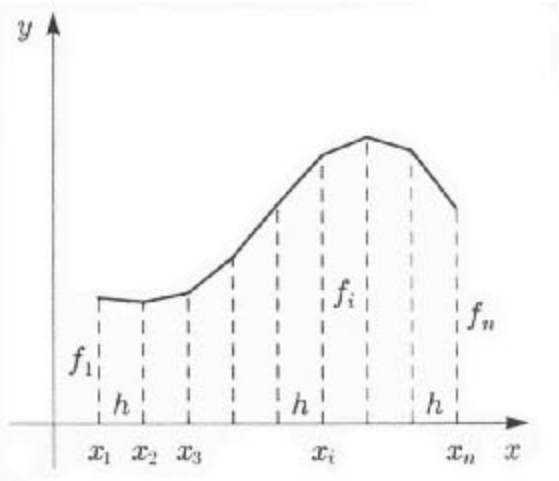


Figura 2.2.2. Formula trapezelor aproximează integrând cu linia poligonală definită de abscisele şi valorile corespunzătoare ale funcţiei

In acest scop împărţim, aşa cum se vede în figura 2.2.2, intervalul [*a*,*b*] în (*n*-1) subintevalele egale de lungime

$$h = (b-a)/(n-1) \qquad (2.2.5)$$

prin intermediul punctelor echidistante

$$x_i = a + (i-1)h, \quad i = 1,2,...,n \qquad (2.2.6)$$

Desemnând prin $f_i \equiv f(x_i)$ valorile integrandului în nodurile rețelei, aplicăm formula trapezului (2.2.3) fiecărui subinterval $[x_1, x_2],...,[x_{n-1}, x_n]$,

$$\int_a^b f(x)dx \approx \frac{h}{2}(f_1 + f_2) + \frac{h}{2}(f_2 + f_3) + ... + \frac{h}{2}(f_{n-1} + f_n)$$

și, grupând termenii, obținem *formula trapezelor:*

$$\int_a^b f(x)dx \approx h\left[\frac{f_1}{2} + \sum_{i=2}^{n-1} f_i + \frac{f_n}{2}\right]. \qquad (2.2.7)$$

Geometric, această aproximație implică înlocuirea graficului funcției $y = f(x)$ cu linia poligonală care unește punctele $(x_1, f_1), (x_2, f_2),..., (x_n, f_n)$.

Restul formulei trapezelor se compune din resturile corespunzătoare celor ($n$-1) subintervale de integrare,

$$R = -\frac{(n-1)h^3}{12} f''(\xi) = h\left[\frac{f_1}{2} + \sum_{i=2}^{n-1} f_i + \frac{f_n}{2}\right] \qquad (2.2.8)$$

și prezintă o dependență tipică de $h^2$. Această dependență are ca și consecință faptul că, spre exemplu, prin reducerea lungimii subintervalelor la valoarea $h/2$ sau, echivalent, prin dublarea numărului de subintervale considerând ($2n$-1) puncte de integrare, eroarea de integrare scade de 4 ori.

Funcția *Trapez*, prezentată în continuare, calculează integrale definite prin metoda trapezelor. Datele de intrare ale rutinei sunt limitele domeniului de integrare, $a$ și $b$, și numărul punctelor de integrare, $n$. Numele funcției care trebuie integrată este transmis ca parametru corespunzător parametrului formal *Func*. Valoarea integralei este returnată prin numele funcției.

```
double Trapez (double a, double b, int n)
/* Calculeaza integrala functiei Func pe intervalul [a,b]
utilizand formula trapezelor cu n puncte de integrare */
{
     double h, s;
     int i;

     h = (b - a) / (n - 1);
     s = 0.5 * (Func(a) + Func(b));
     for (i = 1; i <= (n - 2); i++)
          s += Func(a + i * h);

     return h * s;
}
```

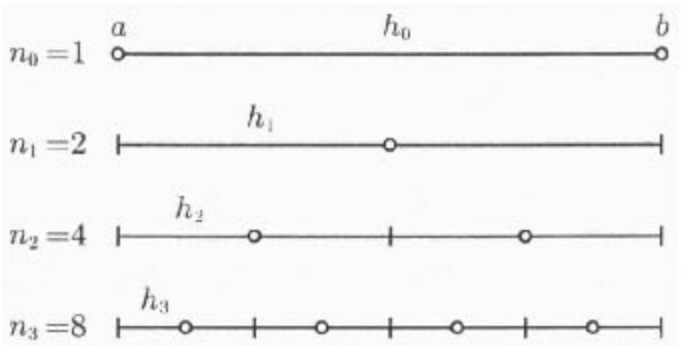


Figura 2.2.3. Schema de înjumătățire a intervalelor în metoda trapezelor cu control automat al pasului. Prin cerințe sunt figurate valorile care trebuie calculate în fiecare etapă

Pentru generarea absciselor de integrare, adunarea repetată a lungimii pasului, $x_{i+1} = x_i + h$, poate provoca acumularea de erori în cazul numerelor mari de puncte de integrare, fiind deci preferabilă utilizarea relației

$$x_{i+1} = a + ih, \quad i = 0,1,\ldots,n\text{-}1.$$

In plus, valori generate cu ajutorul acesteia din urmă pot fi comunicate direct ca argumente în apelul funcției *Func*.

Controlul pasului şi implicit al preciziei de integrare se realizează în practică adesea prin proces de „exprimare" numerică. Acest proces implică de obicei calculul integralei pentru o valoare oarecare $h$ a pasului, corespunzând unui număr $n$ de puncte de integrare, şi compararea valorii astfel obținute cu valoarea rezultată pentru pasul $h/2$, corespunzând unui număr de $(2n\text{-}1)$ puncte de integrare. Injumătățirea pasului trebuie continuată, în principiu, până când eroarea relativă pentru două aproximații consecutive ale integralei scade sub o valoare admisibilă $\varepsilon$.

Calculul integralei cu o precizie prestabilită prin metoda înjumătățirii pasului poate fi semnificativ optimizat observând că, potrivit figurii 2.2.3, care reprezintă rețelele punctelor de integrare pentru primele patru etape ale procesului de înjumătățire, nu este necesară evaluarea funcției în toate punctele de integrare, ci doar în cele (figurate prin cercuri) prin care au fost înjumătățite subintervalele rețelei utilizate în etapa anterioară. Restul valorilor funcției, deja calculate la pașii anteriori, sunt folosite implicit stabilind o relație de recurență între două aproximații consecutive ale integralei.

Primele aproximații furnizate de formula trapezelor pot fi scrise:

$$T_0 = h_0\left[\frac{f(a)}{2} + \frac{f(b)}{2}\right], \quad h_1 = b - a$$

$$T_1 = h_1\left[\frac{f(a)}{2} + f(a+h_1) + \frac{f(b)}{2}\right], \quad h_1 = h_0/2 \qquad (2.2.9)$$

$$T_2 = h_2\left[\frac{f(a)}{2} + f(a+h_2) + f(a+2h_2) + f(a+3h_2) + \frac{f(b)}{2}\right], \quad h_2 = h_1/2 \qquad (2.2.10)$$

.....

Se poate verifica fără dificultate că acestă succesiune de aproximații poate fi descrisă cu ajutorul procesului iterativ

$$T_0 = \frac{h}{2}\left[f(a) + f(b)\right], \quad h_0 = b - a, n_0 = 1 \qquad (2.2.11)$$

$$T_k = \frac{1}{2}\left[T_{k-1} + h_{k-1}\sum_{i=1}^{n_{k-1}} f\left(a + (i - 1/2)h_{k-1}\right)\right], \qquad (2.2.12)$$

$$h_k = h_{k-1}/2, n_k = 2n_{k-1}, k = 1,2,\ldots \qquad (2.2.13)$$

unde $h_k = (b-a)/n_k$ este lungimea subintervalelor după etapa $k$, iar $n_k = 2^k$ reprezintă numărul corespunzător de subintervale sau, echivalent, numărul punctelor noi de integrare şi de înjumătăţire a subintervalelor pentru etapa următoare.

Procesul recurent (2.2.11) - (2.2.13) trebuie continuat până când diferenţa relativă dintre două aproximaţii succesive ale integralei devine mai mică sau egală cu o toleranţă prestabilită $\varepsilon$. Pentru a evita împărţirea cu 0 implicată de evaluarea erorii relative în cazul $T_k = 0$, exprimăm criteriul de convergenţă sub forma

$$|T_k - T_{k-1}| \le \varepsilon |T_k|. \qquad (2.2.14)$$

Algoritmul (2.2.11) – (2.2.14) al metodei trapezului cu control automat al pasului de integrare se regăseşte sub forma funcţiei *TrapezControl* care urmează:

```
#include <stdio.h>
#include <math.h>
#define Pi 3.14159265358979323846264338327795

double Func(double x)
{
     return atan(x);
}

double TrapezControl(double a, double b)
/* Calculeaza integrala functiei Func pe intervalul [a,b]
utilizand formula trapezelor cu control automat al pasului de
integrare */
{
     const double eps = 1e-6;
     const int kmax = 30;
     double h, sum, t, t0;
     long i, n;
     int k;

     h = b - a;
     n = 1;
     t0 = 0.5 * h * (Func(a) + Func(b));     //aprox. initiala

     for (k = 1; k <= kmax; k++)
     {
          sum = 0.0;
          for (i = 1; i <= n; i++)
               sum += Func(a + (i - 0.5) * h);
          t = 0.5 * (t0 + h * sum);
          if (fabs(t - t0) <= eps * fabs(t))
               break;
          h *= 0.5;
          n *= 2;
          t0 = t;
     }

     if (k >= kmax)
```

```
        printf("TrapezControl: nr. maxim de iteratii
depasit!\n");

    return t;
}

void main()
{
    printf("integrala functiei atan(x) pe intervalul [-pi/2
3pi/2] are valoarea: %.8f\n",TrapezControl(-Pi/2, 3*Pi/2));
}
```

Parametri formali ai acestei rutine au aceeaşi semnificaţie cu cei ai funcţiei *Trapez*. Pentru ca numărul punctelor noi de integrare n, ca putere a lui 2, să nu depăşească valoarea maximă de tip *long* reprezentabilă (pentru majoritatea implementărilor limbajului C egală cu $2^{31}-1$), numărul de înjumătăţiri ale subintervalelor este limitat prin variabila *kmax* la 30. Dacă, datorită convergenţei slabe a procesului, se epuizează cele *kmax* iteraţii posibile fără a se fi atins precizia dorită *eps* în calculul integralei, se iese din ciclul înjumătăţirilor cu variabila de control *k* incrementată la valoarea *kmax*+1 şi se emite un mesaj de eroare.

## *2.3. Formula lui Simpson*

### 2.3.1. Aspecte teoretice

O formulă de cuadratură cu un grad de precizie mai ridicat decât formula trapezului se obţine particularizând formulele Newton – Cotes pentru $n = 3$. Se obţin coeficienţii Cotes:

$$H_1 = \frac{1}{4}\int_0^2 (q-1)(q-2)dq = \frac{1}{6} \qquad (2.3.1)$$

$$H_2 = -\frac{1}{2}\int_0^2 q(q-2)dq = \frac{2}{3} \qquad (2.3.2)$$

$$H_3 = \frac{1}{4}\int_0^2 q(q-1)dq = \frac{1}{6} \qquad (2.3.3)$$

şi, având în vedere că $b-a \equiv x_3 - x_1 = 2h$, rezultă *formula lui Simpson*:

$$\int_{x_1}^{x_3} f(x)dx \approx \frac{h}{3}(f_1 + 4f_2 f_3). \qquad (2.3.4)$$

Din punct de vedere geometric, această formulă implică înlocuirea curbei $y=f(x)$ cu parabola $y = P_2(x)$ definită de punctele $(x_1, f_1), (x_2, f_2)$ şi $(x_3, f_3)$.

Admiţând că $f(x)$ aparţine clasei funcţiilor continue împreună cu primele patru derivate pe $[a,b]\left(f(x) \in C^4[a,b]\right)$, se poate obţine o estimare a restului formulei lui Simpson printr-o tehnică similară celei utilizate în cazul formulei trapezului,

$$R = -\frac{h^5}{90} f^{(4)}(\xi), \xi \in [a,b] \qquad (2.3.5)$$

Este important de observat că restul formulei lui Simpson depinde de $h^5$, în timp ce restul formulei trapezului depinde doar de $h^3$. Se constată că formula Simpson este exactă nu numai pentru polinoame de ordinul doi, ci şi pentru cele de ordinul trei.

Pentru a stabili o formulă de interes practic, care să asigure o precizie satisfăcătoare pentru un integrand arbitrar, se generalizează relaţia (2.3.4), similar formulei trapezelor,

făcând uz de *aditivitatea* integralei față de intervalul de integrare. Astfel, divizând intervalul $[a,b]$ printr-un număr *impar*, $n=2m+1$, de puncte echidistante,

$$x_i = a + (i-1)h, \quad i = 1,2,...,n,$$

caracterizate prin echidistanța

$$h = \frac{b-a}{n-1} = \frac{b-a}{2m},$$

se poate aplica formula lui Simpson (2.3.4) pentru fiecare din cele $m$ subintervale duble de lungime $2h$ determinate de cele trei puncte de rețea consecutive: $[x_1, x_3], [x_3, x_5], ..., [x_{n-2}, x_n]$. Geometric, această abordare revine la aproximarea integrandului prin arce de parabolă pe fiecare pereche de subintervale consecutive. Este evident că pentru un număr par de puncte de rețea, formula lui Simpson nu poate fi aplicată de un număr întreg de ori. Cu toate acestea, integrala pe întregul interval $[a,b]$ poate fi scrisă:

$$\int_a^b f(x)dx \approx \frac{h}{3}(f_1 + 4f_2 + f_3) + \frac{h}{3}(f_3 + 4f_4 + f_5) + ... + \frac{h}{4}(f_{n-2} + 4f_{n-1} + f_n).$$

Regrupând termenii rezultă de aici *formula lui Simpson generalizată*,

$$\int_a^b f(x)dx \approx \frac{h}{3}(f_1 + 4\sigma_2 + 2\sigma_1 + f_n), \tag{2.3.6}$$

unde

$$\sigma_1 = \sum_{i=1}^{(n-3)/2} f_{2i+1}, \quad \sigma_2 \sum_{i=1}^{(n-1)/2} f_{2i} \tag{2.3.7}$$

Suma $\sigma_1$ se compune din termenii de indice impar, în timp ce $\sigma_2$ însumează termenii cu indice par.

Admițând că $f(x) \in C^{(4)}[a,b]$, restul formulei lui Simpson generalizate se obține însumând resturile de tipul (4.3.5) pentru fiecare din cele $m$ subintervale de lungime $2h$ ale intervalului $[a,b]$:

$$R = -\frac{mh^5}{90} f^{(4)}(\xi) = -\frac{(b-a)h^4}{180} f^{(4)}(\xi), \quad \xi \in [a,b]. \tag{2.3.8}$$

Funcția *Simpson0*, listată în continuare, codifică întocmai formulele (2.3.6) – (2.3.7). Semnificația parametrilor este aceeași cu cea a parametrilor funcției *Trapez*, descrisă anterior, și numele funcției utilizator este transmis și aici prin lista de argumente.

```
float Simpson0 (float Func(float), float a, float b, int n)
/* Calculeaza integrala functiei Func pe intervalul [a,b]
utilizand formula lui Simpson cu n puncte de integrare */
{
     float h, s1, s2;
     int i, par = 0;

     if ((n/2)*2 == n) n++;    //incrementeaza n daca este par

     h = (b - a) / (n - 1);
     s1 = s2 = 0.0;
     for (i = 1; i <= (n - 2); i++) {
          if (par) s1 += Func(a + i * h);
            else s2 += Func(a + i * h);
          par = ~par;
     }
```

```
    return (h/3) * (Func(a) + 4*s2 + 2*s1 + Func(b));
}
```

In situațiile în care în mod eronat numărul punctelor de integrare *n*, comunicat rutinei, este par, el este incrementat. Adunarea alternativă a termenilor la o sumă sau alta este controlată de valorile variabilei întregi *par*, care, prin acțiunea operatorului „~" de incrementare bit-cu-bit, ia alternativ valori nule și nenule.

Se poate realiza o codificare mai eficientă rescriind relațiile (2.3.7) sub forma

$$\sigma_1 = \sum_{i=1}^{(n-3)/2} f\left(x_{2i+1}\right),\ x_{2i+1} = a + i(2h) \qquad (2.3.9)$$

$$\sigma_2 = f(x_2) + \sum_{i=1}^{(n-3)/2} f\left(x_{2(i+1)}\right),\ x_{2(i+1)} = x_{2i+1} + h\,, \qquad (2.3.10)$$

astfel încât numărul termenilor din cele două sume să fie egal, indicele *i* parcurgând același șir de valori. In implementarea dată mai jos nu mai este necesară structura de selecție *if*, deoarece la fiecare iterație este adunat câte un termen fiecărei sume.

```
float Simpson (float Func(float), float a, float b, int n)
/* Calculeaza integrala functiei Func pe intervalul [a,b]
utilizand formula lui Simpson cu n puncte de integrare */
{
    float h, h2, s1, s2, x;
    int i;

    if ((n/2)*2 == n) n++;    //incrementeaza n daca este par

    h = (b - a) / (n - 1); h2 = 2*h;
    s1 = 0.0; s2 = Func(a+h);
    for (i = 1; i <= (n - 3)/2; i++) {
        x = a + i*h2;
        s1 += Func(x); s2 += Func(x+h);
    }

    return (h/3) * (Func(a) + 4*s2 + 2*s1 + Func(b));
}
```

Ca și în cazul metodei trapezelor, se poate concepe un algoritm având la bază formula lui Simpson, care să realizeze în mod automat adaptarea pasului de integrare pentru efectuarea cuadraturii cu o anumită precizie prestabilită ε. In esență, trebuie comparate aproximațiile calculate recent ale integralei, corespunzătoare valorilor *h* și *h/2* ale pasului de integrare, înjumătățind pasul până când eroarea relativă pentru două aproximații consecutive scade sub valoarea ε.

Având în vedere aproximațiile date de formula trapezelor pentru valori ale pasului de integrare înjumătățite succesiv, se poate verifica fără dificultate că aproximațiile formulei lui Simpson pot fi exprimate în raport cu cele dintâi:

$$S_1 = \frac{h_1}{3}\left[f(a) + 4f(a+h_1) + f(b)\right] = \frac{4T_1 - T_0}{3}$$

$$S_2 = \frac{h_2}{3}\left[f(a) + 4f(a+h_2) + 2f(a+h_2) + 4f(a+3h_2) + f(b)\right] = \frac{4T_2 - T_1}{3}$$

..... (2.3.11)

Generalizând aceste rezultate, avem

$$S_k = \frac{4T_k - T_{k-1}}{3} \qquad (2.3.12)$$

Cu acestea, algoritmul pentru calculul recursiv al aproximaţiilor formulei lui Simpson poate fi descris cu ajutorul relaţiilor:

$$T_0 = \frac{h_0}{2}\left[f(a)+f(b)\right], \quad h_0 = b-a, \quad n_0 = 1 \qquad (2.3.13)$$

$$T_k = \frac{1}{2}\left[T_{k-1} + h_{k-1}\sum_{i=1}^{n_{k-1}} f\left(a+(i-1/2)h_{k-1}\right)\right], \quad k = 1,2,... \qquad (2.3.14)$$

$$S_k = \frac{4T_k - T_{k-1}}{3}, \quad h_k = h_{k-1}/2, \quad n_k = 2n_{k-1}. \qquad (2.3.15)$$

Acest proces este continuat până când eroarea relativă în calculul integralei devine mai mică sau egală cu toleranţa prestabilită ε. Ca şi în cazul metodei trapezelor, pentru a trata unitar şi situaţiile în care $S_k = 0$, scriem acest criteriu sub forma:

$$\left|S_k - S_{k-1}\right| \le \varepsilon\left|S_k\right|. \qquad (2.3.16)$$

Argumentele, variabilele şi constantele locale ale funcţiei *SimpsonControl*, scrisă pe baza acestui algoritm, păstrează semnificaţia pe care o au în cazul funcţiei *TrapezControl*.

```
#include <stdio.h>
#include <math.h>
#define Pi 3.14159265358979323846264338327795

float func(float x)
{
     return atan(x);
}

float SimpsonControl(float Func(float), float a, float b)
/* Calculeaza integrala functiei Func pe intervalul [a,b]
utilizand formula lui Simpson cu control automat al pasului de
integrare */
{
     const float eps = 1e-6;  //precizia relativa a integralei
     const int kmax = 30;   //numar maxim de injumatatiri
     float h, s, s0, sum, t, t0;
     long i, n;
     int k;

     h = b - a;
     n = 1;
     s0 = t0 = 0.5 * h *(Func(a) + Func(b)); //aprox. initiala

     for (k = 1; k <= kmax; k++)
     {
          sum = 0.0;
          for (i = 1; i <= n; i++)
               sum += Func(a + (i - 0.5) * h);
          t = 0.5 * (t0 + h*sum);
```

```
            s = (4*t - t0)/3;                    //noua aproximatie
            if (fabs(s - s0) <= eps * fabs(s))
                 break;                          //testeaza convergenta
            h *= 0.5;
            n *= 2; s0 = s;
            t0 = t;
      }

      if (k >= kmax)
            printf("SimpsonControl:   nr.   maxim   de   iteratii
depasit!\n");

      return s;
}

void main()
{
      printf("integrala functiei atan(x) pe intervalul [-pi/2
3pi/2] are valoarea: %.8f\n",SimpsonControl(func,-Pi/2,
3*Pi/2));
}
```

### 2.3.2. Desfășurarea laboratorului

1. Se va aplica algoritmul metodei trapezelor şi algoritmul metodei lui Simpson, ambele cu control automat al pasului de integrare, pentru a calcula valoarea integralelor de la punctele a)...e). Precizia relativă va fi 6 zecimale iar rezultatele se vor afişa cu 8 zecimale. Se va alcătui un tabel cu diferențele între valoarea integralei calculată direct şi cea rezultată în urma aplicării metodei trapezelor respectiv Simpson pentru fiecare din cele 5 integrale.

a) $\int_2^7 \frac{1}{x}dx = \ln|x|\Big|_2^7$

b) $\int_{-1}^{1} x^{10}dx = \frac{x^{11}}{11}\Big|_{-1}^{1}$

c) $\int_{-5}^{0} e^x dx = e^x\Big|_{-5}^{0}$

d) $\int_0^{\pi} \sin x dx = -\cos x\Big|_0^{\pi}$

e) $\int_{-\frac{\pi}{2}}^{\frac{3\pi}{2}} \arctan x dx = x\arctan x - \frac{1}{2\ln(1+x^2)}\Big|_{-\frac{\pi}{2}}^{\frac{3\pi}{2}}$

# L3. Rezolvarea numerică a ecuațiilor diferențiale ordinare

## *3.1. Metoda Euler*

### 3.1.1. Aspecte teoretice

Problemele modelate cu ecuații diferențiale ordinare se caracterizează printr-o singură variabilă independentă și una sau mai multe variabile dependente. În aplicațiile științifice și tehnice, variabila dependentă este timpul (t) sau spațiul (x). Indiferent de ordinul ecuației diferențiale, care rezultă din modelarea matematică a problemei, rezolvarea numerică se poate reduce întotdeauna la o ecuație diferențială de ordinul I.

De exemplu, o ecuație diferențială de ordinul II:

$$\frac{d^2y}{dx^2} + q(x)\frac{dy}{dx} = r(x)$$

este echivalentă cu un sistem de două ecuații diferențiale de ordinul I:

$$\frac{dy}{dx} = z(x)$$

$$\frac{dz}{dx} = r(x) - q(x)z(x)$$

În general, o ecuație diferențială nu are soluție unică. În aplicațiile concrete, se poate obține soluție unică dacă se cunosc valorile inițiale ale funcției necunoscute (*problemă Cauchy*) sau valorile pe frontiera domeniului de definiție (*problemă Dirichlet*).

Formularea completă a unei probleme ce constă în rezolvarea unei ecuații diferențiale de ordinul I (*problemă Cauchy*) este:

$$y' = f(x,y)$$

$$y(x_0) = y_0 \quad ,$$

unde $x_0$ este valoarea inițială a variabilei independente.

Metoda Euler este foarte importantă, nu atât din punct de vedere practic (se acumulează erori mari de aproximare) cât mai ales conceptual, pentru înțelegerea metodelor numerice evoluate, de mare eficiență.

Fie problema Cauchy:

$$y' = f(x,y)$$

$$y(x_0) = y_0, \qquad (3.1.1)$$

Dezvoltăm funcția *y(x)* în serie Taylor, într-o vecinătate a punctului $x_0$:

$$y(x) = y(x_0) + y'(x_0)(x - x_0) + \frac{(x - x_0)^2}{2!}y''(x_0) + \ldots \quad ,$$

Introducem pasul de discretizare $h = x - x_0$ și rescriem relația de mai sus neglijând termenii de ordin superior lui 2:

$$y(x) = y(x_0) + h \cdot y'(x_0) \quad .$$

Termenii din partea dreaptă sunt cunoscuți din relațiile (3.1.1) iar valoarea lui $y(x)$ se evaluează în $x = x_0 + h$.

$y(x_0+h)=y(x_0)+h\cdot f(x_0,y_0)$

Notăm $y(x_0)=y_0,\quad y(x_0+k\cdot h)=y_k$, $k=1,2,\ldots,n$, şi obţinem:

Pasul 1: $y_1=y_0+h\cdot f(x_0,y_0)$,

Pasul 2: $y_2=y_1+h\cdot f(x_1,y_1)$

Pasul 3: $y_3=y_2+h\cdot f(x_2,y_2)$

. . . . . .

Pasul n+1: $y_{n+1}=y_n+h\cdot f(x_n,y_n)$, unde $x_n=x_0+n\cdot h$.

Funcţia necunoscută $y(x)$ se determină astfel prin valorile sale în puncte echidistante: $y_1,\ y_2,\ y_3,\ \ldots\ y_n$.

Programul în C++ care determină valorile funcţiei *y(x)* este listat mai jos.

```
#include <stdio.h>
 float f(float x, float y) //funcţia f(x,y)
    {
    return y / (y-x);
    }

 int main(void)
 {
   float x=0.0,  y=1.0, h=0.5; // valorile iniţiale x0,y0,h
   int n=0;
   printf("n=%3d x=%8.4f y=%8.4f \n",n,x,y);

   for(n=1;n<=10;n++) {
     y=y+h*f(x,y);            // relaţia yn+1=yn+h·f(xn,yn)
     x=x+h;
     printf("n=%3d x=%8.4f y=%8.4f \n",n,x,y);
     }
     return 0;
}
```

### *3.2. Metoda Euler modificată*

#### 3.2.1. Aspecte teoretice

Dezavantajul principal al algoritmului Euler îl constituie eroarea de calcul cumulată (determinată de neglijarea termenilor de ordin superior lui 2). Pentru a reduce această eroare se impune folosirea unui pas de discretizare *h*, foarte mic, ceea ce duce la creşterea timpului de execuţie. Agoritmul lui Euler poate fi mai eficient dacă se utilizează în calcul, valoarea funcţiei *f(x,y)* la ambele capete ale intervalui de discretizare *h*. De fapt, la fiecare pas, valoarea $y_{n+1}$ se calculează în două etape. Prima etapă este cea din algoritmul Euler iar în a doua etapă se efectuează o corecţie a valorii obţinute.

Pas 1: $y_1 = y_0 + h \cdot f(x_0, y_0)$,
$y_1' = f(x_1, y_1)$;

Pas 1 – corecție: $y_1 = y_0 + h \cdot (y_0' + y_1')/2$,
$y_1' = f(x_1, y_1)$;

Pas 2: $y_2 = y_1 + h \cdot f(x_1, y_1)$,
$y_2' = f(x_2, y_2)$;

Pas 2 – corecție: $y_2 = y_1 + h \cdot (y_1' + y_2')/2$,
$y_2' = f(x_2, y_2)$;

. . . . . .

Pas n+1: $y_{n+1} = y_n + h \cdot f(x_n, y_n)$,
$y_{n+1}' = f(x_n, y_n)$;

Pas n+1 – corecție: $y_{n+1} = y_n + h \cdot (y_n' + y_{n+1}')/2$,
$y_{n+1}' = f(x_{n+1}, y_{n+1})$;

unde $x_n = x_0 + n \cdot h$.

În etapa a II-a, se corectează valoarea lui $y_{n+1}$ considerând media derivatelor în punctele $x_n$ și $x_{n+1}$. Pentru creșterea preciziei, secvența de corecție se poate relua încă o dată. Cerința de a folosi un pas de discretizare h, cât mai mic se menține și în cazul metodei Euler modificate.

Programul în C++ care determină valorile funcției *y(x)* este listat în continuare. Problema Cauchy din program este:

$y' = \frac{y}{y - x}$; $x \in [0, \infty)$

$y_0 = y(0) = 1$; (Soluția analitică, exactă, este $y(x) = x + \sqrt{x^2 + 1}$).

```
// Algoritmul Euler -modificat
#include <stdio.h>
#include <math.h>
float f(float x, float y)
      {
      return y/(y-x);
      }

int main(void)
{
float x=0.0,  y=1.0, h=0.05, y1, y2;
int n=0;

printf("n=%3d x=%8.4f y=%8.4f \n",n,x,y);
   for(n=1;n<=50;n++) {
      y1=f(x,y);
```

```
        y=y+h*f(x,y);
        x=x+h;y2=f(x,y);
        y=y+h*(y1+y2)/2;
        printf("n=%3d x=%8.4f y=%8.4f  val_ex%8.4f \n",
                                    n,x-h,y,x+sqrt(x*x+1));
        }
return 0;}
```

### *3.3. Metode Runge – Kutta*

#### 3.3.1. Aspecte teoretice

Se bazează pe evaluări ale funcției *f(x,y)* (cunoscută ca expresie analitică) și derivatelor sale în raport cu variabila independentă *x*, în mai multe puncte ale intervalului de integrare. Se asigură astfel o precizie mai bună, ca urmare a faptului că funcția *f(x,y)* este dată iar derivatele de ordin superior ale funcției necunoscute *y(x)* sunt exprimate prin derivate parțiale de ordin superior ale lui *f(x,y(x))*.

Fie problema Cauchy:

$y'(x) = f(x,y);$

$y(x_0) = y_0;$

Se evaluează derivatele de ordin superior ale funcției *y(x)*:

$y'(x) = f(x,y);$

$$y''(x) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\cdot\frac{dy}{dx} = f_x(x,y) + f_y(x,y)\cdot f(x,y) = f_x + f_y f \tag{3.3.1}$$

$$y'''(x) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial x \partial y}\frac{dy}{dx} + \frac{\partial^2 f}{\partial x \partial y}\frac{dy}{dx} + \frac{\partial^2 f}{\partial y^2}\left(\frac{dy}{dx}\right)^2 + \frac{\partial f}{\partial y}\frac{d^2 y}{dx^2} =$$

$$= f_{xx} + f_{xy}y' + f_{xy}y' + f_{yy}(y')^2 + f_y y'' = f_{xx} + 2ff_{xy} + f_{yy}f^2 + f_y\left(f_x + ff_y\right) = \tag{3.3.2}$$

$$= f_{xx} + 2ff_{xy} + f_{yy}f^2 + f_x f_y + ff_y^2.$$

Algoritmul Runge – Kutta de ordinul 2, se bazează pe dezvoltarea funcției *y(x)* în serie Taylor luând în considerație termenii până la ordinul 2 de derivare. Formulele de bază, iterative, sunt:

$$y_{n+1} = y_n + ak_1 + bk_2 \quad unde \tag{3.3.3}$$

$k_1 = hf(x_n, y_n),$

$k_2 = hf(x_n + \alpha\cdot h, y_n + \beta\cdot k_1)$

Constantele $a, b, \alpha, \beta$, urmează a fi determinate din condiția ca (5.3.3) să reprezinte o dezvoltare în serie Taylor a funcției *y(x)* în jurul punctului $x_n$, pentru $x = x_{n+1}$:

$$y(x_{n+1}) = y(x_n) + hy'(x_n) + \frac{h^2}{2}y''(x_n) + \frac{h^3}{6}y'''(x_n) + \cdots\cdots =$$

$$= y(x_n) + hf(x_n, y_n) + \frac{h^2}{2}\left(f_x + ff_y\right)_n + \frac{h^3}{6}\left(f_{xx} + 2ff_{xy} + f_{yy}f^2 + f_x f_y + ff_y^2\right)_n + \cdots \tag{3.3.4}$$

Cu indicele *n* la paranteze, am indicat că evaluarea funcției se face în $(x_n, y_n)$.

Pe de altă parte, folosind dezvoltarea în serie Taylor pentru funcţii de două variabile, în jurul punctului $(x_n, y_n)$, obţinem:

$$f(x_n + \alpha \cdot h, y_n + \beta \cdot k_1) = f(x_n, y_n) + \alpha \cdot h f_x + \beta \cdot k_1 f_y + \frac{\alpha^2 h^2}{2} f_{xx} + \alpha \cdot h \cdot \beta \cdot k_1 f_{xy} + \frac{\beta^2 k_1^2}{2} f_{yy} + \cdots \quad (3.3.5)$$

unde toate derivatele au fost evaluate în punctul $(x_n, y_n)$.

În expresia (3.3.3), înlocuim parametrii $k_1$ , $k_2$ folosind expresia (3.3.5):

$$y_{n+1} = y_n + (a+b)h \cdot f + b \cdot h^2 (\alpha \cdot f_x + \beta \cdot f f_y) + bh^3 \left( \frac{\alpha^2}{2} f_{xx} + \alpha\beta f_{xy} + \frac{\beta^2}{2} f^2 f_{yy} \right) + \cdots \quad (3.3.6)$$

Comparând expresiile (3.3.4) cu (3.3.6) şi identificând coeficienţii pentru $h$ şi $h^2$ obţinem relaţiile:

$$\begin{aligned} a + b &= 1 \\ b \cdot \alpha &= b \cdot \beta = \frac{1}{2} \end{aligned} \quad (3.3.7)$$

Sistemul (3.3.7) are mai multe soluţii, fiind format din trei ecuaţii cu patru necunoscute. O soluţie simplă şi echilibrată este

$$a = b = \frac{1}{2}, \qquad \alpha = \beta \quad (3.3.8)$$

**Algoritmul Runge – Kutta de ordinul 2:**

Pentru problema Cauchy:

$y'(x) = f(x,y)$;

$y(x_0) = y_0$;

Se alege pasul de integrare $h$, $x_n = x_0 + n \cdot h$ şi se calculează pentru $n=1, 2, 3, \ldots N$, valorile funcţiei $y(x_n)$ :

$k_1 = hf(x_n, y_n)$,

$k_2 = hf(x_n + h, \quad y_n + k_1)$

$y_{n+1} = y_n + \frac{1}{2} k_1 + \frac{1}{2} k_2$.

Algoritmul este de "ordinul 2" pentru că în seriile Taylor se iau în consideraţie termenii ce conţin derivate până la ordinul 2, inclusiv.

Algoritmul Runge – Kutta de ordinul 4 este similar dar se iau în consideraţie termenii ce conţin derivate până la ordinul 4, inclusiv; în final, din identificarea termenilor ce conţin $h, h^2, h^3, h^4$, se obţin 11 ecuaţii cu 13 necunoscute, deci două necunoscute se aleg arbitrar.

**Algoritmul Runge – Kutta de ordinul 4:**

Pentru problema Cauchy:

$y'(x) = f(x,y)$;

$y(x_0) = y_0$;

Se alege pasul de integrare $h$, $x_n = x_0 + n \cdot h$ şi se calculează pentru $n=1, 2, 3, \ldots N$, valorile funcţiei $y(x_n)$ :

$$k_1 = hf(x_n, y_n),$$
$$k_2 = hf\left(x_n + \frac{1}{2}h, \quad y_n + \frac{1}{2}k_1\right),$$
$$k_3 = hf\left(x_n + \frac{1}{2}h, \quad y_n + \frac{1}{2}k_2\right),$$
$$k_4 = hf(x_n + h, \quad y_n + k_3),$$
$$y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4).$$

Algoritmul Runge – Kuta devine şi mai performant prin controlul adaptiv al pasului de iteraţie, h; când viteza de variaţie a funcţiei y(x) creşte, trebuie micşorat pasul iar când funcţia are o variaţie lentă, pasul poate fi mărit. Viteza de variaţie a unei funcţii este dată de valoarea derivatei sale.

```
      // Runge-Kutta ordin 2:
#include <stdio.h>
#include <math.h>
float f(float x, float y)
      {
      return y/(y-x);
      }

int main(void)
{
float x=0.0,  y=1.0, h=0.05, k1, k2 ;
 int n=0;
 printf("n=%3d x=%8.4f y=%8.4f \n",n,x,y);

   for(n=1;n<=50;n++) {
       k1=h*f(x,y);
       k2=h*f(x+h,y+k1);
       y=y+(k1+k2)/2;
       x=x+h;
       printf("n=%3d x=%8.4f y=%8.4f
val_ex:%8.4f\n",n,x,y,x+sqrt(x*x+1));
       }
       return 0;
}
```

```
 // Runge-Kutta de ordin 4

#include <stdio.h>
#include <math.h>
float f(float x, float y)
      {
      return y/(y-x);
      }
```

```
int main(void)
{
float x=0.0,  y=1.0, h=0.05, k1, k2, k3, k4 ;
 int n=0;
 printf("n=%3d x=%8.4f y=%8.4f \n",n,x,y);

   for(n=1;n<=50;n++) {
      k1=h*f(x,y);
      k2=h*f(x+h/2.0,y+k1/2.0);
      k3=h*f(x+h/2.0,y+k2/2.0);
      k4=h*f(x+h,y+k3);
      y=y+(k1+2*k2+2*k3+k4)/6.0;
      x=x+h;
      printf("n=%3d x=%8.4f y=%8.4f
val_ex:%8.4f\n",n,x,y,x+sqrt(x*x+1));
      }
      return 0;
     }
```

### 3.3.2. Desfășurarea laboratorului

1. Se va aplica algoritmul metodei Euler modificată și algoritmul metodei Runge – Kutta pentru a rezolva ecuațiile diferențiale de la punctele a)...c). La fiecare ecuație, pentru soluțiile ultimilor 4 pași, se determină eroarea procentuală față de valoarea exactă după formula:

$$procent = \left| \frac{exact - aprox}{exact} \right| \times 100 .$$

a) $y' = \frac{3-4y}{2x}$, $y(1) = -4$; soluția analitică: $y(x) = \frac{3}{4} - \frac{19}{4x^2}$

b) $y' = \frac{x}{y}$, $y(2) = -1$; soluția analitică: $y(x) = -\sqrt{x^2 - 3}$

c) $y' = 2 - e^{-4x} - 2y$, $y(0) = 1$; soluția analitică: $y(x) = 1 + \frac{1}{2}e^{-4x} - \frac{1}{2}e^{-2x}$

# L4. Cuadraturi gaussiene

## *4.1. Formula Gauss-Legendre*

### 4.1.1. Aspecte teoretice

Calculul unei integrale cu ajutorul oricărei formule Newton-Cotes implică valori ale integrandului corespunzătoare unei rețele de puncte echidistante. Scheme de ordin diferit se deosebesc în esență prin numărul punctelor de integrare (nu prin modul lor de alegere) şi prin ponderile asociate. Prin comparație, ideea de bază a metodelor de integrare gaussiene constă atât în alegerea optimă a ponderilor, cât şi a nodurilor de integrare, astfel încât să rezulte formule de cuadratură cu grad de precizie maxim pe anumite clase de funcții (de exemplu, pe clasa polinoamelor algebrice). Având la dispoziție un număr dublu de grade de libertate care pot fi fixate (noduri şi ponderi), se pot obține formule de cuadratură cu ordin de precizie dublu față de formulele Newton-Cotes cu acelaşi număr de termeni.

Deoarece orice integrală pe un interval finit [a,b] poate fi transformată printr-o schimbare liniară de variabilă într-o integrală pe intervalul [-1,1],

$$\int_a^b f(\xi)d\xi = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}x+\frac{b+a}{2}\right)dx, \tag{4.1.1}$$

rezultă că formulele de cuadratură pentru intervale finite pot fi dezvoltate, fără a le reduce generalitatea, pentru intervalul standard [-1,1].

Ne punem problema alegerii punctelor de integrare $x_1$, $x_2$, ..., $x_n$ şi a ponderilor $w_1$, $w_2$, ..., $w_n$ astfel încât formula de cuadratură

$$\int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} w_i f(x_i) \tag{4.1.2}$$

să fie *exactă* pentru toate polinoamele de grad $N$ cât mai mare posibil. Deoarece această formulă implică $2n$ coeficienți care trebuie determinați ($w_i$, $x_i$, $i=1, 2, ..., n$) şi acelaşi număr de coeficienți caracterizează complet un polinom de grad $(2n-1)$, este evident că gradul maxim este $N = 2n-1$.

Pentru ca formula de cuadratură (4.1.2) să fie exactă pentru orice polinom de grad mai mic sau egal cu $(2n-1)$, este necesar şi suficient ca ea să fie satisfăcută pentru *şirul fundamental* de funcții

$$1, x, x^2, \ldots, x^{2n-1}, \tag{4.1.3}$$

adică, să aibă loc

$$\int_{-1}^{1} x^k dx = \sum_{i=1}^{n} w_i x_i^k, \quad k = 0,1,\ldots,2n-1. \tag{4.1.4}$$

Într-adevăr, orice polinom de grad mai mic sau egal cu (*2n-1*) poate fi reprezentat sub forma unei combinații liniare a funcțiilor șirului fundamental (4.1.3):

$$Q(x) = \sum_{k=0}^{2n-1} a_k x^k.$$

Integrând ambii membri ai acestei relații între $-1$ și $1$ și ținând cont de formulele (4.1.4), se obține succesiv:

$$\int_{-1}^{1} Q(x)dx = \sum_{k=0}^{2n-1} a_k \int_{-1}^{1} x^k dx = \sum_{k=0}^{2n-1} a_k \sum_{i=1}^{n} w_i x_i^k = \sum_{i=1}^{n} w_i \sum_{k=0}^{2n-1} a_k x_i^k,$$

adică

$$\int_{-1}^{1} Q(x)dx = \sum_{i=1}^{n} w_i Q(x_i),$$

ceea ce demonstrează propoziția.

În principiu, pentru determinarea nodurilor $x_i$ și a ponderilor $w_i$ ale formulei de cuadratură (4.1.2), poate fi folosit sistemul neliniar (4.1.4), având ca termeni liberi integralele

$$\int_{-1}^{1} x^k dx = \begin{cases} 2/(k+1) & k \text{ par} \\ 0 & k \text{ impar.} \end{cases}$$

Dacă $n$ este mic, sistemul (4.1.4) poate fi rezolvat analitic. În schimb, pentru $n$ mare, rezolvarea analitică prezintă dificultăți deosebite. De aceea, vom utiliza o altă cale pentru determinarea nodurilor și ponderilor formulei de cuadratură (4.1.2).

În spațiul funcțiilor de pătrat integrabil pe intervalul [-1,1], notat $L_2[-1,1]$, polinoamele Legendre $P_n(x)$ ($n=0, 1, 2, \ldots$) joacă un rol cu totul deosebit, deoarece constituie un *sistem ortogonal complet*, sau o *bază*.

*Ortogonalitatea* polinoamelor Legendre se exprimă prin anularea produsului scalar dintre două polinoame de ordin diferit,

$$\int_{-1}^{1} P_m(x) P_n(x) dx = 0, \text{ dacă } m \neq n, \tag{4.1.5}$$

iar *completitudinea* lor revine la faptul că orice funcție $f \in L_2[-1,1]$ poate fi dezvoltată în mod unic într-o serie convergentă în raport cu ele.

În particular, orice polinom de ordinul $k$ poate fi scris ca o combinație liniară de polinoame Legendre de ordin mai mic sau egal cu $k$,

$$Q_k(x) = \sum_{m=0}^{k} c_m P_m(x) \tag{4.1.6}$$

Înmulțind scalar ambii membri ai acestei dezvoltări cu polinomul Legendre $P_n(x)$ de ordin $n>k$ și folosind proprietatea de ortogonalitate (4.1.5), obținem:

$$\int_{-1}^{1} Q_k(x) P_n(x) dx = \sum_{m=0}^{k} c_m \int_{-1}^{1} P_m(x) P_n(x) dx = 0, \quad k < n.$$

Considerând în particular $Q_k(x) \equiv x^k$, avem

$$\int_{-1}^{1} x^k P_n(x) dx = 0, \quad k < n. \tag{4.1.7}$$

Pe de altă parte, fiind de ordin mai mic sau egal cu $2n-1$, integrandul $x^k P_n(x)$ satisface formula de cuadratură (4.1.2),

$$\int_{-1}^{1} x^k P_n(x) dx = \sum_{i=1}^{n} w_i x_i^k P_n(x_i) \tag{4.1.8}$$

și, comparând (4.1.7) cu (4.1.8), rezultă sistemul

$$\sum_{i=1}^{n} w_i x_i^k P_n(x_i) = 0, \quad k = 0,1,\ldots,n-1. \tag{4.1.9}$$

Ecuațiile acestui sistem sunt satisfăcute simultan pentru valori arbitrare $w_i$ numai dacă

$$P_n(x_i) = 0, \quad i = 1,2,\ldots,n. \tag{4.1.10}$$

Prin urmare, pentru a obține precizie maximă în formulele de cuadratură (4.1.2), este suficient să considerăm ca puncte de integrare $x_i$ cele $n$ zerouri ale polinomului Legendre $P_n(x)$, care sunt reale și distincte și sunt situate în intervalul (-1,1).

Pentru a determina ponderile $w_i$ asociate zerourilor $x_i$, punem polinomul Legendre $P_n(x)$ sub forma

$$P_n(x) = c\prod_{i=1}^{n}(x - x_i) \tag{4.1.11}$$

și definim polinomul de grad $(n-1)$

$$Q_{n-1}^{(k)}(x) = \frac{P_n(x)}{x - x_k} = c\prod_{i\neq k}^{n}(x - x_i), \tag{4.1.12}$$

care se anulează pentru toate zerourile lui $P_n(x)$ cu excepția lui $x_k$. Pe de altă parte, derivata lui $P_n(x)$ este

$$P_n'(x) = c\sum_{j=1}^{n}\prod_{i\neq j}^{n}(x - x_i)$$

şi se observă cu uşurinţă că pentru $x = x_k$ are loc relaţia

$$Q_{n-1}^{(k)}(x_k) = P_n'(x_k), \tag{4.1.13}$$

deoarece în expresia lui $P_n'(x_k)$ se anulează toţi termenii care conţin factorul $(x - x_k)$.

Formula de cuadratură (4.1.2) rămâne exactă pentru $\left[Q_{n-1}^{(k)}(x_k)\right]^2$, care este un polinom de gradul *(2n-2)* şi se deduce succesiv:

$$\int_{-1}^{1}\left[Q_{n-1}^{(k)}(x)\right]^2 dx = \sum_{i=1}^{n} w_i\left[Q_{n-1}^{(k)}(x)\right]^2 = w_k\left[Q_{n-1}^{(k)}(x_k)\right]^2 = w_k\left[P_n'(x_k)\right]^2, \tag{4.1.14}$$

unde s-a ţinut cont că $Q_{n-1}^{(k)}(x)$ se anulează în zerourile lui $P_n(x)$ cu excepţia lui $x_k$. Pe de altă parte, utilizând (4.1.12) şi integrand prin părţi, rezultă:

$$\int_{-1}^{1}\left[Q_{n-1}^{(k)}(x)\right]^2 dx = -\frac{P_n^2(1)}{1-x_k} - \frac{P_n^2(-1)}{1+x_k} + 2\int_{-1}^{1} Q_{n-1}^{(k)}(x)P_n'(x)dx. \tag{4.1.15}$$

Având în vedere proprietatea $P_n^2(1) = P_n^2(-1) = 1$ şi faptul că $Q_{n-1}^{(k)}(x)P_n'(x)$ este un polinom de ordinul *(2n-2)*, pentru care este exactă formula de cuadratură (4.1.2), relaţiile (4.1.14) şi (4.1.15) conduc la

$$w_k\left[P_n'(x_k)\right]^2 = -\frac{2}{1-x_k^2} + 2\sum_{i=1}^{n} w_i Q_{n-1}^{(k)}(x_i)P_n'(x_i)$$

sau

$$w_k\left[P_n'(x_k)\right]^2 = -\frac{2}{1-x_k^2} + 2w_k Q_{n-1}^{(k)}(x_k)P_n'(x_k).$$

Ţinând cont de (4.1.13), obţinem în final pentru ponderile $w_i$:

$$w_i = \frac{2}{\left(1-x_i^2\right)\left[P_n'(x_i)\right]^2}, \quad i = 1,2,\ldots,n. \tag{4.1.16}$$

Formula de cuadratură (4.1.2), cu cele *n* puncte de integrare egale cu zerourile polinomului Legendre $P_n(x)$ şi ponderile asociate date de relaţia (4.1.16), se numeşte *formula de cuadratură Gauss-Legendre* şi este exactă pentru polinoame până la ordinul *(2n-1)*. Numele particular al acestei formule este menit să o deosebească de alte scheme din familia cuadraturilor gaussiene, construite pe baza aceloraşi principii, însă pentru alte intervale de integrare şi utilizând proprietăţile altor sisteme de polinoame ortogonale (Cebîşev, Laguerre, Hermite etc.).

În cazul unui interval de integrare [a,b] oarecare (cu a şi b finite), formula de cuadratură Gauss-Legendre are forma

$$\int_a^b f(\xi)d\xi = \frac{b-a}{2}\sum_{i=1}^{n} w_i f(\xi_i), \tag{4.1.17}$$

unde punctele de integrare sunt definite în raport cu abscisele $x_i$ din intervalul standard [-1,1] prin

$$\xi_i = \frac{b-a}{2}x_i + \frac{b+a}{2}. \tag{4.1.18}$$

Se poate demonstra că restul corespunzător are expresia

$$R_n = \frac{(b-a)^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3} f^{(2n)}(\xi), \quad \xi \in [a,b]. \tag{4.1.19}$$

În tabelul de mai jos sunt cuprinse câteva seturi de abscise şi ponderi pentru cuadratura Gauss-Legendre. Trebuie notat faptul că pentru orice $n$ nodurile de integrare $x_i$ sunt distribuite simetric faţă de origine în interiorul intervalului (-1,1), iar ponderile nodurilor simetrice sunt egale. În cazul valorilor $n$ impare, originea însăşi este punctul central de integrare ($x_{(n/2)+1} = 0$).

| $n$ | $i$ | $x_i$ | $w_i$ |
|---|---|---|---|
| 1 | 1 | 0 | 2 |
| 2 | 1, 2 | ±0,57735027 | 1 |
| 3 | 1, 3 | ±0,77459667 | 0,55555556 |
| | 2 | 0 | 0,88888889 |
| 4 | 1, 4 | ±0,86113631 | 0,34785484 |
| | 2, 3 | ±0,33998104 | 0,65214516 |
| 5 | 1, 5 | ±0,90617985 | 0,23692688 |
| | 2, 4 | ±0,53846931 | 0,47862868 |
| | 3 | 0 | 0,56888889 |

Un prim exemplu de implementare este programul de mai jos în care funcţia `QGaussLeg6` este folosită pentru a calcula $\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \arctan y dy$, utilizând formula Gauss-Legendre cu 6 puncte de integrare.

```
#include <stdio.h>
#include <math.h>
#define Pi 3.14159265358979323846264338327 95

float func(float y)
{
        return atan(y);
}

float QGaussLeg6(float Func(float), float a, float b)
/* Calculeaza integrala functiei Func pe intervalul [a,b] utilizand cuadratura Gauss-Legendre cu 6 puncte de
integrare */
{
        const float x[]={0., 0.2386191861, 0.6612093865, 0.9324695142};
```

```
        const float w[]={0., 0.4679139346, 0.3607615730, 0.1713244924};
        float f, s, xc, xi;
        int i;

        f = 0.5*(b – a); xc = 0.5*(b+a);
        s = 0.;

        for (i = 1; i <= 3; i++)
        {
                xi = f*x[i];
                s += w[i]*(Func(xc+xi)+Func(xc-xi));
        }
        return s *= f;
}

void main()
{
        printf("Integrala functiei arctan y pe intervalul [-pi/2 3pi/2] are valoarea: %.8f\n",
QGaussLeg6(func,-Pi/2, 3*Pi/2));
}
```

Datorită simetriei în raport cu intervalul (-1,1), este suficient să se declare numai nodurile şi ponderile pentru semi-intervalul pozitiv (primele componente, x[0] şi w[0], nu sunt folosite). Corespunzător, abscisele din intervalul [a,b] sunt generate simetric faţă de centrul *xc* al intervalului, ponderile nodurilor simetrice fiind egale.

O utilizare mai elegantă a cuadraturilor gaussiene presupune posibilitatea determinării prin calcul direct a absciselor şi ponderilor pentru formule cu număr arbitrar de termeni. Pornind de la relaţia de mărginire a zerourilor polinomului Legendre $P_n(x)$,

$$\cos\left(\frac{2i-1}{2n+1}\pi\right) \le x_i \le \cos\left(\frac{2i}{2n+1}\pi\right),$$

un procedeu simplu se bazează pe calculul iterativ al fiecărui zero cu ajutorul metodei Newton,

$$x_i^{(k+1)} = x_i^{(k)} - \frac{P_n\left(x_i^{(k)}\right)}{P_n'\left(x_i^{(k)}\right)}, \quad k = 0,1,2,\ldots, \tag{4.1.20}$$

pornind de la aproximaţia iniţială

$$x_i^{(0)} = \cos\left(\frac{i-0,5}{n+0,5}\pi\right). \tag{4.1.21}$$

Odată determinate zerourile $x_i$, ponderile $w_i$ rezultă imediat din formula (4.1.16).

Rutina `XGaussLeg` listată mai jos returnează, prin tablourile `x` şi `w`, *n* abscise şi ponderi pentru cuadratura Gauss-Legendre pe intervalul [a,b]. Această rutină apelează la rândul ei funcţia `Legendre` care evaluează polinomul Legendre de ordin *n* şi derivata sa în punctul *x* pe baza formulelor de recurenţă:

$$P_i(x) = \begin{cases} 1, & i = 0 \\ x, & i = 1 \\ [(2i-1)xP_{i-1}(x) - (i-1)P_{i-2}(x)]/i, & i = 2,3,\ldots,n \end{cases} \tag{4.1.22}$$

$$P_n'(x) = \begin{cases} n(n+1)P_n(x)/(2x), & x = \pm 1 \\ n[xP_n(x) - P_{n-1}(x)]/(x^2-1), & x \ne \pm 1 \end{cases} \tag{4.1.23}$$

```
float Legendre(int n, float x, float *d)
/* Evalueaza polinomul Legendre de ordinul n in punctul x, returnand derivata in *d */
{
        float f, fm1, fm2;
        int i;

        if (n == 0) {
                f = 1.0; *d = 0.0;
        } else {
                f = x; fm1 = 1.0;
                for (i=2; i<=n; i++) {
                        fm2 = fm1; fm1 = f;
                        f = ((2*i-1)*x*fm1 - (i-1)*fm2)/i;
                }
                *d = (x*x - 1.0) ? n*(x*f - fm1)/(x*x - 1.0) : 0.5*n*(n+1)*f/x;
        }
        return f;
}

void XGaussLeg(float a, float b, float x[], float w[], int n)
/* Calculeaza abscise si ponderi pentru formula de cuadratura Gauss-Legendre
a,b - limitele domeniului de integrare
w[] - tabloul ponderilor
x[] - tabloul absciselor
n - numarul punctelor de integrare */
{
        const float pi = acos (-1.0);           //constanta pi
        const float eps = 1e-6;                 //criteriu relativ de precizie
        float d, f, xc, z;
        int i;

        for (i = 1; i <=(n/2); i++) {
                z = cos(pi*(i-0.5)/(n+0.5));            //aproximatie initiala pentru zero
                do {                                    //rafinare cu metoda Newton
                        f = Legendre(n,z,&d) / d;
                        z -= f;
                } while (fabs(f) > eps*fabs(z));
                x[i] = -z; x[n-i+1] = z;                //zerouri simetrice
                w[i] = w[n-i+1] = 2.0/((1.0 - z*z)*d*d);        //ponderi
        }

        if ((n/2)*2 != n) {                             //numar de puncte impar
                Legendre(n,0.0,&d);
                x[n/2+1] = 0.0;
                w[n/2+1] = 2.0/(d*d);
        }

        f = 0.5*(b-a); xc = 0.5*(b+a);                  //scalare la intervalul [a,b]
        for (i=1;i<=n;i++) {
                x[i] = f*x[i] + xc;
                w[i] = f*w[i];
        }
}
```

Prin procedeul descris mai sus sunt determinate efectiv numai cele *n/2* zerouri şi ponderi corespunzătoare semi-intervalului (0,1), restul fiind completate în tablourile x şi w prin simetrizare. Cazul numărului de puncte *n* impar este tratat separat, funcţia Legendre fiind apelată numai pentru calculul derivatei d. În finalul rutinei, abscisele şi ponderile corespunzătoare intervalului standard [-1,1] sunt scalate pentru intervalul [a,b].

Programul de mai jos exemplifică utilizarea rutinei `XGaussLeg` pentru a calcula $\int_{-\frac{\pi}{2}}^{\frac{3\pi}{2}} \arctan y dy$, utilizând formula Gauss-Legendre cu $n$ puncte de integrare. Programul afişează valoarea integralei calculată atât numeric cât şi prin înlocuire în expresia analitică rezultată prin calcul matematic.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define Pi 3.14159265358979323846264338327 95

float func(float y)
{
        return atan(y);
}

float analitic(float u)
{
        return u*atan(u) – 1.0/(2.0*log(1.0+u*u));
}

float QGaussLeg(float Func(float), float a, float b, int n)
/* Calculeaza integrala functiei Func pe intervalul [a,b] utilizand cuadratura Gauss-Legendre cu n puncte de
integrare */
{
        float s, *x, *w;
        int i;

        x = (float*) malloc(n*sizeof(float)); x--;
        w = (float*) malloc(n*sizeof(float)); w--;
        XGaussLeg(a,b,x,w,n);

        s = 0.0;
        for (i = 1; i <= n; i++) s += w[i]*Func(x[i]);

        free(x+1); free(w+1);
        return s;
}

void main()
{
        printf("Integrala functiei arctan y pe intervalul [-Pi/2, 3Pi/2] are valoarea:\n\t calcul numeric: %.8f\n\t
calcul matematic: %.8f\n", QGaussLeg(func,-Pi/2, 3*Pi/2,5), analitic(3*Pi/2) – analitic(-Pi/2));
}
```

### 4.1.2. Desfăşurarea laboratorului

Se va utiliza formula Gauss-Legendre cu 5, 10 apoi 25 puncte de integrare pentru a calcula valoarea integralelor de la punctele a)...e). Se va alcătui un tabel comparativ cu diferenţele procentuale între valoarea exactă (calcul matematic) şi cea aproximativă (calcul numeric) a integralelor în fiecare dintre cele trei situaţii. Procentele se determină utilizând formula:

$$procent = \left| \frac{exact - aprox}{exact} \right| \times 100 .$$

a) $\int_2^7 \frac{1}{x} dx = \ln|x| \Big|_2^7$

b) $\int_{-1}^{1} x^{10} dx = \frac{x^{11}}{11}\Big|_{-1}^{1}$

c) $\int_{-5}^{0} e^{x} dx = e^{x}\Big|_{-5}^{0}$

d) $\int_{0}^{\pi} \sin x dx = -\cos x\Big|_{0}^{\pi}$

e) $\int_{-\frac{\pi}{2}}^{\frac{3\pi}{2}} \arctan x dx = x \arctan x - \frac{1}{2\ln(1+x^2)}\Big|_{-\frac{\pi}{2}}^{\frac{3\pi}{2}}$