Lucrarea 1

GRAFURI DE PROCEDURĂ

CALCULUL VALORII UNUI POLINOM FOLOSIND UN NUMAR MINIM DE OPERATII DE INMULTIRE

SCOPUL LUCRĂRII

În prima parte a lucrării sunt aduse în discuţie grafurile de procedură, ca metodă de calcul a marginii erorii relative pentru diferite expresii şi aplicarea acestei metode în câteva situaţii practice concrete.

In a doua parte a lucrarii se propune calculul valorii unui polinom folosind un numar minim de operatii de inmultire.

1.1. GRAFURI DE PROCEDURĂ

Aceste grafuri sunt *binare*, adică într-un nod intră cel mult două mărimi şi iese o singură mărime. Pentru fiecare operaţie aritmetică marca grafului se calculează diferit.

Operaţia de adunare.

Se dau valorile erorilor relative ε_x şi ε_y ale mărimilor care se însumează şi eroarea de rotunjire ε_r.

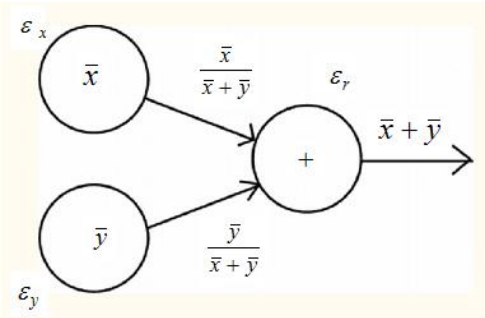

Fig.1.1. - Graful operaţiei de adunare

Eroarea la ieşirea grafului este:

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

Operaţia de scădere

Eroarea la ieşirea grafului este dată în urmatoarea formula, iar graful este prezentat în fig. 1.2.

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

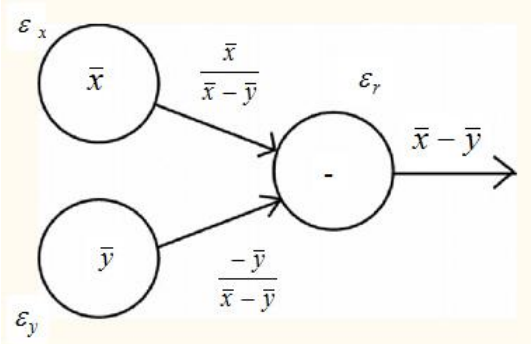

Fig.1.2. - Graful operaţiei de scădere

Operaţia de înmulţire

$$\varepsilon_{xy} = \varepsilon_x + \varepsilon_y + \varepsilon_r$$

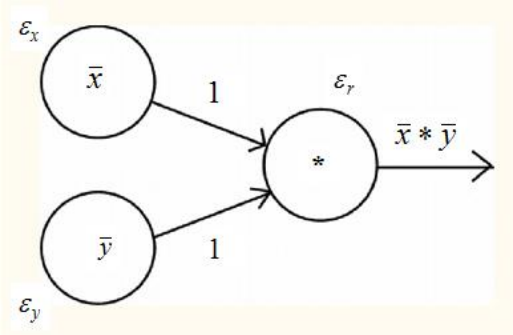

Fig.1.3. - Graful operaţiei de înmulţire

Operaţia de împărţire. Eroarea la ieşirea grafului este:

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

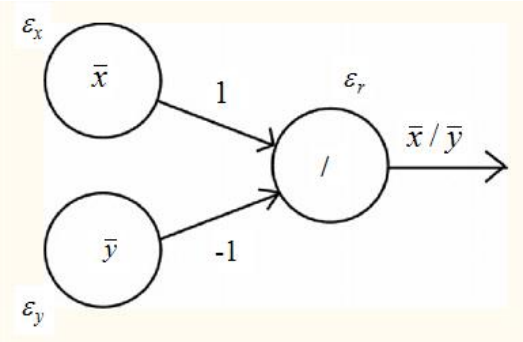

Fig.1.4. - Graful operaţiei de înmulţire

Operaţia de extragere a rădăcinii pătrate are marca grafului constantă egală cu ½ şi este prezentată în figura 1.5.

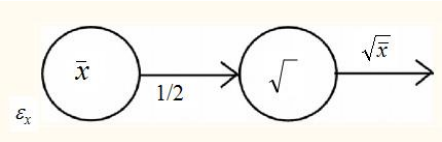

Fig.1.5. - Graful operaţiei de extragere de rădăcină pătrată

Eroarea mărimii la ieşirea grafului este:

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

Pentru uşurinţa calculului mărcilor grafului se ţine seama de modul de calcul prezentat anterior pentru fiecare operaţie aritmetică. Cunoscând erorile relative ale mărimilor care intră într-o expresie în care avem operaţiile de bază ale aritmeticii, putem să determinăm eroarea finală a expresiei cu ajutorul grafurilor de procedură.

1.2 PROBLEME DE LABORATOR

1. Fie *a* un număr pozitiv rotunjit în mod obişnuit, iar 3 nu are eroare. Se dau expresiile:

   $u=a+a+a$, şi

$v=3a$

Să se arate, cu ajutorul grafurilor de procedură, că marginea erorii relative a lui u este mai mare ca marginea erorii relative a lui *v*.

2. Se consideră expresiile $u = a^2 + 2a + 1$ şi $v = (a+1)^2$ unde a este un număr pozitiv rotunjit în mod obişnuit şi presupunem că 1 şi 2 nu au erori. Să se arate că pentru a foarte mare marginile erorii relative a lui *u* şi *v* devin aproximativ egale iar pentru a foarte mic marginea erorii relative a lui u devine aproximativ de trei 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 şi pentru cazul când a este exact.
3. Se consideră un circuit R, L, C alimentat de la o sursă U. Ştiind că frecvenţa sursei are variaţia ε_f, iar R, L, C au variaţiile $\varepsilon_R, \varepsilon_L, \varepsilon_c$, să se traseze graful impedanţei circuitului şi să se determine o margine a erorii relative a ei.
4. Se consideră circuitul din figura 1.6 in care $R_1 = 4.7k\Omega$ şi toleranţa $\varepsilon_1 = 10\%$, $R_2 = 6.8k\Omega$ şi toleranţa $\varepsilon_2 = 5\%$, $R_3 = 7{,}5k\Omega$ cu toleranţa $\varepsilon_3 = 10\%$ şi R4 = 10kΩ cu toleranţa ε_4 = 10%. Să se calculeze erorile relative a rezistenţelor. Să se deducă formula de calcul a rezistenţei echivalente a circuitului între punctele A şi B. Scrieţi toate valorile în virgulă mobilă şi considerând că toate erorile sunt mai mici decât eroarea relativă a rotunjirii simetrice determinaţi eroarea relativă maximă a rezistenţei echivalente. Consideraţi numărul de cifre semnificative a valorilor *t* = 2.

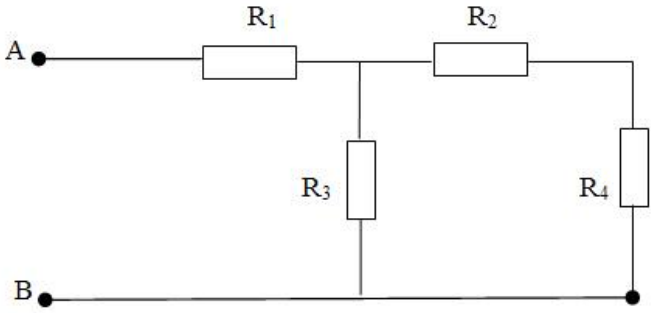

Fig. 1.4 - Circuitul pentru problema 4

5. Se consideră amplificatorul operaţional din figura 1.7. Considerând rezistenţele $R_1 = 100k\Omega$ cu toleranţa de $\varepsilon_1 = 10\%$ şi $R_2 = 10M\Omega$ cu toleranţa de $\varepsilon_2 = 10\%$ să se determine eroarea maximă a amplificării. Se calculează erorile relative a rezistenţelor, se fac calculele în virgulă mobilă si considerăm numărul de cifre semnificative *t = 2*.

$$A = -\frac{R_2}{R_1}$$

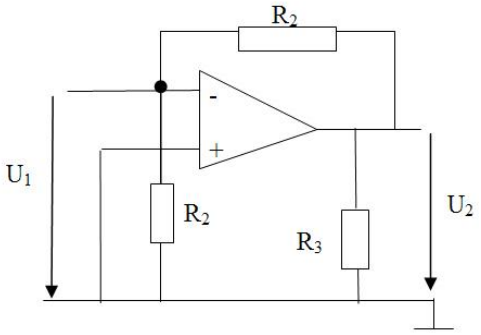

Fig. 1.5 - Circuitul pentru problema59

6. Folosirea simultană a tipurilor reale şi întregi

Vom alege un scurt program, ce presupune utilizarea ambelor tipuri de date (reale şi întregi) pentru a rezolva problema propusă.

Orice programator va trebui să folosească cu atenţie tipurile mixte, pentru că este vorba despre spaţii de memorie în principiu diferite, dar mai ales de două mulţimi de numere, în speţă **Z** şi **R**, care nu au o echivalenţă imediată între valorile lor.

Va trebui utilizat şi mecanismul *conversiilor explicite*, pentru a garanta păstrarea informaţiei, şi a evita pierderea acesteia - datorată de exemplu micşorării spaţiului de memorie prin trecerea de la real la întreg.

Pentru programul următor, compilatorul avertizează programatorul asupra unor conversii între tipuri de date incompatibile. Aceste avertismente se referă la situaţiile în care nu apare operatorul de conversie explicită. Pentru celelalte situaţii totul este în regulă, pentru că odată ce întâlneşte conversia explicită compilatorul presupune că programatorul ştie ce face. Avertismentele ţin şi de modul în care este configurat compilatorul, având în vedere că se pot impune opţiuni în linia de comandă în momentul lansării sale în execuţie.

```
// tipuri mixte de date
#include<iostream>
using namespace std;

int main()
{
   int nr ;            // implicit cu semn
   unsigned int nr1;   // fara semn
   float rez = -2.1;

// folosirea mixta a tipurilor de date - fara conversie implicita
// Numarul intreg este cu semn
   nr = rez;
   cout << "\n Fara conversie implicita -> numarul intreg este "
                                                    << nr;

// folosirea mixta a tipurilor de date - cu conversie implicita
// Numarul intreg este cu semn
   nr = (int)rez;
   cout << "\n Cu conversie implicita -> numarul intreg este "
                                                    << nr;

// =============
// folosirea mixta a tipurilor de date - cu conversie implicita
// Numarul intreg este fara semn
   nr1 = rez;
   cout << "\n Fara conversie implicita -> numarul intreg este "
                                                 << nr1;

// folosirea mixta a tipurilor de date - cu conversie implicita
// Numarul intreg este fara semn
   nr1 = (int)rez;
   cout << "\n Cu conversie implicita -> numarul intreg este "
                                                 << nr1;
// final program
   cin >> nr;
   return 0;
}
```

Compilatorul va avertiza programatorul prin mesajele:

```
Line 14 [Warning] assignment to `int' from `float'
Line 25 [Warning] assignment to `unsigned int' from `float'
```

Este vorba despre liniile 14 şi 25 din program, în care nu apar operatorii de conversie explicită, şi se impune unui număr real să fie salvat într-un număr întreg. Astfel de situaţii pot uşor să fie trecute cu vederea, ceea ce are consecinţe importante în momentul rulării programului.

Pentru situaţia în care numărului real i se impune să fie reţinut în spaţiul asociat unui număr întreg scurt (2B), fără a se face apel la conversia explicită, s-ar putea da următoarea interpretare operaţiei: se pleacă de la bitul 0, se numără 16b (adică 2B) şi ceea ce rezultă este salvat în numărul întreg. Însă nu este vorba despre o trunchiere fizică a biţilor mantisei! Rezultatul oferit de compilator în absenţa conversiei explicite este într-adevăr eronat, dar pentru situaţia în care numărul întreg este fără semn. Pentru celălalt caz, deşi rezultatele sunt similare şi cu şi fără operatorul de conversie explicită, este indicat ca de fiecare dată să se folosească operaţia de conversie, pentru că nu se ştie ce generaţie de compilator este folosit.

CALCULUL VALORII SI A DERIVATEI UNUI POLINOM FOLOSIND UN NUMAR MINIM DE OPERATII DE INMULTIRE

$$P_n(x) = (...((a_n x + a_{n-1})x + a_{n-2})x + ... + a_1)x + a_0$$

1.3. Algoritm. Valoarea Polinomului

```
real ValPol ( intreg grad, real coef[], real point)
{                    //declararea si definirea variabilelor locale
intreg i;
real b;
                     //calculul valorii polinomului
b=coef[grad];
pentru i=grad ÷ 1
   b=b*point + coef[grad-1];
returneaza b;
}
```

1.4. Algoritm. Valoarea Derivatei Polinomului

```
real DerPol ( intreg grad, real coef[], real point)
{                    //declararea si definirea variabilelor locale
intreg i;
real der;
                     //calculul valorii derivatei polinomului
der=grad*coef[grad];
pentru i=grad-1 ÷ 1
   b=b*point + i*coef[i];
returneaza b;
}
```

Lucrarea 2

REZOLVAREA NUMERICĂ A ECUAŢIILOR ALGEBRICE

2.1. METODE DIRECTE PENTRU REZOLVAREA ECUAŢIILOR

2.1.1. Algoritm. METODA BISECŢIEI pentru ecuaţii transcendente

```
Real Bis_Functie
(
   real ls,                    // limita stânga a intervalului de calcul
   real ld,                    // limita dreapta a intervalului de calcul
   real er                     // eroarea de calcul (precizia impusa)
   int *k                      // contor - pasii necesari pt gasirea   radacinii
)
{
                               // declaraţiile variabilelor locale:
   real xm;                    // mijlocul intervalului
   real rad;                   // radacina


xm=(ls+ld)/2;                  // conditia de intrare in ciclul 'while'
cat timp( (fabs(ld-ls) > er) SI (f(xm) != 0) )
{  *k=(*k)+1;
   xm=(ls+ld)/2;
   daca ( f(xm) * f(ls)<0 ) ld = xm;
   altfel ls = xm;             // se stabilesc noile limite de interval
}
rad = xm;                      // la fel de bine radacina este fie a fie b
return rad;                    // intoarce radacina la numele functiei
}
```

2.2. METODE INDIRECTE PENTRU REZOLVAREA ECUAŢIILOR

2.2.1. Algoritm. METODA APROXIMARILOR SUCCESIVE

```
Real  Aprox_Succesive
(
   real ls,                    // limita stânga a intervalului de calcul
   real ld,                    // limita dreapta a intervalului de calcul
   real x0,                    // punctul de start - aproximaţia iniţiala
   real er,                    // eroarea de calcul
   int *k                      // contor - pasii necesari pt gasirea   radacinii

)
{
                               // declaraţiile variabilelor locale:
real pc;                       // punctul curent de derivare
real xn, xn_1;                 // aproximatiile curentă şi anterioara ale radacinii
real der;                      // retine valoarea derivatei lui fi(x)
real h;                        // pasul de derivare intre doua abscise

pc = ls;                       //verifica dc fi(x) are derivata subunitara pe  tot intervalul [ls,ld]
repeta
 { der = (fi(pc+h) - fi(pc)) / h;
   daca (fabs(der) >=1) {
                        tipareste mesaj de „eroare";
```

```
                                incheie fortat programul
                                 }
      pc=pc+h;                      // trec la urmatoarea abscisa
    } cat timp(pc<=ld);

   xn = x0;
   repeta
       {
      *k=(*k)+1;                    // contor - pasii necesari pt gasirea   radacinii
      xn_1 = xn;
      xn = fi(xn_1);
       }
   cat timp(fabs(xn-xn_1) >= er);

   rad = xn;                        // valoarea radacinii este valoarea lui 'xn'

   return rad;                      // intoarce radacina la numele functiei
   }
```

Obs.: Este foarte important sa calculati una sau mai multe functii fi(x), pentru a verifica apoi daca indeplinesc conditia de contractie, valoarea absoluta a derivatei sa fie subunitara.

Exemplu:

Daca f(x)=exp(3*x)-2*x*x-1.618 functiile fi(x) pot avea una din expresiile:

1) fi(x)=sqrt((exp(3*x)-1.618)/2);
2) fi(x)=ln(2*x*x+1.618)/3;
3) fi(x)= exp(3*x)-2*x*x-1.618+x;

2.3. METODA TANGENTEI sau NEWTON RAPHSON

2.3.1. Algoritmul. Newton-Raphson pentru ecuaţii transcendente

```
real  NewtonRaphson
(
   real x0,                     // valoarea initiala necesara algoritmului
   real er,                     // eroare de aproximare  radacinii
   real h,                      // valoarea pasului de derivare
   int nMax                     // numarul maxim de iteratii impus
   intreg *k                    // calculeaza numarul de pasi din care s-a gasit solutia
)
{                               // declaratiile variabilelor locale:
   real xn, xn_1;               // valoarea aproximată, curent si anterior a radacinii
   real der;                    // valoarea derivatei
   xn=x0; *k=0;
   repetă
   {
      xn_1=xn;
      der=( f(xn_1+h)-f(xn_1) )/h;
      dacă (der == 0) {tiparste „eroare”; exit(0)}        //ies fortat din program
      xn = xn_1 - f(xn_1)/der;
      nMax = nMax-1;
      (*k)++;
   } cat timp((fabs(xn-xn_1)>=er) SI (nMax>0));
   dacă (nMax= =0) {tiparste „eroare”; exit(0)};          //ies fortat din program
   return xn;                   //  s-a returnat valoarea solutiei in caz de succes
}
```

Lucrarea 3

REZOLVAREA NUMERICĂ A SISTEMELOR LINIARE

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

3.1.METODE DIRECTE

Aceste metode sunt caracterizate prin aceea că pot oferi soluţia exactă a sistemului de rezolvat, dacă sunt îndeplinite condiţiile de existenţă.

3.1.1. METODA LUI GAUSS DE ELIMINARE (ALGORITMUL)

```
real Gauss
(
   întreg n,                    // ordinul sistemului
   real A[][N],                 // matricea sistemului
   real B[],                    // vectorul termenilor liberi
   real X[]                     // vectorul soluţiilor
)
{                               // declaraţiile şi definiţiile variabilelor locale
       întreg i;                // indicele liniei
       întreg k;                // indice suplimentar al liniei
       întreg j;                // indice coloană
       real m;                  // multiplicator
       real d;                  // determinantul sistemului

   pentru i = 1,n-1 {           // for 1
     pentru k = i+1,n  {
         m = A[k][i]/A[i][i];
         pentru j = i,n  A[k][j] = A[k][j] - m*A[i][j];
         B[k] = B[k] - m*B[i];
                        }
   }                            // for 1

   d=1;                         // iniţializarea cu 1 a valoarii determinantului
   dacă (A[n][n] = = 0)  returnează 0;
   pentru i =1,n  d = d*A[i][i];
                                // Rezolvă sistem superior triunghiular
   pentru i = n,1 {
         x[i] = B[i];
         pentru j = n,i+1 x[i] = x[i] - A[i][j]*x[j];
         x[i] = x[i] / A[i][i];
         }
   returnează d;                // returneaza la numele functiei valoarea determinantului
}
```

3.2. METODE INDIRECTE

3.2.1. METODA LUI JACOBI

Fie sistemul liniar (3.1). Se explicitează fiecare necunoscută din sistem de pe diagonala principală funcţie de celelalte necunoscute ale sistemului. Se alege o soluţie a sistemului oarecare $x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)}$, ce va reprezenta soluţia de start, după care se continuă iteraţiile, până când două soluţii succesive diferă cu mai puţin de o eroare impusă (precizia impusă).

3.2.2 Algoritmul. Metoda lui Jacobi

```
întreg Jacobi
(
   întreg n,                  // ordinul sistemului;
   real A[][N],               // matricea sistemului;
   real B[],                  // vectorul termenilor liberi;
   real X[]                   // vectorul de start la intrare, soluţia la ieşire
)
{
                              // declaraţiile şi definiţiile variabilelor locale
întreg i, j;                  // indici;
real sum;
real Xn[N], Xn_1[N];          // soluţiile la pasul curent şi precedent
întreg sem;

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

                              // se salveaza valorile conţinute de vectorul de start X;
   pentru i= 1,n   Xn[i] = X[i];
   execută
   {
      sem = 0;
      pentru i= 1,n    Xn_1[i] = Xn[i];
      pentru i= 1,n
      {
         Xn[i] = B[i];
         pentru j= 1,n
               dacă (j != i)  Xn[i] = Xn[i] - A[i][j] * Xn_1[j];
         Xn[i] = Xn[i]/A[i][i];
      }
      pentru i= 1,n
         dacă (fabs(Xn[i]-Xn_1[i]) > eps )  sem = 1;
   } cât timp (sem = = 1);
   pentru i= 1,n    X[i] = Xn[i];      // se salveaza valorile solutiei in vectorul de X
   returnează 0;}
```

Lucrarea 4

DERIVAREA NUMERICĂ
INTEGRAREA NUMERICĂ
INTERPOLAREA (partea 1)

4.1. DERIVAREA NUMERICĂ.

4.1.1. DERIVATA NUMERICĂ PRIN DOUĂ PUNCTE

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

4.1.2.DERIVATA NUMERICĂ PRIN TREI PUNCTE

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

Pentru $h_1 = h_2 = h/2$ se obţine formula derivatei numerice *prin două puncte*.

4.1.3. DERIVATA NUMERICĂ PRIN CINCI PUNCTE

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

4.2. INTEGRAREA NUMERICĂ

4.2.1. METODA LUI RICHARDSON

Această metodă dă o precizie mai bună de calcul a integralei numerice decât metoda trapezului şi s-a obţinut prin modificarea metodei trapezului. Este o metodă în care apare *dubla-divizare* a intervalului de integrare.

Se pleacă de la eroarea de trunchiere a metodei trapezului, care pentru o diviziune de lăţime *h* este:

$$e_{T,h} = Ch^2, \quad h = (b-a)/n,$$

Pentru cealaltă diviziune, $k = (b-a)/m$, se obţine eroarea de trunchiere:

$$e_{T,k} = Ck^2$$

Pentru fiecare diviziune se poate scrie:

$$I = I_h + e_{T,h}$$

respectiv

$$I = I_k + e_{T,k}$$

Eliminând constanta *C* între relaţiile de mai sus, se obţine expresia:

$$I = I_h + \frac{I_h - I_k}{\left(\frac{k}{h}\right)^2 - 1}$$

expresie ce poartă denumirea de *formula lui Richardson*.

4.2.1.1. Metoda lui Richardson (algoritm)

```
real I_Richardson
(
   real ls,                    // limita stângă a intervalului de integrare
   real ld,                    // limita dreaptă a intervalului de integrare
   întreg n,                   // numărul de subintervale_1
   întreg m                    // numărul de subintervale_2
)
```

```
{
                                    // declararea şi definirea variabilelor locale
real h;                             // valoarea lungimii unui subinterval divizat  in n sub-diviziuni
real k;                             // valoarea lungimii unui subinterval divizat  im m sub-diviziuni
real sumh;                          // valoarea integralei cu diviziunea h
real sumk;                          // valoarea integralei cu diviziunea k
real sum;                           // valoarea integralei
întreg  i;                          // indice întreg pentru ciclul 'for()'

                                    // instrucţiunile de calcul
```

$h=\frac{ld-ls}{n}$; $k=\frac{ld-ls}{m}$;

```
    Sumh=(f(ls)+f(ld))/2*h;
    Sumk=(f(ls)+f(ld))/2*k;

pentru i= 1 ÷ n-1
        sumh = sumh + h*f(ls + i*h);
pentru i= 1 ÷ m-1
        sumk = sumk + k*f(ls + i*k);
sum=sumh + (sumh-sumk)/( (k/h)*(k/h)-1 );
returnează sum;
}
```

Această metodă determină punctele de divizare ale intervalului de integrare astfel ca eroarea de calcul a integralei să fie minimă.

4.2.2. METODA CUADRATURII GAUSSIENE CU DOUĂ PUNCTE (*tema*)

Această metodă reduce orice interval de integrare [a,b] la intervalul [-1,1] cu ajutorul formulei de *substituţie*:

$$y=\frac{2x-(b+a)}{b-a}$$

Pentru $x=1$ rezultă $y=-1$, iar pentru $x=b$ rezultă $y=1$. Din relaţia (5.11) se poate scrie:

$$x=\frac{1}{2}(b-a)\,y+\frac{1}{2}(b+a)$$

şi de aici:

$$dx=\frac{1}{2}(b-a)\,dy$$

Ca urmare, integrala iniţială în forma $I=\int_a^b f(x)\,dx$ se transformă în integrala:

$$I=\int_{-1}^{+1} f\left[\frac{1}{2}(b-a)y+\frac{1}{2}(b+a)\right]\frac{1}{2}(b-a)dy=\int_{-1}^{+1}\mathrm{fi}(y)dy$$

în care noua variabilă este y.

Integrala de mai sus este în final aproximată printr-o sumă, aşa cum s-a făcut de fiecare dată (din punct de vedere matematic integrala devine o sumă). Mai exact:

$$\int_{-1}^{+1}\mathrm{fi}(y)dy=\sum_{i=1}^{n}k_i\cdot\mathrm{fi}(y_i)$$

Precizia metodei este cea mai bună între metodele de integrare de *tip cuadratură* (pentru funcţii de o variabilă). Această putere de calcul este de fapt justificată, din moment ce metoda foloseşte în rezolvare *polinoamele Legendre*. Echivalentul matematic este acela de aproximare a funcţiei concrete de integrat printr-un polinom Legendre echivalent, pentru un anumit grad dorit de către utilizator. Cu cât gradul preluat în calcule este mai mare, cu atât precizia metodei evident creşte.

Polinoamele Legendre sunt definite pe intervalul [-1, 1]. De aceea se impune schimbarea limitelor de integrare iniţiale. Pe acest interval polinoamele prezintă un număr de rădăcini reale, distincte (gradul de multiplicitate al fiecărei soluţii este 1). Din cauză că aceste rădăcini nu sunt egal depărtate între ele, şi nici faţă de capetele intervalului de integrat, această metodă este clasificată drept metodă *cu divizare variabilă*.

Această precizie de calcul se poate impune în mod direct, prin stabilirea gradului polinomului *Legendre* dorit în calcul.

Există valori tabelate ale rădăcinilor acestor polinoame (notate aici cu Y_i), ca şi a ponderilor (K_i), valori ce intervin în relaţia de calcul de mai sus.

Pentru **N=3** aceste valori sunt: k_1=5/9, k_2=8/9, k_3=5/9 si
y_1=0.7745, y_2=0, y_3= -0.7745

4.2.3. CALCULUL NUMERIC AL INTEGRALELOR DUBLE

Pentru simplitate vom considera domeniul de integrare al funcţiei de două variabile un dreptunghi

$$\iint_D f(x,y)\,dxdy=\int_a^b\int_c^d f(x,y)\,dxdy$$

reprezintă integrala dublă dintr-o funcţie de două variabile. Pentru calculul valorii acestei integrale vom utiliza *formula de cubatură a trapezului* în continuare.

4.2.3.1. FORMULA DE CUBATURĂ A TRAPEZULUI

Intervalele de integrat [a,b] şi [c,d] se împart în subintervale de lungimi egale

$$h=\frac{b-a}{n} \quad \text{şi, respectiv} \quad k=\frac{d-c}{m}$$

şi se consideră dreptunghiul cu vârfurile $[x_i, y_i]$, $[x_{i+1}, y_i]$, $[x_{i+1}, y_{i+1}]$, $[x_i, y_{i+1}]$

$$I=\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} I_{ij}=\frac{kh}{4}\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}\left[f(x_i,y_j)+f(x_i,y_{j+1})+f(x_{i+1},y_j)+f(x_{i+1},y_{j+1})\right]$$

expresie cunoscută sub numele de *formula de **cubatură** a trapezului*. Unde $x_i=a+i*h$, $y_j=c+j*k$;

4.2.3.2. Algoritmul. Metoda cubaturii trapezului

```
real I_Cubatura_Trapez
(
   real a,              // limita stângă a intervalului de integrare  pe axa Ox
   real b               // limita dreaptă a intervalului de integrare pe axa Ox
   real c,              // limita stângă a intervalului de integrare  pe axa Oy
   real d,              // limita dreaptă a intervalului de integrare  pe axa Oy
   întreg n,            // numărul de subintervale pe axa Ox
   întreg m             // numărul de subintervale pe axa Oy
)

{
                   // declararea şi definirea variabilelor locale
întreg i, j;
real h;                 // valoarea lungimii unui subinterval diviziunea n
real k;                 // valoarea lungimii unui subinterval diviziunea m
real sum;               // valoarea integralei
```

```
                    // instrucţiunile de calcul ale metodei
```

$h=\frac{b-a}{n}$; $k=\frac{d-c}{m}$;

```
sum = 0;
pentru i= 0 ÷  n-1
  pentru j= 0 ÷  m-1
   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, c+(j+1)*k) );
returnează   sum;
}
```

4.3.INTERPOLAREA (partea 1)

4.3.1. INTERPOLAREA POLINOMIALĂ LAGRANGE

Se consideră funcţia dată prin urmatorul tabel:

x	x_0	x_1	.	x_k	.	x_n
y	y_0	y_1	.	y_k	.	y_n

$$P_n(x)=\sum_{i=0}^{n}P_n(x_i)\frac{\Pi_i(x)}{\Pi_i(x_i)}=\sum_{i=0}^{n}y_i\frac{\prod_{j=0,j\neq i}^{n}(x-x_j)}{\prod_{j=0,j\neq i}^{n}(x_i-x_j)}=\sum_{i=0}^{n}y_i\prod_{j=0,j\neq i}^{n}\frac{x-x_j}{x_i-x_j}$$

Polinomul de gradul *n* care trece prin *n*+1 puncte date, numit şi *polinomul de interpolare al lui Lagrange*, are forma:

$$P_n(x)=\sum_{i=0}^{n}y_i\prod_{j=0,j\neq i}^{n}\frac{x-x_j}{x_i-x_j}$$

4.3.1.1. Algoritmul. Polinomul lui Lagrange

```
real Lagrange
(
   întreg n,          // gradul polinomului de interpolare
   real x[ ],         // vectorul absciselor punctelor cunoscute
   real y[ ],         // vectorul ordonatelor punctelor cunoscute
   real x̄ ,           // punctul în care se calculează interpolarea
)
{                     // declararea variabilelor locale functiei
întreg  i, j ;
real sum ;
real prod ;
                                  // corpul de instrucţiuni al funcţiei
   sum=0;
   pentru i= 0 ÷ n
   {
   prod=1;
   pentru j= 0 ÷ n   dacă ( j ≠ i )   prod = prod * (x̄ - x[j])/(x[i] - x[j]);
   sum = sum + y[i] * prod;
   }
   returnează sum; }          // valoarea în punctul cerut
```

Lucrarea 5

INTERPOLAREA (partea 2 a)

METODE DE OPTIMIZARE (partea 1 a)

5.1 POLINOMUL DE INTERPOLARE DE SPEŢA ÎNTÂI AL LUI NEWTON

Face parte din clasa metodelor ***cu pas constant*** între abscise. Aceasta se traduce prin a avea o anumită zonă preferată, în care precizia este cea mai bună pentru un interval acoperit prin punctele x_0, .. ,x_n.

Acest polinom de interpolare se exprimă funcţie de ***diferenţele finite***. Fie $f:[a,b]\to\mathbf{R}$ şi reţeaua $x_0, x_1, x_2, ..., x_n$ cu pasul constant *h*.

Definiţia 5.1. Se numeşte *diferenţă finită* de *ordinul întâi* expresia:

$$\Delta f(x) = f(x+h) - f(x)$$

unde *h* este pasul constant.

Diferenţa finită de *ordinul n* se poate defini recursiv, folosindu-se de diferenţa finită de ordin *n-1*, şi are expresia de definiţie:

$$\Delta^n f(x) = \Delta\left(\Delta^{n-1} f(x)\right)$$

x_i	y_i	Δy_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	Δy_0					
x_1	y_1	Δy_1	$\Delta^2 y_0$	$\Delta^3 y_0$			
x_2	y_2	Δy_2	$\Delta^2 y_1$	$\Delta^3 y_1$	$\Delta^4 y_0$	$\Delta^5 y_0$	
x_3	y_3	Δy_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	Δy_4	$\Delta^2 y_3$	$\Delta^3 y_3$	$\Delta^4 y_2$		
x_5	y_5	Δy_5	$\Delta^2 y_4$				
x_6	y_6						

Definiţia. Se numeşte *putere generalizată de ordinul n a lui* x expresia:

$$x^{[n]} = x(x-h)(x-2h)...(x-(n-1)h)$$

Pentru h=0 puterea generalizată coincide cu puterea obişnuită.

1. *Diferenţa finită a puterii generalizate* este:

$$\Delta x^{[n]} = nhx^{[n-1]}$$

Fie funcţia tabelată dată în *Tabelul de mai sus*, unde reţeaua $x_0, x_1, x_2, x_3, ..., x_n$ este cu pasul constant *h*.

Prin cele *n+1* puncte trece un polinom de gradul *n* pe care îl căutăm de forma:

$$P_n(x) = C_0 + C_1(x-x_0)^{[1]} + C_2(x-x_0)^{[2]} + ... + C_n(x-x_0)^{[n]}$$

unde

$$(x-x_0)^{[i]} = (x-x_0)(x-x_1)...(x-x_{i-1}) \quad , \quad i = 1, 2, ...n$$

Ţinând cont de formulele de calcul ale coeficienţilor, *polinomul lui Newton de interpolare de speţa întâi* poate fi scris astfel:

$$P_n(x_0) = y_0 + \frac{\Delta y_0}{1!h}(x-x_0)^{[1]} + \frac{\Delta^2 y_0}{2!h^2}(x-x_0)^{[2]} + \cdots + \frac{\Delta^n y_0}{n!h^n}(x-x_0)^{[n]}$$

5.1.1 Algoritmul. Newton 1

```
real Newton_1
(
întreg n,                // gradul polinomului de interpolare
real x0,                 // abscisa primului punct cunoscut
real h,                  // pasul constant între abscisele cunoscute
real y[ ],               // vectorul ordonatelor punctelor cunoscute
real xp,                 // abscisa punctului în care se face interpolarea
)
{                        // declararea şi definirea variabilelor locale
real sum ;
real prod ;
integ i, j ;
                         // corpul de instrucţiuni al funcţiei
sum = y[0];              // pornesc cu suma de la y0
prod = 1 ;
 pentru i= 1… n
   {                     //calculul diferenţelor finite
   pentru j= 0… n-i    y[j] = y[j+1] - y[j];
```

$$prod = prod * \left(x_p - \left(x_0 + (i-1) * h\right)\right) * \frac{1}{h} * \frac{1}{i};$$

```
   sum = sum + y[0]*prod ;
   }
returnează sum ;              // valoarea interpolată
}
```

5.2 POLINOMUL DE INTERPOLARE DE SPEŢA A DOUA AL LUI NEWTON

Pentru funcţia dată în *Tabelul 6.1* se caută un polinom de gradul *n* care trece prin cele *n+1* puncte, sub forma:

$$P_n(x) = C_0 + C_1(x - x_n) + C_2(x - x_n)(x - x_{n-1}) + ... + C_n(x - x_n)(x - x_{n-1})...(x - x_1) \quad)$$

Polinomul de gradul *n* poate fi scris sub forma:

$$P_n\left(x\right) = y_n + \frac{\Delta y_n}{1!h}\left(x - x_n\right)^{[1]} + \frac{\Delta^2 y_{n-1}}{2!h^2}\left(x - x_{n-1}\right)^{[2]} + \cdots + \frac{\Delta^n y_1}{n!h^n}\left(x - x_1\right)^{[n]}$$

5.3 POLINOMUL LUI NEWTON DE INTERPOLARE CU DIFERENŢE DIVIZATE

Fie funcţia $f(x)$ dată sub forma celei prezentate în tabelul (5.1) unde reţeaua $x_0, x_1, x_2, ..., x_n$ din domeniul de definiţie al funcţiei nu are pas constant. Fiind o metodă cu pas variabil între abscise, precizia sa nu este preferenţială.

Definiţia 5.3. Se numeşte *diferenţă divizată de ordinul k+i* a funcţiei f expresia :

$$f\left(x_{i-1}, x_i, x_{i+1}, ..., x_{i+k}\right) = \frac{f\left(x_i, x_{i+1}, ..., x_{i+k}\right) - f\left(x_{i-1}, x_i, ..., x_{i+k-1}\right)}{x_{i+k} - x_{i-1}}$$

Se determină polinomul de gradul *n*, de forma :

$$P_n(x) = C_0 + C_1(x - x_0) + C_2(x - x_0)(x - x_1) + ... + C_n(x - x_0)(x - x_1)...(x - x_{n-1})$$

cunoscând *n+1* puncte (x_i, y_i), *i*=0, ...,*n* , care verifică polinomul.

Se observă că $P_n(x_0) = y_0 = C_0$.

Se calculează *diferenţa divizată* de ordinul întâi pentru polinomul $P_n(x)$ şi se face $x = x_1$. Rezultă:

$$P_n(x_0, x_1) = C_1.$$

Calculând în continuare, se determină diferenţa divizată de ordinul *k* şi luând pe $x = x_k$ se obţine valoarea coeficientului C_k :

$$C_k = P_n(x_0, x_1, x_2, ..., x_k), \qquad k=0, 1, ..., n.$$

Polinomul se scrie acum sub forma:

$$P_n(x) = y_0 + P_n(x_0, x_1)(x - x_0) + P_n(x_0, x_1, x_2)(x - x_0)(x - x_1) + ...$$
$$+ P_n(x_0, ..., x_n)(x - x_0)(x - x_1)...(x - x_{n-1})$$

unde fiecare diferenţă divizată se calculează cu ajutorul formulei din definitia (5.3). Polinomul obţinut poartă numele de *polinomul lui Newton de interpolare cu diferenţe divizate.*

5.3.1 Algoritmul . Newton 3

```
real Newton_3
(
   întreg n,                    // numărul de puncte date ale funcţiei
   real x[ ],                   // vectorul absciselor punctelor date
   real y[ ],                   // vectorul ordonatelor punctelor date
   real x̄ ,                     // punctul în care se interpolează funcţia
)
{                               // declararea şi definirea variabilelor locale
întreg i, j ;
real sum ;
real prod ;
                                // corpul de instrucţiuni al funcţiei
   sum = y[0] ;
   prod = 1 ;
   pentru i= 1.. n
   {
                                // calculul diferenţelor divizate
         pentru j= 0.. n-i      y_j = (y_{j+1} - y_j)/(x_{j+i} - x_j);

         prod = prod*(x̄ - x_{i-1}) ;
         sum = sum + y[0]*prod ;
   }
         returnează  sum ;}
```

5.4. METODE DE OPTIMIZARE (partea 1)

PREZENTAREA TEORETICĂ

Metodele de optimizare se clasifică, în raport cu problema care se pune pentru funcţia ţintă sau scop, astfel:

1. Metode ce determină expresia analitică a funcţiei, care aproximează cel mai bine o funcţie tabelată dată prin puncte;
2. Metode ce determină diferiţi parametri ai funcţiei scop (ţintă), pentru a obţine un extrem al funcţiei.

5.4.1. METODA CELOR MAI MICI PĂTRATE

Presupunem că avem o funcţie definită printr-un tabel de valori. Se mai spune că funcţia este dată implicit, deci nu i se cunoaşte forma în mod direct. Problema care se pune este să determinăm funcţia analitică *care aproximează cel mai bine* datele din tabel sau curba care trece prin punctele tabelului.

5.4.1.1 REGRESIA LINIARĂ

Se consideră funcţia tabelată:

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 funcţiei. Se cere să se determine funcţia liniară de forma generală

$$y = ax + b$$

care să aproximeze cel mai bine funcţia tabelată ***astfel ca eroarea să fie minimă***.

$$a = \frac{m\sum_{i=1}^{m} x_i y_i - \sum_{i=1}^{m} x_i \sum_{i=1}^{m} y_i}{m\sum_{i=1}^{m} x_i^2 - \left(\sum_{i=1}^{m} x_i\right)^2}$$

$$b = \frac{\sum_{i=1}^{m} x_i^2 \sum_{i=1}^{m} y_i - \sum_{i=1}^{m} x_i \sum_{i=1}^{m} x_i y_i}{m\sum_{i=1}^{m} x_i^2 - \left(\sum_{i=1}^{m} x_i\right)^2}$$

$$\mathrm{E} = \sum_{i=1}^{nm} (y_i - ax_i - b)^2$$

5.4.1.2. Algoritm. Regresia liniară

```
void Reg_Lin
(
   întreg m,            // numărul experienţelor
   real x[ ],           // vectorul absciselor funcţiei
   real y[ ],           // vectorul ordonatelor funcţiei
   real *pA,            // adresa coeficientului lui x (pointer)
   real *pB             // adresa termenului liber (pointer)
   real *E              // adresa erorii (pointer)
)
{                       // declararea şi definirea variabilelor locale
   real sx, sy, sxx, sxy ;
   întreg i ;

                        // corpul de instrucţiuni al funcţiei
   sx= 0 ; sy = 0 ; sxy = 0 ; sxx = 0;
   pentru i= 1 ÷ m
   {
         sx = sx + x[i] ;
         sy = sy + y[i] ;
         sxy = sxy + x[i]*y[i] ;
         sxx = sxx + x[i]*x[i] ;
   }
         *pA = (m·sxy-sx·sy)/(m·sxx-sx·sx) ;   *pB = (·sxx·sy-sx·sxy)/(m·sxx-sx·sx) ;
                        // se calculeaza eroarea
   pentru i= 1 ÷ m
         *E=*E+(y[i]-(*pa)*x[i]-(*pb))²
  }
```

5.4.2.1.REGRESIA HIPERBOLICĂ

Fie funcţia numerică dată în tabel. Cautam determinării funcţiei hiperbolice: $y = \frac{1}{ax+b}$ care aproximează cel mai bine funcţia numerică. Valorile lui *a* şi *b* sunt:

$$a = \frac{(\sum_{i=1}^{m} x_i)(\sum_{i=1}^{m} \frac{1}{y_i}) - m\sum_{i=1}^{m} \frac{x_i}{y_i}}{\left(\sum_{i=1}^{m} x_i\right)^2 - m\sum_{i=1}^{m} x_i^2}$$

$$b = \frac{(\sum_{i=1}^{m} \frac{1}{y_i})(\sum_{i=1}^{m} x_i^2) - (\sum_{i=1}^{m} \frac{x_i}{y_i})(\sum_{i=1}^{m} x_i)}{m\sum_{i=1}^{m} x_i^2 - \left(\sum_{i=1}^{m} x_i\right)^2}$$

$$E = \sum_{i=1}^{m} (\frac{1}{y_i} - ax_i - b)^2$$

Pentru a liniariza rezolvarea sistemului s-a lucrat cu inversele valorilor lui *y*, adică *1/y*, motiv pentru care sumele *sy* şi *sxy* conţin valorile inversate pentru *y*.

5.4.2.2. Algoritm. Regresia hiperbolică

```
void Reg_Hip
(
întreg m,                   // numărul experienţelor
real x[ ],                  // vectorul absciselor funcţiei numerice
real y[ ],                  // vectorul ordonatelor funcţiei numerice
real *pA                    // adresa coeficientului lui a (pointer)
real *pB                    // adresa coeficientului lui b (pointer)
real *E                     // adresa erorii (pointer)
)
{                           // declararea şi definirea variabilelor locale
real sx , sy , sxx , sxy ;
întreg i ;
                            // corpul de instrucţiuni al funcţiei
   sx = 0 ; sy = 0 ; sxy = 0 ; sxx = 0 ;
   pentru i= 1÷m
   {
         sx = sx + x[i] ;
         sy = sy+ 1/y[i] ;
         sxy= sxy+ x[i]/y[i] ;
         sxx = sxx + x[i]*x[i] ;
   }
   *pA = (m·sxy − sx·sy)/(m·sxx − sx sx) ; *pB = (·sxx·sy-sxy sx)/(m·sxx-sx·sx) ;
                            // se calculeaza eroarea
   pentru i= 1÷ m
         *E=*E+( y[i]-1/((*pa)*x[i]+(*pb)) )^2 ;

  }
```

Lucrarea 6

METODE DE OPTIMIZARE (partea 2 a)
REZOLVAREA NUMERICĂ A ECUAŢIILOR DIFERENŢIALE DE ORDINUL I

6.1. REGRESIA EXPONENŢIALĂ

Dacă se constată că funcţia numerică din tabelul 8.1, reprezentată grafic, se aseamănă foarte mult cu o exponenţială, atunci se consideră funcţia exponenţială, de forma generală:

$$y = a \cdot b^x$$

în care a şi b sunt constante pozitive.

Pentru un calcul comod al constantelor a şi b se logaritmează funcţia:

$$\ln y = \ln a + x \ln b$$

şi se determină constantele astfel încât eroarea pătratică de ansamblu să fie *minimă*.

$$a = \exp\left\{\frac{\left[(\sum_{i=1}^{m} \ln y_i)(\sum x_i^2) - (\sum_{i=1}^{m} x_i)(\sum_{i=1}^{m} x_i \ln y_i)\right]}{\left[m\sum_{i=1}^{m} x_i^2 - \left(\sum_{i=1}^{m} x_i\right)^2\right]}\right\}$$

$$b = \exp\left\{\frac{\left[m\sum_{i=1}^{m} x_i \ln y_i - (\sum_{i=1}^{m} x_i)(\sum_{i=1}^{m} \ln y_i)\right]}{\left[m\sum_{i=1}^{m} x_i^2 - \left(\sum_{i=1}^{m} x_i\right)^2\right]}\right\}$$

$$E = \sum_{i=1}^{m} (y_i - ab^{x_i})^2$$

6.1.1 Algoritm. Regresia exponenţială

```
void Reg_Exp
(
   întreg m,          // numărul absciselor funcţiei
   real x[ ],         // vectorul absciselor funcţiei numerice
   real y[ ],         // vectorul ordonatelor funcţiei numerice
   real *pA,          // adresa coef lui a al fct exponentiale (pointer)
   real *pB           // adresa coef lui b al funcţiei exponentiale (pointer)
   real *E            // adresa erorii (pointer)
)
{                     // declararea şi definirea variabilelor locale
real sx  ;
real sxx ;
real sy ;
real sxy ;
întreg i ;
                      //  corpul de instrucţiuni al funcţiei
sx= 0 ; sy = 0 ; sxy = 0 ; sxx = 0 ;
pentru i= 1 ÷ m
```

```
    {
            sx = sx + x[i] ;
            sxx = sxx + x[i]*x[i] ;
            sy = sy+log( y[i]) ;
            sxy = sxy+ x[i] * log( y[i]) ;
    }
```

$$*pA = \exp\left(\frac{s_y \cdot s_{xx} - s_x \cdot s_{xy}}{m \cdot s_{xx} - s_x \cdot s_x}\right); \qquad *pB = \exp\left(\frac{m s_{xy} - s_x s_y}{m \cdot s_{xx} - s_x \cdot s_x}\right);$$

// se calculeaza eroarea

```
    pentru i= 1÷ m
                *E=*E+( y[i]-(*pa)*(*pb)x[i] )2
}
```

6.2. REGRESIA GEOMETRICĂ

Dacă funcţia numerică din tabel se aseamănă cu o funcţie de tip geometric, vom căuta să determinăm funcţia de forma:

$$y = ax^b$$

care să aproximeze cel mai bine funcţia numerică. Adică aceeaşi condiţie de eroare pătratică, de ansamblu, minimă.

Şi în această situaţie se recurge la o logaritmare a relaţiei, pentru a aduce rezolvarea în domeniul liniar. Avem:

$$lny = ln(a) + b*ln(x)$$

Se obţin următoarele valori pentru a şi b:

$$a = \exp\frac{\left(\sum_{i=1}^{m}\ln(y_i)\right)\left(\sum_{i=1}^{m}\ln^2(x_i)\right) - \left(\sum_{i=1}^{m}\ln(x_i)\right)\left(\sum_{i=1}^{m}\ln(x_i)\ln(y_i)\right)}{m\left(\sum_{i=1}^{m}\ln^2(x_i)\right) - \left(\sum_{i=1}^{m}\ln(x_i)\right)^2}$$

$$b = \frac{m\left(\sum_{i=1}^{m}\ln(y_i)\ln(x_i)\right) - \left(\sum_{i=1}^{m}\ln(x_i)\right)\left(\sum_{i=1}^{m}\ln(y_i)\right)}{m\left(\sum_{i=1}^{m}\ln^2(x_i)\right) - \left(\sum_{i=1}^{m}\ln(x_i)\right)^2}$$

$$E = \sum_{i=1}^{m}(y_i - ax_i^b)^2$$

6.2.1 Algoritmul. Regresia geometrică

```
void Reg_Geo
(
 întreg m,                  // numărul de abscise
 real x[ ],                 // vectorul absciselor funcţiei numerice
 real y[ ],                 // vectorul ordonatelor funcţiei numerice
 real *pA,                  // adresa coef lui a al fct geometrice (pointer)
 real *pB                   // adresa coef lui a al fct geometrice (pointer)
 real *E                    // adresa erorii (pointer)

)
{                           // declararea şi definirea variabilelor locale
real sx ;
```

```
real sy ;
real sxy ;
real sxx  ;
întreg i ;

                                    // corpul de instrucţiuni al funcţiei
    sx = 0 ; sy = 0 ; sxx = 0 ; sxy= 0 ;
    pentru i= 1÷ m
    {
            sx = sx + log(x[i]) ;
            sxx = sxx + log(x[i])*log(x[i]) ;
            sy = sy + log (y[i]) ;
            sxy = sxy + log(x[i])* log(y[i]) ;
    }
```

$$*pA = \exp\left(\frac{s_y \cdot s_{xx} - s_x \cdot s_{xy}}{m \cdot s_{xx} - s_x \cdot s_x}\right); \qquad *pB = \frac{m \cdot s_{xy} - s_x \cdot s_y}{m \cdot s_{xx} - s_x \cdot s_x};$$

```
                                    // se calculeaza eroarea
    pentru i= 1÷ m
                *E=*E+( y[i]-(*pa)*x[i](*pb) )2
}
```

6.3. REGRESIA TRIGONOMETRICĂ

Se consideră funcţia numerică dată în tabelul 8.1 şi funcţia trigonometrică de forma:

$$y = a + b \cdot \cos(\omega \cdot x)$$

care aproximează cel mai bine funcţia numerică.

Valorile lui *a* şi *b* sunt:

$$a = \frac{\left(\sum_{i=1}^{m} y_i\right)\left(\sum_{i=1}^{m} \cos^2 \omega x_i\right) - \left(\sum_{i=1}^{m} \cos \omega x_i\right)\left(\sum_{i=1}^{m} y_i \cos \omega x_i\right)}{m\left(\sum_{i=1}^{m} \cos^2 \omega x_i\right) - \left(\sum_{i=1}^{m} \cos \omega x_i\right)^2}$$

$$b = \frac{m\left(\sum_{i=1}^{m} y_i \cos \omega x_i\right) - \left(\sum_{i=1}^{m} \cos \omega x_i\right)\left(\sum_{i=1}^{m} y_i\right)}{m\left(\sum_{i=1}^{m} \cos^2 \omega x_i\right) - \left(\sum_{i=1}^{m} \cos \omega x_i\right)^2}$$

$$E = \sum_{i=1}^{m} (y_i - a - b * \cos(wx^i))^2$$

$\omega = 2 \cdot \pi \cdot f = \dfrac{2 \cdot \pi}{T}$ unde f - frecvenţa, ω - pulsaţia, iar T - perioada.

ω se stabileşte funcţie de periodicitatea funcţiei numerice date.

6.3.1 Algoritmul. Regresia trigonometrică

```
void Reg_Tri
(
    întreg m,                   // numărul  absciselor funcţiei numerice
    real x[ ],                  // vectorul absciselor funcţiei numerice
    real y[ ],                  // vectorul ordonatelor funcţiei numerice
    real *pA,                   // adresa coef a din expresia fct trig. (pointer)
    real *pB                    // adresa coef b din expresia fct trig. (pointer)
    real *E                     // adresa erorii (pointer)
)
```

```
{                                   // declararea şi definirea variabilelor locale
real ω;                             // pulsaţia (w =2*PI*T)
real s y ;
real scos 2x: ;
real scos x  ;
real sycosx ;
întreg i ;
   s y = 0 ; scos 2x = 0 ; scos x = 0 ; sycos x = 0 ; // corpul de instrucţiuni al funcţiei
   pentru i= 1÷ m
   {
   s y = s y + y[i] ;
   scos 2x = scos 2x + cos(ω xi) * cos(ω xi) ;
   scos x = scos x + cos(ω xi) ;
   sycos x = sycos x + yi* cos(ω xi) ;
   }
   *pA = (s_y · s_cos 2x - s_cos x · s_y cos x) / (m · s_cos 2x - s_cos x · s_cos x) ;   *pB = (m · s_y cos x - s_cos x · s_y) / (m · s_cos 2x - s_cos x · s_cos x) ;
                                    // se calculeaza eroarea
   pentru i= 1÷ m
                 *E=*E+( y[i]-(*pa)-(*pb)*cos(w*x[i]) )2

}
```

REZOLVAREA NUMERICĂ A ECUAŢIILOR DIFERENŢIALE DE ORDINUL 1

6.4. METODA LUI EULER ÎMBUNĂTĂŢITĂ (media pantelor)

Această metodă face parte din grupul de metode de tip *Runge-Kutta de ordin 2*. deoarece dă aceleaşi soluţii ca şi metoda lui Taylor până la h^2 .

Utilizează media pantelor din punctele (x_m, y_m) şi (x_{m+1}, y_{m+1}) unde $x_{m+1} = x_m + h$ iar $y_{m+1} = y_m + hy'_m$ este ordonata obţinută din ecuaţia tangentei în punctul M la curbă cu panta $y'_m = f(x_m, y_m)$. Grafic metoda este prezentată în figura 6.1. În acest punct H, de coordonate $(x_{m+1},\ y_m + hy'_m)$ se calculează panta la curbă, anume:

$$\overline{y}_{m+1} = f\left(x_m + h, y_m + hy'_m\right)$$

Se face media dintre panta dreptei prin punctul M şi panta dreptei prin punctul H, şi se notează cu:

$$z_m = \frac{1}{2}\left(f\left(x_m, y_m\right) + f\left(x_m + h, y_m + hy'_m\right)\right)$$

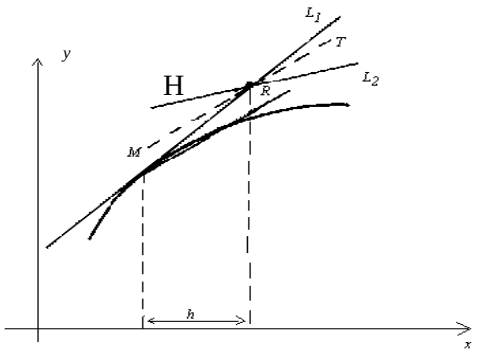

Fig. 6.1. - Calculul grafic al soluţiei prin metoda lui Euler îmbunătăţită.

Dreapta de pantă z_m este HT. Cu această pantă z_m se trasează coarda prin M (echivalent cu a spune că se translatează dreapta HT în punctul M) şi se intersectează cu dreapta (verticala) din punctul $x = x_{m+1}$, rezultând punctul R, de coordonate (x_{m+1}, y_{m+1}) considerat al doilea punct de pe curba soluţie, primul fiind considerat $M(x_m, y_m)$. Ecuaţia coardei prin M este:

$$y = y_m + 0.5 \cdot \left(f(x_m, y_m) + f\left(x_m + h, y_m + hy_m^{'}\right)\right)(x - x_m)$$

Punctul R, care reprezintă soluţia numerică a ecuaţiei diferenţiale, are ordonata:

$$y_{m+1} = y_m + 0.5 \cdot h \cdot \left(f(x_m, y_m) + f\left(x_m + h, y_m + hy_m^{'}\right)\right)$$

unde $y_m^{'} = f(x_m, y_m)$.

Această formulă (de mai sus) reprezintă formula de calcul a soluţiilor numerice pentru o ecuaţie diferenţială ordinară de ordinul întâi.

6.4.1. Algoritmul. Metoda lui Euler îmbunătăţită

```
void Euler_Imb
(
real x0,                //abscisa pct iniţial prin care trece graficul soluţiei
real y0,                //ordonata pct iniţial prin care trece graficul soluţiei
real h,                 //pasul între abscisele punctelor de calcul
real y[],               // vectorul ordonatelor (ce vor fi calculate)
real x[],               // vectorul absciselor (ce vor fi calculate)
intreg n                //numarul de esntioane ale solutiei ec diferentale
)
{                       // declaraţiile şi definiţiile variabilelor locale
întreg i;

   y[0]= y0;  x[0]=x0;
   pentru i= 1÷ n
   {
x[i]= x[0]+i*h;
y[i]=y[i-1]+0.5*h*(f(x[i-1],y[i-1])+f(x[i],y[i-1]+h*f(x[i-1],y[i-1])));
   }
}
```

6.5. METODA LUI EULER MODIFICATĂ (media punctelor)

În cadrul acestei metode nu se mediază pantele, ci se evaluează panta în punctul mediu.

Considerăm curba din figura 6.2 în care se dă punctul iniţial M prin care să treacă curba soluţie a ecuaţiei diferenţiale şi dorim să determinăm cel de al doilea punct al soluţiei. Prin punctul (x_m, y_m) dat, cunoscut sau determinat, se duce tangenta prin M şi se determină pe această tangentă punctul H de coordonate (x_{m+1}, y_{m+1}). Până în acest punct nu există deosebire cu metoda Euler îmbunătăţită.

Tangenta prin punctul M are ecuaţia

$$y = y_m + f(x_m, y_m)(x - x_m)$$

Prin intersecţia cu dreapta (verticala) în punctul $x = x_m + h$ rezultă coordonatele punctului H:

$H(x_{m+1} = x_m + h,\ \ y_{m+1} = y_m + hf(x_m, y_m)$, prezentat în figura 6.2.

Metoda face media coordonatelor punctelor M şi H şi determină punctul:

$$P\left(x_p = x_m + \frac{h}{2},\ \ y_p = y_m + \frac{h}{2} f(x_m, y_m)\right)$$

Se calculează panta soluţiei *y* în acest punct P şi rezultă:

$$z_m = f\left(x_m + \frac{h}{2},\ \ y_m + \frac{h}{2} f\left(x_m, y_m\right)\right)$$

Se scrie ecuaţia dreptei care trece prin P, de pantă z_m, şi se intersectează cu verticala în punctul următor pe abscisă:

$$x_{m+1} = x_m + h \qquad \varepsilon$$

Dreapta MN are ecuaţia:

$$y = y_m + f\left(x_m + \frac{h}{2},\ \ y_m + \frac{h}{2} f\left(x_m, y_m\right)\right)\left(x - x_m\right)$$

Prin intersecţia dreptei (8.11) cu verticala în punctul $x = x_{m+1} = x_m + h$ se determină ordonata R:

$$y_{m+1} = y_m + h\, f\left(x_m + \frac{h}{2},\ \ y_m + \frac{h}{2} f\left(x_m, y_m\right)\right)$$

care reprezintă formula de calcul a ordonatelor soluţiei ecuaţiei diferenţiale.

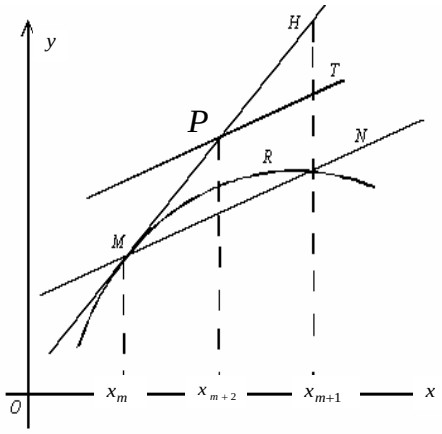

Fig.6.2. - Calculul grafic al soluţiei ecuaţiei diferenţiale prin metoda lui Euler modificată.

6.5.1. Algoritmul. Metoda lui Euler modificată

```
void Eul_Mod
(
real x0,                //abscisa pct iniţial prin care trece graficul soluţiei
real y0,                //ordonata pct iniţial prin care trece graficul soluţiei
real h,                 //pasul între abscisele punctelor de calcul
real y[],               // vectorul ordonatelor (ce vor fi calculate)
real x[],               // vectorul absciselor (ce vor fi calculate)
intreg n                //numarul de esntioane ale solutiei ec diferentale
)

{                       // declaraţiile şi definiţiile variabilelor locale
întreg i;


   x[0]= x0;  y[0]= y0;
   pentru i= 1÷n
   {              // se calculează abscisele şi ordonatele solutiei
   x[i]= x[0] + i*h;
   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]) );
   }
}
```

Lucrarea 4(noua varianta)

4.1. INTEGRAREA NUMERICĂ

4.1.1. METODA LUI RICHARDSON

Această metodă dă o precizie mai bună de calcul a integralei numerice decât metoda trapezului şi s-a obţinut prin modificarea metodei trapezului. Este o metodă în care apare *dubla-divizare* a intervalului de integrare.

Se pleacă de la eroarea de trunchiere a metodei trapezului, care pentru o diviziune de lăţime *h* este:

$$e_{T,h} = Ch^2, \quad h = (b-a)/n,$$

Pentru cealaltă diviziune, $k = (b-a)/m$, se obţine eroarea de trunchiere:

$$e_{T,k} = Ck^2$$

Pentru fiecare diviziune se poate scrie:

$$I = I_h + e_{T,h}$$

respectiv

$$I = I_k + e_{T,k}$$

Eliminând constanta *C* între relaţiile de mai sus, se obţine expresia:

$$I = I_h + \frac{I_h - I_k}{\left(\frac{k}{h}\right)^2 - 1}$$

expresie ce poartă denumirea de *formula lui Richardson*.

4.1.1. Metoda lui Richardson (algoritm)

```
real I_Richardson
(
   real ls,                    // limita stângă a intervalului de integrare
   real ld,                    // limita dreaptă a intervalului de integrare
   întreg n,                   // numărul de subintervale_1
   întreg m                    // numărul de subintervale_2
)
{
                               // declararea şi definirea variabilelor locale
real h;                        // valoarea lungimii unui subinterval divizat  in n sub-diviziuni
real k;                        // valoarea lungimii unui subinterval divizat  im m sub-diviziuni
real sumh;                     // valoarea integralei cu diviziunea h
real sumk;                     // valoarea integralei cu diviziunea k
real sum;                      // valoarea integralei
întreg  i;                     // indice întreg pentru ciclul 'for()'

                               // instrucţiunile de calcul
```

$h = \frac{ld - ls}{n}$; $k = \frac{ld - ls}{m}$;

```
   Sumh=(f(ls)+f(ld))/2*h;
   Sumk=(f(ls)+f(ld))/2*k;

pentru i= 1 ÷ n-1
        sumh = sumh + h*f(ls + i*h);
pentru i= 1 ÷ m-1
        sumk = sumk + k*f(ls + i*k);
sum=sumh + (sumh-sumk)/( (k/h)*(k/h)-1 );
returnează sum;
```

```
}
```

Această metodă determină punctele de divizare ale intervalului de integrare astfel ca eroarea de calcul a integralei să fie minimă.

4.1.2. CALCULUL NUMERIC AL INTEGRALELOR DUBLE

Pentru simplitate vom considera domeniul de integrare al funcţiei de două variabile un dreptunghi

$$\iint_D f(x,y)\,dxdy = \int_a^b \int_c^d f(x,y)\,dxdy$$

reprezintă integrala dublă dintr-o funcţie de două variabile. Pentru calculul valorii acestei integrale vom utiliza *formula de cubatură a trapezului* în continuare.

FORMULA DE CUBATURĂ A TRAPEZULUI

Intervalele de integrat [a,b] şi [c,d] se împart în subintervale de lungimi egale

$$h=\frac{b-a}{n} \quad \text{şi, respectiv} \quad k=\frac{d-c}{m}$$

şi se consideră dreptunghiul cu vârfurile $[x_i, y_i]$, $[x_{i+1}, y_i]$, $[x_{i+1}, y_{i+1}]$, $[x_i, y_{i+1}]$

$$I=\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} I_{ij} = \frac{kh}{4}\sum_{i=0}^{n-1}\sum_{j=0}^{m-1}\left[f(x_i,y_j)+f(x_i,y_{j+1})+f(x_{i+1},y_j)+f(x_{i+1},y_{j+1})\right]$$

expresie cunoscută sub numele de *formula de **cubatură** a trapezului*. Unde $x_i=a+i*h$, $y_j=c+j*k$;

4.1.2. Algoritmul. Metoda cubaturii trapezului

```
real I_Cubatura_Trapez
(
   real a,              // limita stângă a intervalului de integrare  pe axa Ox
   real b               // limita dreaptă a intervalului de integrare pe axa Ox
   real c,              // limita stângă a intervalului de integrare  pe axa Oy
   real d,              // limita dreaptă a intervalului de integrare  pe axa Oy
   întreg n,            // numărul de subintervale pe axa Ox
   întreg m             // numărul de subintervale pe axa Oy
)

{
                        // declararea şi definirea variabilelor locale
întreg i, j;
real h;                 // valoarea lungimii unui subinterval diviziunea n
real k;                 // valoarea lungimii unui subinterval diviziunea m
real sum;               // valoarea integralei

                        // instrucţiunile de calcul ale metodei
h= (b-a)/n ;   k= (d-c)/m ;
sum = 0;
pentru i= 0 ÷  n-1
  pentru j= 0 ÷  m-1
   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, c+(j+1)*k) );
returnează   sum;
}
```

4.2.INTERPOLAREA

4.2.1. I

NTERPOLAREA POLINOMIALĂ LAGRANGE

Se consideră funcţia dată prin urmatorul tabel:

x	x_0	x_1				x_n
y	y_0	y_1				y_n

$$P_n(x)=\sum_{i=0}^{n}P_n(x_i)\frac{\Pi_i(x)}{\Pi_i(x_i)}=\sum_{i=0}^{n}y_i\frac{\prod_{j=0,j\neq i}^{n}(x-x_j)}{\prod_{j=0,j\neq i}^{n}(x_i-x_j)}=\sum_{i=0}^{n}y_i\prod_{j=0,j\neq i}^{n}\frac{x-x_j}{x_i-x_j}$$

Polinomul de gradul *n* care trece prin *n*+1 puncte date, numit şi *polinomul de interpolare al lui Lagrange*, are forma:

$$P_n(x)=\sum_{i=0}^{n}y_i\prod_{j=0,j\neq i}^{n}\frac{x-x_j}{x_i-x_j}$$

4.2.1. Algoritmul. Polinomul lui Lagrange

```
real Lagrange
(
   întreg n,          // gradul polinomului de interpolare
   real x[ ],         // vectorul absciselor punctelor cunoscute
   real y[ ],         // vectorul ordonatelor punctelor cunoscute
   real x̄ ,           // punctul în care se calculează interpolarea
)
{                     // declararea variabilelor locale functiei
întreg  i, j ;
real sum ;
real prod ;
                                  // corpul de instrucţiuni al funcţiei
   sum=0;
   pentru i= 0 ÷ n
   {
   prod=1;
   pentru j= 0 ÷ n   dacă ( j ≠ i )   prod = prod * (x̄ − x[j])/(x[i] − x[j]);
   sum = sum + y[i] * prod;
   }
returnează sum;}      // valoarea în punctul cerut
```

4.2.2. POLINOMUL DE INTERPOLARE DE SPEŢA ÎNTÂI AL LUI NEWTON

Face parte din clasa metodelor ***cu pas constant*** între abscise. Aceasta se traduce prin a avea o anumită zonă preferată, în care precizia este cea mai bună pentru un interval acoperit prin punctele x_0, .. ,x_n.

Acest polinom de interpolare se exprimă funcţie de ***diferenţele finite***. Fie $f:[a,b]\to\mathbf{R}$ şi reţeaua $x_0,x_1,x_2,...,x_n$ cu pasul constant *h*.

Definiţia 5.1. Se numeşte *diferenţă finită* de *ordinul întâi* expresia:

$$\Delta f(x)=f(x+h)-f(x)$$

unde *h* este pasul constant.

Diferenţa finită de *ordinul n* se poate defini recursiv, folosindu-se de diferenţa finită de ordin *n-1*, şi are expresia de definiţie:

$$\Delta^n f(x) = \Delta\left(\Delta^{n-1} f(x)\right)$$

x_i	y_i	Δy_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	Δy_0					
x_1	y_1	Δy_1	$\Delta^2 y_0$	$\Delta^3 y_0$			
x_2	y_2	Δy_2	$\Delta^2 y_1$	$\Delta^3 y_1$	$\Delta^4 y_0$	$\Delta^5 y_0$	
x_3	y_3	Δy_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	Δy_4	$\Delta^2 y_3$	$\Delta^3 y_3$	$\Delta^4 y_2$		
x_5	y_5	Δy_5	$\Delta^2 y_4$				
x_6	y_6						

Definiţia. Se numeşte *putere generalizată de ordinul n a lui* x expresia:

$$x^{[n]} = x(x-h)(x-2h)\ldots\left(x-(n-1)h\right)$$

Pentru h=0 puterea generalizată coincide cu puterea obişnuită.

2. *Diferenţa finită a puterii generalizate* este:

$$\Delta x^{[n]} = nhx^{[n-1]}$$

Fie funcţia tabelată dată în *Tabelul de mai sus*, unde reţeaua $x_0, x_1, x_2, x_3, \ldots, x_n$ este cu pasul constant *h*.

Prin cele *n+1* puncte trece un polinom de gradul *n* pe care îl căutăm de forma:

$$P_n(x) = C_0 + C_1(x-x_0)^{[1]} + C_2(x-x_0)^{[2]} + \ldots + C_n(x-x_0)^{[n]}$$

unde

$$(x-x_0)^{[i]} = (x-x_0)(x-x_1)\ldots(x-x_{i-1}) \quad , \quad i = 1, 2, \ldots n$$

Ţinând cont de formulele de calcul ale coeficienţilor, *polinomul lui Newton de interpolare de speţa întâi* poate fi scris astfel:

$$P_n(x_0) = y_0 + \frac{\Delta y_0}{1!h}(x-x_0)^{[1]} + \frac{\Delta^2 y_0}{2!h^2}(x-x_0)^{[2]} + \cdots + \frac{\Delta^n y_0}{n!h^n}(x-x_0)^{[n]}$$

4.2.2. Algoritmul. Newton 1

```
real Newton_1
(
întreg n,                // gradul polinomului de interpolare
real x0,                 // abscisa primului punct cunoscut
real h,                  // pasul constant între abscisele cunoscute
real y[ ],               // vectorul ordonatelor punctelor cunoscute
real xp,                 // abscisa punctului în care se face interpolarea
)
{                        // declararea şi definirea variabilelor locale
real sum ;
real prod ;
integ i, j ;
                         // corpul de instrucţiuni al funcţiei
sum = y[0];              // pornesc cu suma de la y0
prod = 1 ;
 pentru i= 1… n
   {                     //calculul diferenţelor finite
```

```
    pentru j= 0… n-i    y[j] = y[j+1] - y[j];
```

$$prod = prod * \left(x_p - \left(x_0 + (i-1) * h\right)\right) * \frac{1}{h} * \frac{1}{i};$$

```
    sum = sum + y[0]*prod ;
    }
returnează sum ;                // valoarea interpolată
}
```