Ministerul Educatiei din Republica Moldova
Facultatea Calculatoare Informatica si Microelectronica
Catedra “Informatica Aplicata”

RAPORT

La disciplina Cercetari operationale

Lucrare de Laborator Nr:1

TEMA: **Optimizarea necondiționată. Minimizarea funcțiilor cu ajutorul metodei gradientului cu fracționarea pasului.**

A efectuat: st.gr.MI-101

A verificat: lector superior

Chişinău 2012

Sarcina 12

Să se determine minimul global al funcției pătratice $2x^2 + 2xy + y^2 - 2x - 3y$ cu ajutorul metodei gradientului cu fracționarea pasului cu precizia $\varepsilon = 10^{-5}$.
Volorile initiale : a=3 b=2.

Descrierea algoritmului metodei gradientului cu fracționarea pasului

1. Se alege o valoare arbitrară α (una și aceeași la fiecare iterație), se determină punctul $z = x^{(k)} - \square \nabla f(x^{(k)})$ și se calculează f(z);
2. Se verifică condiția $f(z) - f(x) \leq -\square\square ||\nabla f(x^{(k)})||^2$ unde δ este o constantă arbitrară în intervalul (0,1);
3. Dacă este îndeplinită inegalitatea de mai sus, atunci valoarea α este acceptată și se va lua $\alpha_k = \square$. În caz contrar se fracționează α prin înmulțirea lui α cu un număr arbitrar pozitiv și $\gamma < 1$. Procesul continuă pînă cînd este satisfăcută inegalitatea de mai sus.
4. Se calculează $x^{(k+1)} = x^{(k)} - \alpha_k \nabla f(x^{(k)})$.

Este de dorit să continuăm procesul, pînă cînd $\left|\left|\nabla f(x^{(k)})\right|\right| < \varepsilon$.

Dacă rezolvăm manual această problemă prin metoda gradientului obținem soluția optimă $x = \begin{pmatrix} -0.5 \\ 2 \end{pmatrix}$. Avem nevoie de acest rezultat pentru a verifica corectitudinea calculelor efectuate de către program.

Codul sursa :

```
//---------------------------------------------------------------------------
#include<iostream.h>
#include<math.h>
#include<conio.h>
//---------------------------------------------------------------------------
        double a,b;
//Parametri pentru fiecare varianta
        int nMax;
//Numarul maxim de iteratii
        double eps;
//Valoarea aproximatiei
//---------------------------------------------------------------------------
```

```
        double f(double X[])
{
//Returneaza valoarea functiei in punctul x,y
        return a*X[0]*X[0] + 2*X[0]*X[1] + b*X[1]*X[1] - 2*X[0] - 3*X[1];
}
                void grad(const double *x,double *y)
        {//Calculeaza valoarea gradientului

                        y[0] = 2*a*x[0] + 2*x[1] - 2;y[1] = 2*x[0] + 2*b*x[1] - 3;
        }
        int metGradient(double *x)
        {
           //Metoda Gradientului
                        double y[2]; //Valoarea Gradientului
                        double dir[2];          // Directia de minimizare
                        double alfa,z[2]; // Alfa
                        double norma,t1,t2;
                        int n=0; // Numarul de iteratii
                                do
                                {
                                        grad(x,y);
                                        dir[0] = -y[0];
                                        dir[1] = -y[1];
                                        alfa = 5;
                                        int m=0;
                                do
                                {
                                z[0] = x[0] + alfa*dir[0];
                                z[1] = x[1] + alfa*dir[1];
                                norma = y[0]*y[0] + y[1]*y[1];
                        if(f(z)-f(x)<=-0.5*alfa*norma)m=1;
                        else alfa/=5;
                                }
                        while(!m);
                        x[0] = x[0] + alfa*dir[0];
                        x[1] = x[1] + alfa*dir[1];
                        grad(x,y);
                        n++;
                                }

                while(sqrt(n<nMax)&&(sqrt(y[1]*y[1]+y[0]*y[0])>eps));
                return n;
        }

                        double scalar(const double *x,const double *y,int n)
                {//Inmultirea scalara a 2 vectori
                        double s=0;
                        for(int i=0;i<n;i++)s += x[i]*y[i];
                        return s;
                }

//---------------------------------------------------------------------------
```

```
int main()
{
	double x[2],x1[2];
	int n;
	textbackground(9);
	clrscr();

	cout<<"Se determina minimul global al functiei";
	cout<<"f(x,y)=(a*x*x)+2*x*y+(b*y*y)-2*x-3*y"<<endl;
	cout<<"Introduceti valorile paramerilor a,b:";
	cin>>a>>b;
	cout<<"Introduci valorile initiale:";
	cin>>x[0]>>x[1];
		x1[0] = x[0];
		x1[1] = x[1];
	cout<<"Introduceti aproximatia:";
	cin>>eps;
	cout<<"Introduceti numarul maxim de iteratii:";
	cin>>nMax;
	cout<<"Metoda Gradientului:"<<endl<<endl;
		n=metGradient(x);
	cout<<"x[0]="<<x[0];
	cout<<endl<<"x[1]="<<x[1];
	cout<<endl<<"n="<<n;
	cout<<endl<<"f=="<<f(x);

	cin.get();
	cin.get();
	return 0;
}
```

Exemplu de calcul:

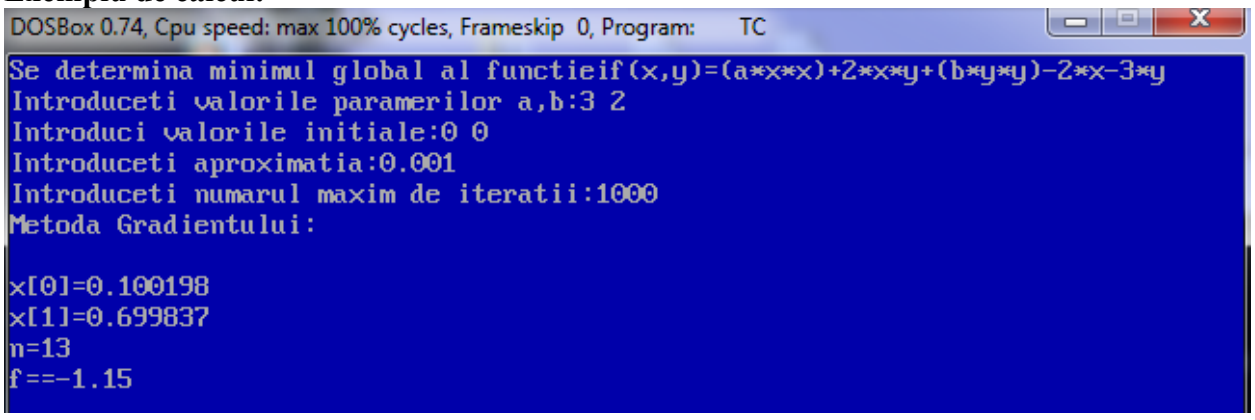

Fig1.valorile initiale (0 0)

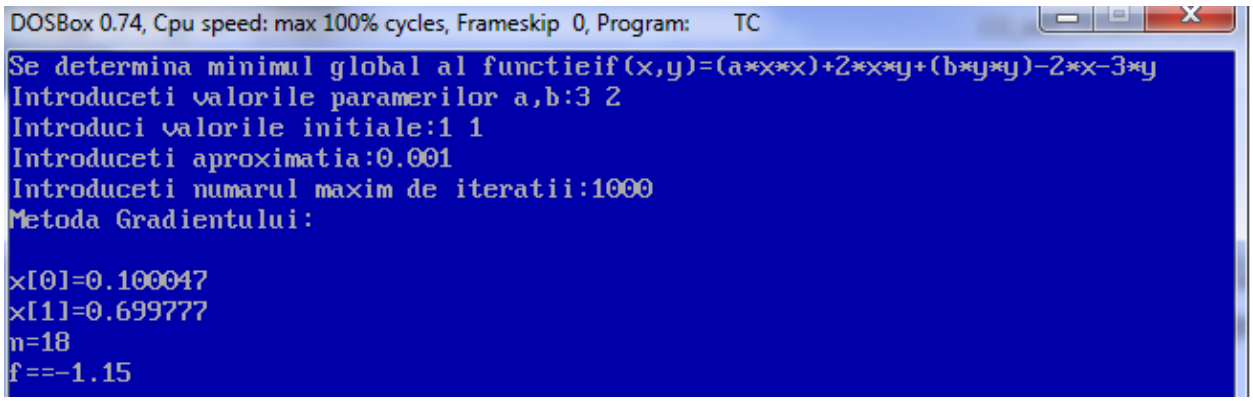

Fig2.valorile initiale (1 1)

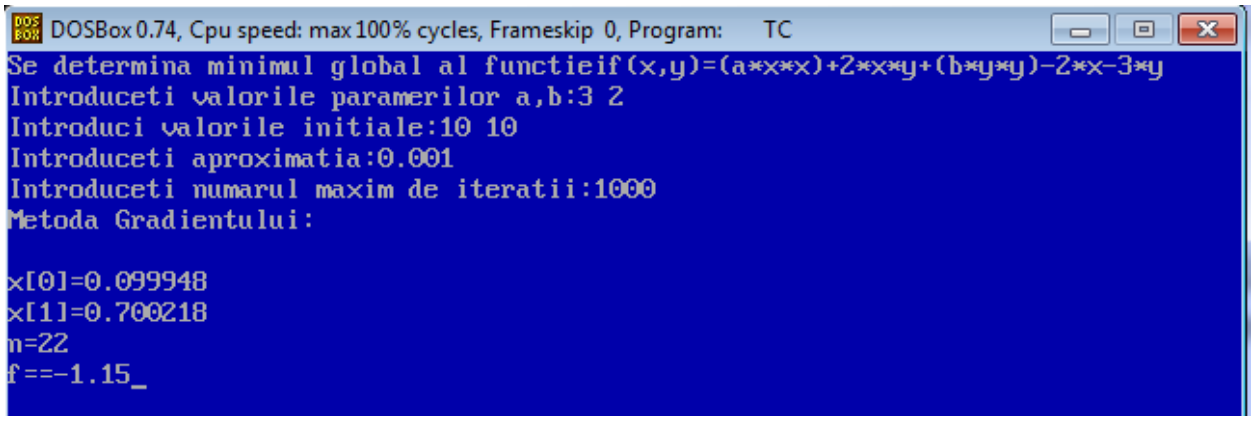

Fig3.valorile initiale (10 10)

In aceasta lucrare de laborator am programat algoritmi de optimizare neconditionata, in special: metoda gradientului si o metoda de directii conjugate – Hestenes-Steifel. Cea dinurma este particulara pentru rezolvarea functiilor patratice (si rezolvarea unui sistem deecuatii liniare cu matricea pozitiv definita). In general aceste metode se deosebesc prinalegerea directia de deplasare si pasul acesteia in pro cesul de minimizare a functiei

Ministerul Educatiei din Republica Moldova
Facultatea Calculatoare Informatica si Microelectronica
Catedra “Informatica Aplicata”

RAPORT

La disciplina Cercetari operationale

Lucrare de Laborator Nr:2

TEMA: **Optimizarea necondiționată. Minimizarea funcțiilor cu ajutorul metodei Hestenes-Stiefel. Compararea metodei Hestenes-Stiefel și metodei gradientului cu fracționarea pasului.**

A efectuat: st.gr.MI-101 Ursu Ion

A verificat: lector superior Catruc Mariana

Chişinău 2012

Sarcina 12

Să se determine minimul global al funcției pătratice $2x^2 + 2xy + y^2 - 2x - 3y$ cu ajutorul metodei Hestenes-Stiefel. Să se compare eficiența algoritmului Hestenes-Stiefel și algoritmului metodei gradientului cu fracționarea pasului. Volorile initiale : a=3 b=2.

Descrierea algoritmului metodei Hestenes-Stiefel

1. Se alege arbitrar $x^{(0)} \in R$ și se determină $\nabla f(x^{(0)})$. Dacă $\nabla f(x^{(0)}) = 0$, atunci $x^{(0)}$ este soluție optimă. STOP. În caz contrar se consideră $d^{(0)} = -\nabla f(x^{(0)})$, și se trece la pasul următor;
2. Se determină lungimea pasului α_k de-a lungul direcției $d^{(k)}$, care pleacă din $x^{(k)}$, ceea ce revine la minimizarea în raport cu parametrul scalar α al funcției $\varphi(\alpha) = f(x^k + \alpha d^{(k)})$. Determinarea lui α_k poate fi efectuată prin formula: $\alpha_k = -\frac{(\nabla f(x^{(k)}), d^{(k)})}{(Ad^{(k)}, d^{(k)})}$;
3. Se construiește o aproximație nouă: $x^{(k+1)} = x^{(k)} + \alpha_k d^{(k)}$, dacă $\nabla f(x^{(k+1)}) = 0$ atunci am aflat soluția optimă, STOP. Altfel trecem la pasul următor;
4. Se construiește direcția $d^{(k+1)} = -\nabla f(x^{(k+1)}) + \frac{(\nabla f(x^{(k+1)}), \nabla f(x^{(k+1)}))}{(\nabla f(x^{(k)}), \nabla f(x^{(k)}))} d^{(k)}$, după care se reia pasul 2.

Algoritmul Hestenes-Stiefel garantează că după un număr finit de iterații ce nu depășește ***n*** să obținem soluția exactă a problemei.
Dacă rezolvăm manual această problemă prin metoda gradientului obținem soluția optimă $x = \begin{pmatrix} -0.5 \\ 2 \end{pmatrix}$. Avem nevoie de acest rezultat pentru a verifica corectitudinea calculelor efectuate de către program.

Codul sursa :

```
//---------------------------------------------------------------------------
#include<iostream.h>
#include<math.h>
#include<conio.h>
//---------------------------------------------------------------------------
        double a,b;
//Parametri pentru fiecare varianta
        int nMax;
//Numarul maxim de iteratii
        double eps;
//Valoarea aproximatiei
//---------------------------------------------------------------------------

        double f(double X[])
{
//Returneaza valoarea functiei in punctul x,y
        return a*X[0]*X[0] + 2*X[0]*X[1] + b*X[1]*X[1] - 2*X[0] - 3*X[1];
}
                void grad(const double *x,double *y)
        {///Calculeaza valoarea gradientului

                        y[0] = 2*a*x[0] + 2*x[1] - 2;y[1] = 2*x[0] + 2*b*x[1] - 3;
        }


                        double scalar(const double *x,const double *y,int n)
                {///Inmultirea scalara a 2 vectori
                        double s=0;
                        for(int i=0;i<n;i++)s += x[i]*y[i];
                        return s;
                }
                int metConj(double *x)
                { //Metoda Hestenes - Stiefel
                        double x1[2];
                        double y[2],y1[2];// Valoarea Gradientului
                        double dir[2]; // Directia
                        double alfa,z[2];
                        double norma;
                        double A[2][2] = {{2*a,2},{2,2*b}}; //Matricea A
                        int n=0; //Numarul de iteratii
                        grad(x,y);
                        dir[0] = -y[0];
                        dir[1] = -y[1];
```

```
                do
        {
                double zmz[2];
                zmz[0] = scalar(A[0],dir,2);
                zmz[1] = scalar(A[1],dir,2);
                grad(x,y);
                alfa = -scalar(y,dir,2)/scalar(zmz,dir,2);
                x1[0] = x[0] + alfa*dir[0];
                x1[1] = x[1] + alfa*dir[1];
                grad(x1,y1);
                x[0] = x1[0];
                x[1] = x1[1];
                if(sqrt(y1[1]*y1[1]+y1[0]*y1[0])<eps)
        {
        n++;
        return n;
        }
                double temp[2],beta;
                beta = scalar(y1,y1,2)/scalar(y,y,2);
                dir[0] = -y1[0] + beta*dir[0];
                dir[1] = -y1[1] + beta*dir[1];
            n++;
            }
        while((n<nMax));
        }
//---------------------------------------------------------------------------
        int main()
{
        double x[2],x1[2];
        int n;
        textbackground(9);
        clrscr();

        cout<<"Se determina minimul global al functiei";
        cout<<"f(x,y)=(a*x*x)+2*x*y+(b*y*y)-2*x-3*y"<<endl;
        cout<<"Introduceti valorile paramerilor a,b:";
        cin>>a>>b;
        cout<<"Introduci valorile initiale:";
        cin>>x[0]>>x[1];
            x1[0] = x[0];
            x1[1] = x[1];
        cout<<"Introduceti aproximatia:";
        cin>>eps;
        cout<<"Introduceti numarul maxim de iteratii:";
        cin>>nMax;

        cout<<endl<<endl<<"Metoda Hestenes-Steifel:"<<endl;
            n=metConj(x1);
        cout<<endl<<"x[0]="<<x1[0]<<endl<<"x[1]="<<x1[1];
        cout<<endl<<"n="<<n;
        cout<<endl<<"f=="<<f(x1);
        cin.get();
```

```
        cin.get();
        return 0;
}
```

Exemplu de calcul:

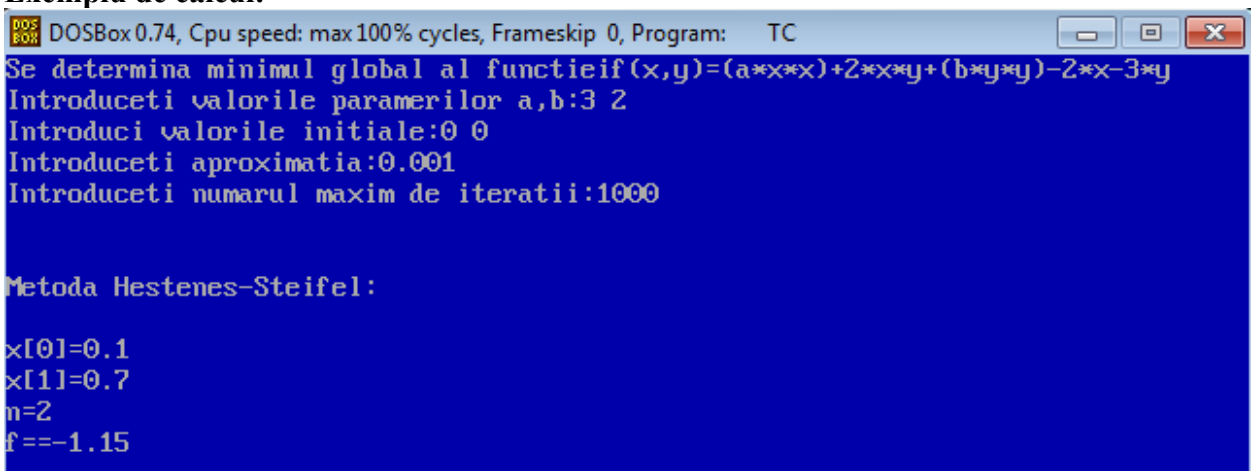

Fig1.valorile initiale (0 0)

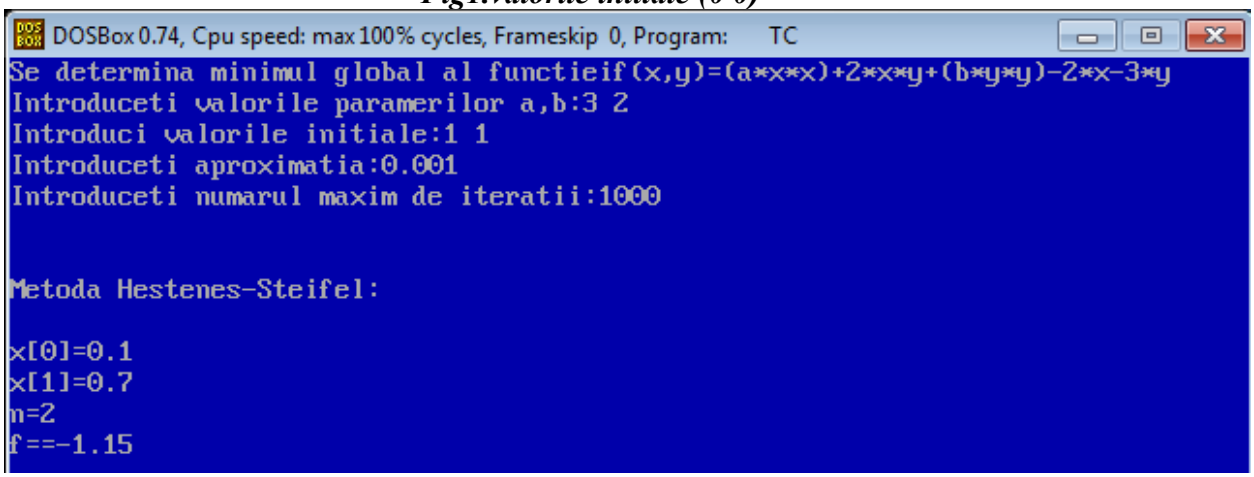

Fig2.valorile initiale (1 1)

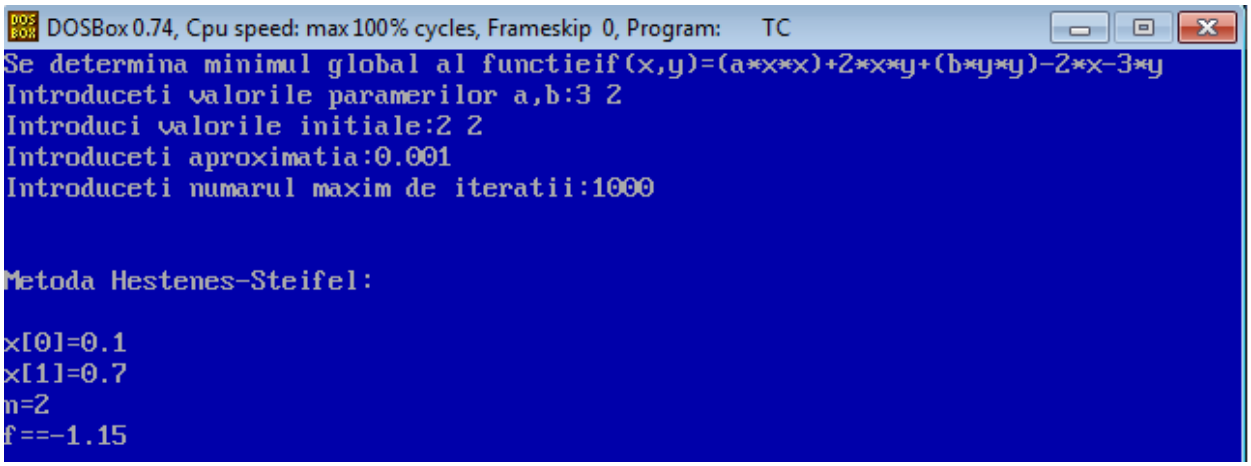

Fig3.valorile initiale (2 2)

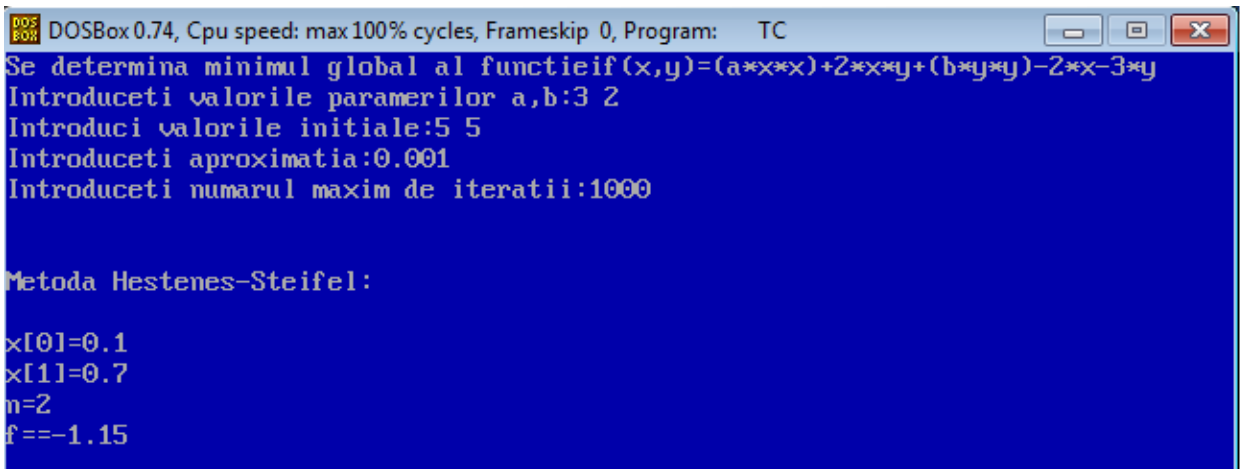

Fig4.valorile initiale (5 5)