Ministerul Educaţiei, Tineretului şi Sportului al Republicii Moldova
Universitate Tehnică a Moldovei
Catedra Automatica şi Tehnologii Informaţionale

Lucrarea de laborator Nr.1
Disciplina: Cercetari Operationale
Tema: Optimizarea neconditionata

A realizat: st. gr. TI-102

A verificat: lector asistent.

Chişinău 1012

Varianta 16

Obiectivele lucrarii:

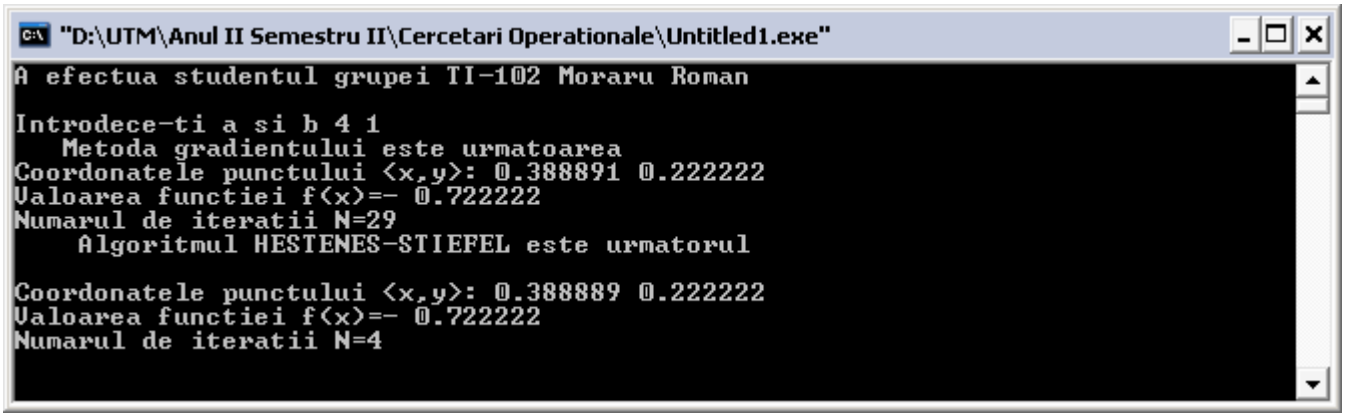
1. Studiul metodelor de optimizare neliniara fara restrictii.
2. Definirea si utilizarea unor proceduri pentru minimul functiilor cu ajutorul metodei gradientului si a metodei de directii conjugate cu o eroare $\varepsilon=10^{-5}$.
3. Analiza rezultatelor obtinute,inclusive stabilitatea tipului minimului:local sau global.
4. Sa se compare rezultatele luind in consideratie numarul de iteratii evaluarilor pentru functie si gradientului ei.

Listingul programului :

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
class OPTIMIZARE
{
 public:
            float F(float x, float y);
            void INPUT(void);
            void GRADIENT(void);
            void METODA_GRADIENTULUI(void);
            void ALGORITMUL_HESTENES_STIEFEL(void);

 private:
            float a,b,ALFA,GAMA,DELTA,EPS,X[2],Z[2],G[2];
};

  float OPTIMIZARE::F(float x, float y)
            {
             return a*x*x+2*x*y+b*y*y-2*x-3*y;
            }

  void OPTIMIZARE::INPUT(void)
            {
             cout<<"Introduce-ti a si b: "; cin>>a>>b;
            }

  void OPTIMIZARE::GRADIENT(void)
            {
             G[0]=2*a*X[0]+2*X[1]-2;
             G[1]=2*X[0]+2*b*X[1]-3;
            }

  void OPTIMIZARE::METODA_GRADIENTULUI(void)
    {
     int N=0;
     float Ak;
     ALFA=0.1; GAMA=0.1; DELTA=0.1;
     X[0]=1; X[1]=1; EPS=0.00001;
     GRADIENT();
     while(sqrt(G[0]*G[0]+G[1]*G[1])>=EPS)
             {
              ++N;
              ALFA=0.1;
              Z[0]=X[0]-ALFA*G[0];
              Z[1]=X[1]-ALFA*G[1];
              while((F(Z[0],Z[1])-F(X[0],X[1]))>-DELTA*ALFA*(G[0]*G[0]+G[1]*G[1]))
                {
                 ALFA*=GAMA;
                 Z[0]=X[0]-ALFA*G[0];
                 Z[1]=X[1]-ALFA*G[1];
                }
              Ak=ALFA;
              X[0]=X[0]-Ak*G[0];
              X[1]=X[1]-Ak*G[1];
              GRADIENT();
             }
     cout<<" Metoda Gradientului"
              <<" este urmatoarea\n";
     cout<<"coordonatele punctului (x,y): "<<X[0]<<" "<<X[1]<<endl;
     cout<<"valoarea functiei f(x)="<<F(X[0],X[1])<<endl;
```

```
    cout<<"numarul de iteratii N="<<N<<endl;
   }

  void OPTIMIZARE::ALGORITMUL_HESTENES_STIEFEL(void)
    {
    int N=0,k=0;
    float Ak,G1[2]={0},d[2],X1[2];
    X[0]=1; X[1]=1;
    GRADIENT();
    while((G[0]!=0&&G[1]!=0))
    {
          ++N;
          if(k++==0){ d[0]=-G[0]; d[1]=-G[1]; }
          else {
               d[0]=d[0]*(G[0]*G[0]+G[1]*G[1])/(G1[0]*G1[0]+G1[1]*G1[1])-G[0];
               d[1]=d[1]*(G[0]*G[0]+G[1]*G[1])/(G1[0]*G1[0]+G1[1]*G1[1])-G[1];
              };
          Ak=-(G[0]*d[0]+G[1]*d[1])/(2*a*d[0]*d[0]+4*d[0]*d[1]+2*b*d[1]*d[1]);
          X1[0]=X[0]; X1[1]=X[1]; G1[0]=G[0]; G1[1]=G[1];
          X[0]=X[0]+Ak*d[0];
          X[1]=X[1]+Ak*d[1];
          if(X[0]==X1[0]&&X[1]==X1[1])break;
          GRADIENT();
    }
    cout<<"\n ALGORITMUL HESTENES-STIEFEL"
          <<" este urmatorul\n";
    cout<<"coordonatele punctului (x,y): "<<X[0]<<" "<<X[1]<<endl;
    cout<<"valoarea functiei f(x)="<<F(X[0],X[1])<<endl;
    cout<<"numarul de iteratii N="<<N<<endl;
    }
void main()
{
 clrscr();
 OPTIMIZARE ob;
 ob.INPUT();
 ob.METODA_GRADIENTULUI();
 ob.ALGORITMUL_HESTENES_STIEFEL();
 getch();
}
```

Rezultatele executiei:



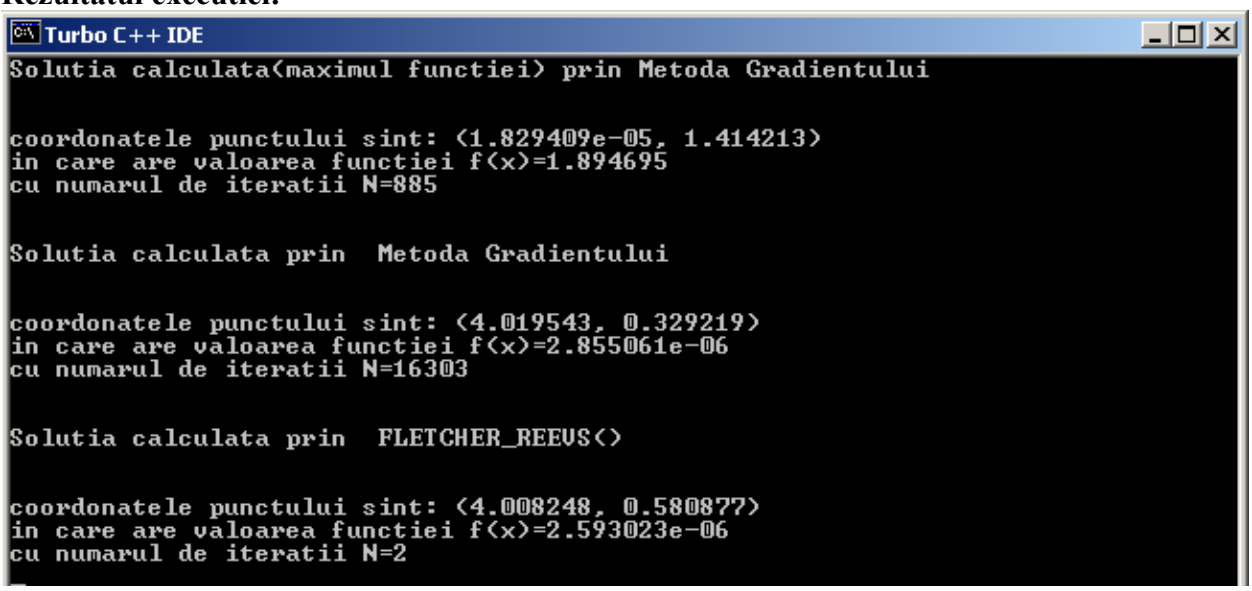
Ecuatie exponential

Listingul programului:

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
class REZOLVARE
{
 public:
        float F(float x, float y)
                { return ( exp(-x*x-y*y)*(4*x*x + 1*y*y ) );}
        void GR()
                {
                d[0]=2*x[0]*exp(-x[0]*x[0]-x[1]*x[1])*(2-x[0]*x[0]-2*x[1]*x[1]);
                d[1]=2*x[1]*exp(-x[0]*x[0]-x[1]*x[1])*(4-x[0]*x[0]-2*x[1]*x[1]);
                }
        float XZ(float x, float g,float a)
                        {return x-a*g;}
        void METODA_GRADIENTULUI(void)
         {              int N=0;
          alfa=0.1; gama=0.1; delta=0.1;
          x[0]=1; x[1]=1; eps=0.00001;
          GR();
          while(sqrt(d[0]*d[0]+d[1]*d[1])>=eps)
         {
         ++N;
         alfa=0.1;
         z[0]=XZ(x[0],d[0],alfa);
         z[1]=XZ(x[1],d[1],alfa);
         while((F(z[0],z[1])-F(x[0],x[1]))>-delta*alfa*(d[0]*d[0]+d[1]*d[1]))
                {
                 alfa*=gama;
                  z[0]=XZ(x[0],d[0],alfa);
                  z[1]=XZ(x[1],d[1],alfa);
                }
                x[0]=XZ(x[0],d[0],alfa);
                x[1]=XZ(x[1],d[1],alfa);
         GR();
        }
          cprintf("\n\nSolutia calculata prin");
          cprintf("  Metoda Gradientului\n\n");
          cout<<"\ncoordonatele punctului sint: ("<<x[0]<<", "<<x[1]<<")"<<endl;
          cout<<"in care are valoarea functiei f(x)="<<F(x[0],x[1])<<endl;
          cout<<"cu numarul de iteratii N="<<N<<endl;
         }
 void FLETCHER_REEVS()
 {
  float d1,d2,beta=0,g1,g2;
        int k=0,n=10;
        alfa=0.1;
        x[0]=1; x[1]=1;
        GR();
        d1=-d[0]; d2=-d[1]; g1=d[0]; g2=d[1];
        while(sqrt(d1*d1+d2*d2)>=eps)
          {
                if(k>=1)
                 {
                  if(k%n==0)beta=0;
                  else
                        {                                       GR();
                         beta=(d[0]*d[0]+d[1]*d[1])/(g1*g1+g2*g2);
                        }
                  d1=-d[0]+beta*d1;
                  d2=-d[1]+beta*d2;
                 }
                x[0]=x[0]+alfa*d1;
                x[1]=x[1]+alfa*d2;
                ++k;
          }
          cprintf("\n\nSolutia calculata prin");
          cprintf("  FLETCHER_REEVS()\n\n");
```

```
            cout<<"\ncoordonatele punctului sint: ("<<x[0]<<", "<<x[1]<<")"<<endl;
            cout<<"in care are valoarea functiei f(x)="<<F(x[0],x[1])<<endl;
            cout<<"cu numarul de iteratii N="<<k<<endl;
 }
          void MAX(void)
           {             int N=0;
            alfa=0.1; gama=0.1; delta=0.1;
            x[0]=1; x[1]=1; eps=0.00001;
            GR();
            while(sqrt(d[0]*d[0]+d[1]*d[1])>=eps)
           {            ++N;  alfa=0.1;
           z[0]=XZ(x[0],d[0],alfa);
           z[1]=XZ(x[1],d[1],alfa);
           while((F(z[0],z[1])-F(x[0],x[1]))>-delta*alfa*(d[0]*d[0]+d[1]*d[1]))
                      {
                       alfa*=gama;
                        z[0]=XZ(x[0],d[0],alfa);
                        z[1]=XZ(x[1],d[1],alfa);
                      }
                     x[0]=XZ(x[0],-d[0],alfa);
                     x[1]=XZ(x[1],-d[1],alfa);
            GR();
           }
            cprintf("Solutia calculata(maximul functiei) prin");
            cprintf(" Metoda Gradientului\n\n");
            cout<<"\ncoordonatele punctului sint: ("<<x[0]<<", "<<x[1]<<")"<<endl;
            cout<<"in care are valoarea functiei f(x)="<<F(x[0],x[1])<<endl;
            cout<<"cu numarul de iteratii N="<<N<<endl;
           }
 private:
          float alfa,gama,delta,eps,x[2],z[2],d[2];
};
void main()
{
 clrscr();
 REZOLVARE ob;
 ob.MAX();
 ob.METODA_GRADIENTULUI();
 ob.FLETCHER_REEVS();
 getch();
}
```

Rezultatul executiei:



Concluzii: Efectuind aceasta lucrare de laborator am luat cunostintra cu optimizarea neconditionata.Am invata a folosi metoda gradientului si metoda directiei conjugate.