|  |  |
| --- | --- |
| Optymalizacja funkcji jednej zmiennej metodami bezgradientowymi | |
| Autorzy | Kozłowski Bartosz, Kopeć Jakub, Kobyłecki Emil |
| Data wysłania | 3.11.2020 |

1. Parametry algorytmu metody ekspansji:  
 -x0: losowa liczba z przedziału <-100;100>,  
 -liczba d: 1,  
 -współczynniki ekspansji: 2 , 1.5, 1.1,  
 -maksymalna ilość iteracji: 1000  
**Kod źródłowy metody ekspansji:**

**double\* expansion(double x0, double d, double alfa, int Nmax, matrix O)**

**{**

**double\* p = new double[2];**

**solution X0(x0), X1(matrix(x0 + d));**

**X0.fit\_fun(matrix(x0));**

**X1.fit\_fun(matrix(x0 + d));**

**if (X0.y == X1.y)//PRÓBA, przypadek I**

**{**

**p[0] = x0;**

**p[1] = x0 + d;**

**return p;**

**}**

**if (X0.y < X1.y) //PRÓBA, przypadek III /\*f(x0)<f(x0+d)\*/**

**{**

**d \*= -1;**

**X1.x = X0.x + d;//PRÓBA**

**X1.fit\_fun(X1.x);**

**if (X0.x <= X1.x)/\*f(x0)<f(x0+d)\*/**

**{**

**p[0] = X1.x(0);**

**p[1] = X0.x(0);**

**return p;**

**}**

**}**

**solution X2;//(X0.x); //36.05**

**int i = 1;**

**while (true)**

**{**

**X2.x = x0 + pow(alfa, i) \* d;**

**X2.fit\_fun();**

**if (X2.y >= X1.y || solution::f\_calls > Nmax)break;**

**X0 = X1;//PRÓBA**

**X1 = X2;//PRÓBA**

**++i;**

**}**

**if (d > 0)**

**{**

**p[0] = X0.x(0);**

**p[1] = X2.x(0);**

**}**

**else**

**{**

**p[0] = X2.x(0);**

**p[1] = X0.x(0);**

**}**

**return p;**

**}**

2. Parametry algorytmu metody Fibonacciego:  
 -przedział <a,b>: wartości wczytywane z plików, do których funkcja expansion wpisywała swoje wyniki,  
 -epsilon: 0.01

**Kod źródłowy metody Fibonacciego:**

**solution fib(double a, double b, double epsilon, matrix O)**

**{**

**int n = log(sqrt(5) \* 100) / log((1 + sqrt(5)) / 2) + 1; //PRÓBNE**

**double\* F = new double[n] {1, 1};**

**for (int i = 2; i < n; ++i) F[i] = F[i - 2] + F[i - 1];**

**solution A(a), B(b), C, D;**

**C.x = B.x(0, 0) - (F[n - 2] / F[n - 1]) \* (B.x(0, 0) - A.x(0, 0));**

**D.x = A.x + B.x - C.x; //PRÓBA**

**C.fit\_fun();**

**D.fit\_fun();**

**for (int i = 0; i <= n - 3; ++i)**

**{**

**if (C.y(0, 0) < D.y(0, 0)) /\*f(C.x)>f(D.x)\*/**

**B.x = D.x; //PRÓBA**

**else**

**A.x = C.x; //PRÓBA**

**C.x = B.x - (F[n - 3 - i] / F[n - 2 - i]) \* (B.x - A.x);//PRÓBA**

**D.x = A.x + B.x - C.x;//PRÓBA**

**C.fit\_fun();**

**D.fit\_fun();**

**//cout << "iteracja "<<i<<endl;**

**//cout << "b-a = " << B.x - A.x << endl;**

**}**

**cout<<"Wywolania FIB: "<<C.f\_calls<<endl;**

**return C;**

**}**

2. Parametry algorytmu metody opartej na interpolacji Lagrange’a:  
 -przedział <a;b>: wartości wczytywane z plików, do których funkcja expansion wpisywała swoje wyniki,  
 -epsilon: 0.1,   
 -gamma: 0.0001,  
 -maksymalna ilość iteracji: 1000  
**Kod źródłowy metody Lagrange’a:**

solution lag(double a, double b, double epsilon, double gamma, int Nmax, matrix O)

{

solution A(a), B(b), C((a + b) / 2), D(0);

A.fit\_fun(); //fit\_fun() oblicza funkcje w celu

B.fit\_fun();

C.fit\_fun();

double l, m;

int i = 1;

l = A.y(0) \* (pow(B.x(0), 2) - pow(C.x(0), 2)) + B.y(0) \* (pow(C.x(0), 2) - pow(A.x(0), 2))

+ C.y(0) \* (pow(A.x(0), 2) - pow(B.x(0), 2));

m = A.y(0) \* (B.x(0) - C.x(0)) + B.y(0) \* (C.x(0) - A.x(0)) + C.y(0) \* (A.x(0) - B.x(0));

if (m <= 0)

{

C.x = NAN;

C.y = NAN;

cout << "333";

return C;

}

D.x = 0.5 \* l / m;//min funkcji w przediale (a,b)

D.fit\_fun();

while (true)

{

//PRÓBA

if (A.x(0) <= C.x(0) && C.x(0) <= D.x(0))

{

//cout << "iteracja " << i << endl;

//cout << "b-a = " << B.x - A.x << endl;

i++;

if (D.y(0) < C.y(0))

{

A = C;

C = D;

}

else

{

B = D;

}

}

else if (A.x(0) <= D.x(0) && D.x(0) < C.x(0))

{

//cout << "iteracja " << i << endl;

//cout << "b-a = " << B.x - A.x << endl;

i++;

if (D.y(0) < C.y(0))

{

B = C;

C = D;

}

else

{

A = D;

}

//cout << "iteracja " << i << endl;

//cout << "b-a = " << B.x - A.x << endl;

}

else

{

C.x = NAN;

C.y = NAN;

cout << "111";

return C;

}

l = A.y(0) \* (pow(B.x(0), 2) - pow(C.x(0), 2)) + B.y(0) \* (pow(C.x(0), 2) - pow(A.x(0), 2))

+ C.y(0) \* (pow(A.x(0), 2) - pow(B.x(0), 2));

m = A.y(0) \* (B.x(0) - C.x(0)) + B.y(0) \* (C.x(0) - A.x(0)) + C.y(0) \* (A.x(0) - B.x(0));

if (m <= 0)

{

C.x = NAN;

C.y = NAN;

cout << "222";

return C;

}

D.x = 0.5 \* l / m;//min funkcji w przediale (a,b)

D.fit\_fun();

if (Nmax <= D.f\_calls - 3 || fabs(A.x(0) - B.x(0)) < epsilon || fabs(D.x(0) - C.x(0)) < gamma)

{

C.fit\_fun();

//cout << "c.x = " << C.x(0) << ", d.x = " << D.x(0) << endl;

cout<<"Wywolania LAG: "<<C.f\_calls<<endl;

return C;

}

}

}

**Problem rzeczywisty.**

Argumenty w wywołaniu metody ekspansji dla problemu rzeczywistego:  
 -x0: losowa liczba z przedziału <-100;100>,  
 -liczba d: 1,  
 -współczynnik ekspansji: 1.1,  
 -maksymalna ilość iteracji: 1000

Argumenty w wywołaniu metody Fibonacciego dla problemu rzeczywistego:  
 -przedział <a;b>: wartości wczytywane z plików, do których funkcja expansion wpisywała swoje wyniki,  
 -epsilon: 0.01,

Argumenty w wywołaniu metody Lagrange’a dla problemu rzeczywistego:  
 -przedział <a;b>: wartości wczytywane z plików, do których funkcja expansion wpisywała swoje wyniki,  
 -epsilon: 0.01,   
 -gamma: 0.0001,  
 -maksymalna ilość iteracji: 1000

**Dyskusja wyników oraz wnioski.**Dla testowej funkcji celu różnica pomiędzy uśrednionymi współrzędnymi minimów lokalnych oraz globalnych jest marginalna. W większości określonych przedziałów <a;b> zarówno metoda Fibonacciego jak i metoda Lagrange’a zwracają taki sam typ minimum funkcji celu(nie licząc przypadków, gdy druga metoda zwraca błędne dane - takie przypadki nie zostały wliczone do średnich w arkuszu kalkulacyjnym).   
Na podstawie wykresów w arkuszu kalkulacyjnym jak i na różnicy średniej ilości wywołań funkcji celu dla obu metod, można wywnioskować, że metoda Lagrange’a potrzebuje mniejszej ilości kroków aby znaleźć minimum funkcji.  
Rozwiązanie zadania rzeczywistego dwiema metodami przyniosło dwa, znacznie różniące się wyniki. Oznacza to, że implementacja jednej z nich nie jest przystosowana do problemu rzeczywistego. Dodatkowo różnice w rezultatach obu metod mają swoje odzwierciedlenie w wartościach komórek w symulacji.

**Funkcja main:**

int main()

{

try

{

#if LAB\_NO==1

double t0 = 0, dt = 0.1, tend = 50;

matrix Y0 = matrix(new double[2]{ 1,0 }, 2);

matrix\* Y = solve\_ode(t0, dt, tend, Y0);

ofstream S("tout.csv");

S << Y[0];

S.close();

S.open("yout.csv");

S << Y[1];

S.close();

#elif LAB\_NO==2

/\*

solution C;

C=fib(0.0001,0.01,0.01);

cout<<"Fib: x= "<<C.x<<" y= "<<C.y<<endl; //wielkość dziury

solution::clear\_calls();

\*/

solution C2=lag(0.0001,0.01, 0.01, 0.0001, 1000);

cout<<"Lag: x= "<<C2.x<<" y= "<<C2.y<<endl; //wielkość dziury

solution::clear\_calls();

/\*

matrix F=matrix(new double[1]{0,0062875},1);

matrix Y0=matrix(new double[3]{5,1,10},3);

matrix \*Y=solve\_ode(0,1,1000,Y0,F);

ofstream S("czasout.csv");

S << Y[0];

S.close();

S.open("rozwout.csv");

S << Y[1];

S.close();

\*/

/\*

ofstream plik1("p0.txt");

ofstream plik2("p1.txt");

ofstream plik3("fcalls.txt");

ofstream plik4("start.txt");

int start;

for (int i = 0; i < 100; i++) {

start = rand() % 200 - 100;

double\* p = expansion(start, 1, 1.1, 1000);

plik1 << p[0] << endl;

plik2 << p[1] << endl;

plik3 << solution::f\_calls<<endl;

plik4 << start << endl;

solution::clear\_calls();

}\*/

/\*

ifstream plik1("p0.txt");

ifstream plik2("p1.txt");

ofstream plik3("x.txt");

ofstream plik4("y.txt");

ofstream plik5("fcalls.txt");

solution C;

double X, Y;

for (int i = 0; i < 100; i++) {

plik1 >> X;

plik2 >> Y;

C = fib(X, Y, 0.01);

plik3 << C.x(0) << endl;

plik4 << C.y(0) << endl;

plik5 << solution::f\_calls << endl;

solution::clear\_calls();

}\*/

//cout << "Fibonacci: x=" << C.x(0) << " y=" << C.y(0) << endl;

/\*

solution C2;

C2 = lag(p[0], p[1], 0.01, 0.0001, 1000);

cout << "Lagrange: x=" << C2.x(0) << " y=" << C2.y(0) << endl;

solution::clear\_calls();

\*/

/\*

solution C2;

double X, Y;

ifstream plik1("p0.txt");

ifstream plik2("p1.txt");

ofstream plik3("x.txt");

ofstream plik4("y.txt");

ofstream plik5("fcalls.txt");

/\*for (int i = 0; i < 100; i++) {

plik1 >> X;

plik2 >> Y;

C2 = lag(X, Y, 0.01, 0.0001, 1000);

plik3 << C2.x(0) << endl;

plik4 << C2.y(0) << endl;

plik5 << solution::f\_calls << endl;

solution::clear\_calls();

}\*/

/\*

C2 = lag(-100, 100, 0.01, 0.0001, 1000);

plik3 << C2.x(0) << endl;

plik4 << C2.y(0) << endl;

plik5 << solution::f\_calls << endl;

/\*solution::clear\_calls();

solution C2;

C2 = lag(-17, 31, 0.01, 0.0001, 1000);

cout << "x = " <<C2.x << ", y = " << C2.y << "fcalls = " << solution::f\_calls << endl;

\*/

#endif

cout << "LAB NUMBER " << LAB\_NO << endl;

cout << "LAB PART " << LAB\_PART << endl << endl;

}

catch (char\* EX\_INFO)

{

cout << EX\_INFO << endl;

}

//system("pause");

return 0;

}

**Funkcja fit\_fun:**

void solution::fit\_fun(matrix O)

{

#if LAB\_NO==2 && LAB\_PART==1

y = -cos(0.1 \* x(0)) \* pow(exp(1), (-pow((0.1 \* x(0) - 2 \* 3.14), 2))) + 0.002 \* pow(0.1 \* x(0), 2);

++f\_calls;

#elif LAB\_NO==2 && LAB\_PART==2

matrix Y0=matrix(new double[3]{5,1,10},3);

matrix \*Y=solve\_ode(0,1,1000,Y0,x);

int \*w=get\_size(Y[1]);

double max=Y[1](0,2);

for (int i=1;i<w[0];i++)

{

if(max<Y[1](i,2))

{

max=Y[1](i,2);

}

}

y=fabs(50-max);

}

**Funkcja diff:**

matrix diff(double t, const matrix &Y, matrix P) //return pochodne of all variables

{

#if LAB\_NO==1 //conditional compilation used for exrcs. 1

double m = 5,d=1.5,k=1,F=0;

matrix dY(Y);

dY(0) = Y(1);

dY(1) = (F - d\*Y(1) - k\*Y(0)) / m;

return dY;

//#else

//matrix dY;

//return dY;

//#endif

#elif LAB\_NO==2 //50:37

static int i=1;

matrix dY(Y);

double a=0.98,b=0.63,g=9.81,PA=1,PB=1,Fin=0.01,DB=0.00365665,Tin=10,TA=90;

double DA=P(0);

//static V=1; //ilosc wody w zbiorniku

//static T=10;//temperatura w zbiorniku

double VAout;

if(Y(0)<=0)

{

VAout=0;

}

else

{

VAout=-a\*b\*DA\*sqrt(2\*g\*Y(0)/PA);

}

double VBout;

if(Y(1)<=0)

{

VBout=0;

}

else

{

VBout=-a\*b\*DB\*sqrt(2\*g\*Y(1)/PB);

}

if(i<=1000)

{

cout << Y(0)<<endl;

i++;

}

dY(0)=VAout;

dY(1)=Fin+VBout-VAout;

dY(2)=-VAout/Y(1)\*(TA-Y(2))+Fin/Y(1)\*(Tin-Y(2));

//V+=dY(1); //Zmiana ilosci wody w zbiorniku

//T+=dY(2); //Zmiana temperatury w zbiorniku

return dY;

#endif

}