|  |  |
| --- | --- |
| Optymalizacja funkcji wielu zmiennej metodami bezgradientowymi | |
| Autorzy | Kozłowski Bartosz, Kopeć Jakub, Kobyłecki Emil |
| Data wysłania | 17.11.2020 |

**1. Parametry algorytmu metody Hooke’a-Jeevesa:**

**dla testowej funkcji celu:**

-losowy wektor startowy z przedziału [<-1;1>;<1;1>],  
 -początkowe długości kroku: 1, 0.1, 0.43,  
 -współczynniki zmniejszania długości kroku alfa: 0.5,

-dokładność obliczeń: 0.0001,  
 -maksymalna ilość wywołań funkcji celu: 1000

**dla problemu rzeczywistego:**

-wektor startowy [4,4],  
 -początkowa długość kroku: 1,  
 -współczynniki zmniejszania długości kroku alfa: 0.9,

-dokładność obliczeń: 0.00001,  
 -maksymalna ilość wywołań funkcji celu: 1000

**2. Parametry algorytmu metody Rosenbrocka:**

**dla testowej funkcji celu:**

-losowy wektor startowy z przedziału [<-1;1>;<1;1>],

-początkowe długości kroku s: 1, 0.1, 0.43,

-macierz kroku początkowego [s,s],  
 -współczynniki zwiększania długości kroku alfa: 2,  
 -współczynnik zmniejszania długości kroku beta: 0.5,

-dokładność obliczeń: 0.0001,  
 -maksymalna ilość wywołań funkcji celu: 1000

**dla problemu rzeczywistego:**

-wektor startowy [4,4],  
 -początkowa długość kroku: 1.5,  
 -współczynniki zmniejszania długości kroku alfa: 0.9,

-współczynnik zmniejszania długości kroku beta: 0.4,

-dokładność obliczeń: 0.00001,  
 -maksymalna ilość wywołań funkcji celu: 1000

**Dyskusja wyników oraz wnioski:**Dla testowej funkcji celu, ilość znalezionych minimów globalnych za pomocą obu metod nieznacznie się od siebie różni. Z tego powodu, nie jest możliwe jednoznaczne stwierdzenie, która metoda jest lepsza. Dodatkowo ilość znalezionych minimów jest uzależniona od punktów początkowych poszukiwań, które były wyznaczane za pomocą generatora liczb pseudolosowych. Liczby wywołań funkcji celu dla obu metod także niewiele się od siebie różnią. Dla jednego konkretnego przypadku, metoda Hooke’a-Jeevesa znalazła minimum globalne przy mniejszej liczbie iteracji niż przy użyciu metody Rosenbrocka. Wynik tej próby jest jednak zależny od pary punktów startowych dla obu metod.  
  
W tabeli trzeciej osiągnięte wartości obiema metodami kolejny raz są do siebie zbliżone. Jedyną zauważalną różnicą jest ilość wywołań funkcji celu.  
  
W symulacji wartości położenia ramienia oraz prędkości ramienia są bardzo do siebie zbliżone(różnica ta jest mniejsza niż przyjęta dokładność obliczeń).

**Funkcja HJ oraz HJ\_trial:**

**solution HJ(matrix x0, double s, double alfa, double epsilon, int Nmax, matrix O)**

**{**

**solution XB, XB\_old, X;**

**XB.x = x0;**

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

**while (true)**

**{**

**//KROK PRÓBNY**

**X = HJ\_trial(XB, s);**

**if (X.y < XB.y)**

**{**

**while (true)**

**{**

**XB\_old = XB;**

**XB = X;**

**X.x = 2.0 \*XB.x - XB\_old.x;**

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

**X = HJ\_trial(X, s);**

**if (X.y >= XB.y)**

**break;**

**if (Nmax < solution::f\_calls)**

**return XB;**

**}**

**}**

**else**

**s \*= alfa;**

**if (s < epsilon || Nmax < solution::f\_calls)**

**{**

**return XB;**

**}**

**}**

**}**

**//KROK PRÓBNY DLA HJ**

**solution HJ\_trial(solution XB, double s, matrix O)**

**{**

**int \*n = get\_size(XB.x);**

**matrix D (n[0], n[0]);**

**for (int i = 0; i < n[0]; i++) {**

**D(i, i) = 1;**

**}**

**solution X;**

**for (int i = 0; i < n[0]; ++i)**

**{**

**X.x = XB.x + s \* D[i];**

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

**if (X.y < XB.y)**

**XB = X;**

**else**

**{**

**X.x = XB.x - s \* D[i];**

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

**if (X.y < XB.y)**

**XB = X;**

**}**

**}**

**return XB;**

**}**

**Funkcja Rosen:**

**solution Rosen(matrix x0, matrix s0, double alfa, double beta, double epsilon, int Nmax, matrix O)**

**{**

**int \*n = get\_size(x0);**

**matrix l(n[0], 1), p(n[0], 1), s(s0);**

**matrix D(n[0], n[0]);**

**for (int i = 0; i < n[0]; i++) {**

**D(i, i) = 1;**

**}**

**solution X, Xt;**

**X.x = x0;**

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

**while (true)**

**{**

**for (int i = 0; i < n[0]; ++i)**

**{**

**Xt.x = X.x + s(i)\*D[i];**

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

**if (Xt.y < X.y)**

**{**

**X = Xt;**

**l(i) += s(i);**

**s(i) \*= alfa;**

**}**

**else**

**{**

**p(i)++;**

**s(i) \*= -1 \* beta;**

**}**

**}**

**bool change = true;**

**for (int i = 0; i < n[0]; ++i)**

**if (l(i) == 0 || p(i) == 0)**

**{**

**change = false;**

**break;**

**}**

**if (change)**

**{**

**matrix Q(n[0], n[0]), v(n[0], 1);**

**for (int i = 0; i < n[0]; ++i)**

**for (int j = 0; j <= i; ++j)**

**Q(i, j) = l(i);**

**Q = D \* Q;**

**v = Q[0] / norm(Q[0]);**

**D = set\_col(D, v, 0);**

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

**{**

**matrix temp(n[0], 1);**

**for (int j = 0; j < i; ++j)**

**temp = temp + trans(Q[i])\*D[j] \* D[j];**

**v = (Q[i] - temp) / norm(Q[i] - temp);**

**D = set\_col(D, v, i);**

**}**

**s = s0;**

**l = matrix(n[0], 1);**

**p = matrix(n[0], 1);**

**}**

**double max\_s = abs(s(0));**

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

**if (max\_s < abs(s(i)))**

**max\_s = abs(s(i));**

**if (max\_s < epsilon || solution::f\_calls > Nmax)**

**{**

**return X;**

**}**

**}**

**}**

**Funkcja main:**

**#elif LAB\_NO == 3 && LAB\_PART == 2**

**srand((unsigned)time(0));**

**double r1=(double)(rand()%101)/100;**

**double r2=(double)(rand()%101)/100;**

**if((rand()%2)==0)r1\*=(-1);**

**if((rand()%2)==0)r2\*=(-1);**

**//HJ**

**matrix x0(2, 1);**

**double s = 0.43;**

**double alfa = 0.9;**

**double epsilon = 0.00001;**

**int Nmax = 10000;**

**solution wynik,test;**

**for (int i = 0; i < 1 ;i++)**

**{**

**r1=(double)(rand()%101)/100;**

**r2=(double)(rand()%101)/100;**

**if((rand()%2)==0)r1\*=(-1);**

**if((rand()%2)==0)r2\*=(-1);**

**x0(0) = r1;//unif(re);**

**x0(1) = r2;//unif(re);**

**test.x=x0;**

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

**//Wartości początkowe**

**cout <<x0(0) << " " << x0(1)<<" ";//<<test.y(0)<<endl;**

**wynik = HJ(x0, s, alfa, epsilon, Nmax);**

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

**//cout << "HJ:"<<endl <<"x1="<< wynik.x(0)<<" x2="<<wynik.x(1) <<" y="<<wynik.y(0)<< endl <<"f\_calls = "<<solution::f\_calls<<endl;**

**//HJ**

**cout <<wynik.x(0)<<" "<<wynik.x(1) <<" "<<wynik.y(0)<<" "<<solution::f\_calls;//<<endl;**

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

**matrix s0(2,1);**

**s0(0)=s;**

**s0(1)=s;**

**alfa=1.2;**

**solution wynik\_2=Rosen(x0,s0,alfa,alfa/2,epsilon,Nmax);**

**wynik\_2.fit\_fun();**

**//cout << "Rosen:"<<endl <<"x1="<< wynik\_2.x(0)<<" x2="<<wynik\_2.x(1) <<" y="<<wynik\_2.y(0)<< endl <<"f\_calls = "<<solution::f\_calls<<endl;**

**cout <<" "<< wynik\_2.x(0)<<" "<<wynik\_2.x(1) <<" "<<wynik\_2.y(0)<<" "<<solution::f\_calls<<endl;**

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

**}**

**#elif LAB\_NO == 3 && LAB\_PART == 2**

**matrix x0(2, 1);**

**double s = 1;**

**double alfa = 0.9;**

**double epsilon = 0.00001;**

**int Nmax = 1000;**

**solution wynik,test;**

**x0(0) = 4;**

**x0(1) = 4;**

**wynik = HJ(x0, s, alfa, epsilon, Nmax);**

**cout <<wynik;**

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

**matrix s0(2,1);**

**s0(0)=s;**

**s0(1)=s;**

**alfa=1.5;**

**solution wynik\_2=Rosen(x0,s0,alfa,0.4,epsilon,Nmax);**

**cout <<" "<< wynik\_2;**

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

**#endif**

**Funkcja fit\_fun dla problemu rzeczywistego:**

**void solution::fit\_fun(matrix O)**

**{**

**#if LAB\_NO==3 && LAB\_PART==2**

**double alfa\_referencyjne=3.14,omega\_referencyjne=0;**

**matrix Y0(2,1);**

**matrix \*Y=solve\_ode(0,0.1,100,Y0,x);//x to wektor z k1 & k2**

**for(int i=0;i<Y[0].n;i++)**

**{**

**y=y+10\*pow(alfa\_referencyjne - Y[1](i, 0), 2) + pow(omega\_referencyjne - Y[1](i, 1), 2) + pow(x(0)\*(alfa\_referencyjne - Y[1](i, 0)) + x(1)\*(omega\_referencyjne - Y[1](i, 1)), 2); //pow((alfa\_referencyjne-Y[1](i,0)),2)+pow((omega\_referencyjne-Y[1](i,1)),2)+pow(x(0)\*(pow((alfa\_referencyjne-Y[1](i,0)),2)+x(1)\*(pow((omega\_referencyjne-Y[1](i,0)),2))),2);**

**}**

**y=y\*0.1;**

**++f\_calls;**

**#endif**

**}**

**Funkcja diff:**

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

{

...

#elif LAB\_NO==3 && LAB\_PART==2

double masa\_ramie=1.,masa\_ciezarka=10.,dlugosc\_ramienia=0.5,b=0.5,alfa\_referencyjne=3.14,omega\_referencyjne=0,I=((1/3)\*masa\_ramie\*pow(dlugosc\_ramienia,2)+masa\_ciezarka\*pow(dlugosc\_ramienia,2));

double k1=P(0),k2=P(1); //w P przesyałmy k1 oraz k2

double M=k1\*(alfa\_referencyjne-Y(0))+k2\*(omega\_referencyjne-Y(1));

matrix dY(2,1);

dY(0)=Y(1);

dY(1)=(M-b\*Y(1))/I;

cout<<Y(0)<<" "<<Y(1)<<endl; //PRZEMIESZCZENIE I PRĘDKOŚĆ

return dY;

#endif

}