

METODY NUMERYCZNE

ZADANIE 18

Artur Guniewicz

18*. Startując z kilku losowo wybranych punktów początkowych, spróbuj numerycznie znaleźć minimum czterowymiarowej funkcji Rosenbrocka

$$f(x_1, x_2, x_3, x_4) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2 + 100(x_3 - x_2^2)^2 + 100(x_4 - x_3^2)^2. \quad (15)$$

Metoda: Metoda Gradientów

Dwuwymiarowa funkcja Rosenbrocka jest używana do przedstawiania zachowań algorytmów optymalizacji. Jej minimum globalne znajduje się w punkcie $(x,y) = (1,1)$, a wartość funkcji w tym punkcie wynosi $f(x,y) = 0$.

Wzór tej funkcji:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

A jej wielowymiarowym rozwinięciem jest:

$$f(x) = \sum_{i=1}^{N-1} [(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2] \quad \forall x \in \mathbb{R}^N$$

Metoda gradientów jest iteracyjnym algorytmem wyszukiwania minimum zadanej funkcji.

Założenia:

1. funkcja jest ciągła i różniczkowalna
2. funkcja w badanej dziedzinie jest ściśle wypukła

Algorytm zaczyna się wyborem punktu startowego, w którym to obliczany jest kierunek poszukiwań rozwiązania. Jeżeli następny punkt nie spełnia warunku stopu algorytmu całe postępowanie jest powtarzane.

Kryterium stopu:

$$\begin{aligned} \|\nabla f(\mathbf{x}_k)\| &\leq \epsilon, \\ \|\mathbf{x}_{k+1} - \mathbf{x}_k\| &\leq \epsilon. \end{aligned}$$

ϵ to zadana precyzja, a $\|\cdot\|$ zadana norma.

Kod programu:

```
/*
*****
*
*           Artur Guniewicz - Zadanie 18
*
*
*****
*/

#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define MAX 10    // maksymalny wymiar funkcji
#define No_of_Iter 120 // ilość iteracji
#define Max 64

double P0[MAX], Pmin[MAX];
double Ymin, Y0;
double H = 1.0;
double Err = 1.0;
double S[MAX], G[MAX];
int condition;

double Delta = 1e-64; // tolerancja dla punktów
double Epsilon = 1e-64; // tolerancja dla wartości funkcji

// funkcja Rosenbrocka
double RosenbrockFunction(double *P)
{
    return ((1 - P[0]) * (1 - P[0]) + 100 * (P[1] - P[0] * P[0]) * (P[1]
- P[0] * P[0]) + 100 * (P[2] - P[1] * P[1]) * (P[2] - P[1] * P[1]) +
100 * (P[3] - P[2] * P[2]) * (P[3] - P[2] * P[2]));
}

void Gradient(double *P)
{

```

```

// gradient
G[0] = (-2.0 + 2 * P[0] - 400 * P[0] * P[1] + 400 * P[0] * P[0] *
P[0]);
G[1] = (200 * P[1] - 200 * P[0] * P[0] - 400 * P[1] * P[2] + 400 *
P[1] * P[1] * P[1]);
G[2] = (200 * P[2] - 200 * P[1] * P[1] - 400 * P[2] * P[3] + 400 *
P[2] * P[2] * P[2]);
G[3] = (200 * P[3] - 200 * P[2] * P[2]);

// "strona" gradientu
S[0] = -G[0] / sqrt(G[0] * G[0] + G[1] * G[1] + G[2] * G[2] + G[3] *
G[3]);
S[1] = -G[1] / sqrt(G[0] * G[0] + G[1] * G[1] + G[2] * G[2] + G[3] *
G[3]);
S[2] = -G[2] / sqrt(G[0] * G[0] + G[1] * G[1] + G[2] * G[2] + G[3] *
G[3]);
S[3] = -G[3] / sqrt(G[0] * G[0] + G[1] * G[1] + G[2] * G[2] + G[3] *
G[3]);
}

void Quadmin(int NC, double Delta, double Epsilon)
{
    double P1[MAX];
    double P2[MAX];
    double H0, H1, H2, Hmin, E0, E1, E2, Y1, Y2, D;
    int i, J;

    condotion = 0;
    J = 0;

    for (i = 0; i < NC; i++)
    {
        P1[i] = P0[i] + H * S[i];
        P2[i] = P0[i] + 2.0 * H * S[i];
    }

    Y1 = RosenbrockFunction(P1);
    Y2 = RosenbrockFunction(P2);

    while ((J < No_of_Iter) && (condotion == 0))

```

```

{
    if (Y0 <= Y1)
    {
        Y2 = Y1;
        H = H / 2.0;

        for (i = 0; i < NC; i++)
        {
            P2[i] = P1[i];
            P1[i] = P0[i] + H * S[i];
        }

        Y1 = RosenbrockFunction(P1);
    }

    else
    {
        if (Y2 < Y1)
        {
            Y1 = Y2;
            H = 2.0 * H;

            for (i = 0; i < NC; i++)
            {
                P1[i] = P2[i];
                P2[i] = P0[i] + 2.0 * H * S[i];
            }

            Y2 = RosenbrockFunction(P2);
        }

        else
            condotion = -1;
    }
}

if (H < Delta)
    condotion = 1;

D = 4.0 * Y1 - 2.0 * Y0 - 2.0 * Y2;

```

```

if (D < 0)
    Hmin = H * (4.0 * Y1 - 3.0 * Y0 - Y2) / D;

else
{
    condotion = 4;
    Hmin = H / 3.0;
}

for (i = 0; i < NC; i++)
    Pmin[i] = P0[i] + Hmin * S[i];

Ymin = RosenbrockFunction(Pmin);

H0 = fabs(Hmin);
H1 = fabs(Hmin - H);
H2 = fabs(Hmin - 2.0 * H);

if (H0 < H)
    H = H0;

if (H1 < H)
    H = H1;

if (H2 < H)
    H = H2;

if (H < Delta)
    condotion = 1;

E0 = fabs(Y0 - Ymin);
E1 = fabs(Y1 - Ymin);
E2 = fabs(Y2 - Ymin);

if (E0 < Err)
    Err = E0;

else if (E1 < Err)
    Err = E1;

```

```

    else if (E2 < Err)
        Err = E2;

    else if ((E0 == 0) && (E1 == 0) && (E2 == 0))
        Err = 0;

    if (Err < Epsilon)
        condotion = 2;

    if ((condotion == 2) && (H < Delta))
        condotion = 3;

    J++;
}

int main()
{
    int NC = 4; // liczba składników gradientu
    int counter = 0;
    int i, j, ILE_PKT;

printf("-----
----- \n");

    printf("----- Metoda Gradientów
----- \n");

printf("-----
----- \n");

    printf("\nPodaj ilość losowych punktów początkowych: ");
    scanf("%d", &ILE_PKT);

    printf("\n");
    printf("\n");

printf("-----
----- \n");

```

```

    printf("--- Rozpoczęto losowanie %d punktów w kwadracie [-100,
100]x[-100, 100]x[-100, 100]x[-100, 100] --- \n", ILE_PKT);

printf("-----
----- \n");

printf("\n");
printf("\n");

srand(time(NULL));

for (j = 0; j < ILE_PKT; j++)
{
    P0[0] = ((rand() % 201) - 100 + rand() / ((double)RAND_MAX));
    P0[1] = ((rand() % 201) - 100 + rand() / ((double)RAND_MAX));
    P0[2] = ((rand() % 201) - 100 + rand() / ((double)RAND_MAX));
    P0[3] = ((rand() % 201) - 100 + rand() / ((double)RAND_MAX));

    printf("Wylosowano punkt: \n");
    printf("%.10lf, %.10lf, %.10lf, %.10lf) \n", P0[0], P0[1],
P0[2], P0[3]);

    Y0 = RosenbrockFunction(P0);

    while ((counter < Max) && (H < Delta) || (Err > Epsilon))
    {
        Gradient(P0);
        Quadmin(NC, Delta, Epsilon);

        for (i = 0; i < NC; i++)
            P0[i] = Pmin[i];

        Y0 = Ymin;

        counter++;
    }

printf("-----
----- \n");

```

```

    printf("Minimum lokalne (lub globalne) jest w punkcie: \n");
    printf("(%.10lf, %.10lf, %.10lf, %.10lf) \n", Pmin[0], Pmin[1],
Pmin[2], Pmin[3]);
    printf("Wartość funkcji w minimum to: %lf \n\n", Ymin);

printf("-----
----- \n");

    if (condiotion == 0)
        printf("Zbieżność nie osiągnieta ponieważ osiągnięto
maksymalną ilość iteracji \n");

    if (condiotion == 1)
        printf("Osiagnieto zbieżność odciętej \n");

    if (condiotion == 2)
        printf("Osiagnieto zbieżność odciętej \n");

    if (condiotion == 3)
        printf("Osiagnieto zbieżność dla obu współrzędnych \n");

    if (condiotion == 4)
        printf("Zbieżność jest wątpliwa, ponieważ pojawiło się
dzielenie przez 0 \n");

    printf("\n");
    printf("\n");
}

return 0;
}

```

Kompilacja:

g++ Zadanie18.cpp -o Zadanie18 && ./Zadanie18

Wyniki:

Przykładowe wyniki dla 4 wylosowanych punktów początkowych:

```
----- Metoda Gradientów -----  
  
Podaj ilość losowych punktów początkowych: 4  
  
--- Rozpoczęto losowanie 4 punktów w kwadracie [-100, 100]x[-100, 100]x[-100, 100]x[-100, 100] ---  
  
Wylosowano punkt:  
(22.2295165356, -4.0019882675, 22.0394685785, 3.0775880223)  
  
Minimum lokalne (lub globalne) jest w punkcie:  
(1.0000000068, 1.0000000136, 1.0000000272, 1.0000000544)  
Wartość funkcji w minimum to: 0.000000  
  
Osiągnięto zbieżność odciętej  
  
Wylosowano punkt:  
(19.5400894040, -45.8076739594, 82.9862011536, 24.8128255931)  
  
Minimum lokalne (lub globalne) jest w punkcie:  
(1.0000000068, 1.0000000136, 1.0000000272, 1.0000000544)  
Wartość funkcji w minimum to: 0.000000  
  
Osiągnięto zbieżność odciętej  
  
Wylosowano punkt:  
(-25.1944319854, -22.6954784182, 32.0572794974, -80.5890361497)  
  
Minimum lokalne (lub globalne) jest w punkcie:  
(1.0000000068, 1.0000000136, 1.0000000272, 1.0000000544)  
Wartość funkcji w minimum to: 0.000000  
  
Osiągnięto zbieżność odciętej  
  
Wylosowano punkt:  
(-51.3803081607, -89.5358250819, 26.0932134716, 69.3006467159)  
  
Minimum lokalne (lub globalne) jest w punkcie:  
(1.0000000068, 1.0000000136, 1.0000000272, 1.0000000544)  
Wartość funkcji w minimum to: 0.000000  
  
Osiągnięto zbieżność odciętej
```

Komentarz:

Program jest napisany w języku C, jednak zapisany został z rozszerzeniem .cpp oraz z dodatkową biblioteką <iostream>. Po jej usunięciu program można skompilować kompilatorem języka C.