Marcin Rociek
Informatyka, II rok

Metody Obliczeniowe w Nauce i Technice

laboratorium

zestaw 10: minimalizacja funkcji dwóch zmiennych

Znaleźć minimum funkcji

1. $f(x_1, x_2) = x_1^2 + (x_2 - 1)^2$
2. $f(x_1, x_2) = 100(x_1 - x_2)^2 + (1 - x_1)^2$
3. $f(x_1, x_2) = 837.9658 - x_1 \sin\left(\sqrt{|x_1|}\right) - x_2 \sin\left(\sqrt{|x_2|}\right)$

metodą simplex oraz wybraną metodą gradientową.

---

a): metoda simplex

Do wykonania tego zadania wykorzystałem funkcję **amoeba** z biblioteki *Numerical Recipes*, znajdującą minimum wielowymiarowej funkcji metodą simplex. Jako punkty początkowe simpleksu wybierane są trzy punkty leżące w okolicy prawdziwego minimum.

Treść programu:

```
#include <stdio.h>
#include <math.h>
#include "matrix.h"
#include "nrutil.h"

extern void amoeba(double **p, double *y,int ndim, double ftol,
                   double (*funk)(double *), int *nfunk);

#define MP 3
#define NP 2
#define FTOL 1.0e-6
#ifndef M_PI
  #define M_PI 3.14159265358979323846
#endif

/*
 * Badane funkcje
 */
double func1(double *x)
{
  return x[1]*x[1]+(x[2]-1.0)*(x[2]-1.0);
}

double func2(double *x)
{
  return 100.0*(x[1]-x[2])*(x[1]-x[2])+(1.0-x[1])*(1.0-x[1]);
}

double func3(double *x)
{
  return 837.9658-x[1]*sin(sqrt(fabs(x[1])))-x[2]*sin(sqrt(fabs(x[2])));
}

double (*funcs[3])(double *x) = { func1, func2, func3 };

/*
 * Prawdziwe wartosci minimow dla badanych funkcji
 */
double realmins[3][2] = { { 0.0, 1.0 },
                          { 1.0, 1.0 },
                          { 420.9687, 420.9687 } };

/*
 * Czesc glowna programu
 */
void main(void)
{
```

```
  int f, i, nfunc, j, ndim = NP;
  double *x, *y, **p;

  x = dvector(1, NP);
  y = dvector(1, MP);
  p = dmatrix(1, MP, 1, NP);

  for (f=0; f<3; f++)
  {
    printf("\nfunkcja %d\n", f+1);

    printf(" startowe wspolrzedne simpleksu:\n");

    // ustalenie wierzcholkow startowych simpleksu
    for (i=1; i<=MP; i++)
    {
      float angle = (i-1)*2.0*M_PI/3.0;
      x[1] =  p[i][1] = realmins[f][0]+5.0*cos(angle)+0.1;
      x[2] =  p[i][2] = realmins[f][1]+5.0*sin(angle);
      printf("[%d] ", i);
      vec_show(x, NP);
      y[i] = (*funcs[f])(x);
    }

    // szukanie minimum
    amoeba(p, y, ndim, FTOL, funcs[f], &nfunc);

    printf(" znalezione minimum:\n");

    // wypisanie wynikow
    for (i=1; i<=MP; i++)
    {
      printf("[%d] ", i);
      for (j=1; j<=NP; j++)
        printf("%f ", p[i][j]);
      printf("     f=%f\n", y[i]);
    }
    printf("ilosc wywolan funkcji: %d\n", nfunc);
  }

  free_dmatrix(p, 1, MP, 1, NP);
  free_dvector(y, 1, MP);
  free_dvector(x, 1, NP);
}
```

Wynik działania programu:

```
funkcja 1
 startowe wspolrzedne simpleksu:
[1] 5.100000 1.000000
[2] -2.400000 5.330127
[3] -2.400000 -3.330127
 znalezione minimum:
[1] -0.000000 1.000000      f=0.000000
[2] -0.000000 1.000000      f=0.000000
[3] 0.000000 1.000000      f=0.000000
ilosc wywolan funkcji: 264

funkcja 2
 startowe wspolrzedne simpleksu:
[1] 6.100000 1.000000
[2] -1.400000 5.330127
[3] -1.400000 -3.330127
 znalezione minimum:
[1] 1.000000 1.000000      f=0.000000
[2] 1.000000 1.000000      f=0.000000
[3] 1.000000 1.000000      f=0.000000
ilosc wywolan funkcji: 272

funkcja 3
 startowe wspolrzedne simpleksu:
[1] 426.068700 420.968700
[2] 418.568700 425.298827
[3] 418.568700 416.638573
 znalezione minimum:
[1] 420.968756 420.968752      f=0.000025
[2] 420.968745 420.968752      f=0.000025
[3] 420.968748 420.968741      f=0.000025
ilosc wywolan funkcji: 80
```

Jak widać metoda ta jest niezbyt efektywna. Ilość kroków nie jest zbyt rewelacyjna (zwłaszcza w porównaniu z badaną niżej metodą gradientową). Wadą tej metody jest również konieczność rozważnego dobrania punktów początkowych simpleksu. W pierwszym podejściu spróbowałem umieszczać je w wierzchołkach trójkąta równobocznego, którego środkiem jest prawdziwe minimum funkcji. Ku mojemu zdumieniu okazało się jednak, że funkcja **amoeba** w ogóle nie chce znaleźć minimum pierwszej funkcji (tzn. kończyła pracę po 0 iteracjach). Powodem takiego zachowania jest symetryczność funkcji pierwszej. Wartości tej funkcji w trzech punktach oddalonych od punktu *(0, 1)* o taką samą odległość są identyczne, a warunkiem zakończenia metody simpleksu jest właśnie równośćwartości funkcji w punktach tworzących simpleks. Dlatego też trójkąt tworzący startowy simpleks jest przesunięty wzdłuż x_1 o *0.1*.

---

b): metoda gradientowa

W tym zadaniu wykorzystałem funkcję **frprmn** z biblioteki *Numerical Recipes* znajdującą minimum wielowymiarowej funkcji metodą gradientową (*Fletcher-Reeves-Polak-Ribiere minimization*).

Treść programu:

```
#include <stdio.h>
#include <math.h>
#include "nr.h"
#include "nrutil.h"

#define NDIM 2
#define FTOL 1.0e-6
#define PIO2 1.5707963

/*
 * Badane funkcje
 */
float func1(float *x)
{
  return x[1]*x[1]+(x[2]-1.0)*(x[2]-1.0);
}

float func2(float *x)
{
  return 100.0*(x[1]-x[2])*(x[1]-x[2])+(1.0-x[1])*(1.0-x[1]);
}

float func3(float *x)
{
  return 837.9658-x[1]*sin(sqrt(fabs(x[1])))-x[2]*sin(sqrt(fabs(x[2])));
}

float (*funcs[3])(float *x) = { func1, func2, func3 };

/*
 * Funkcje wyznaczajace pochodne czastkowe badanych funkcji
 */
void dfunc1(float *x, float *df)
{
  df[1] = 2.0*x[1];
  df[2] = 2.0*(x[2]-1.0);
}

void dfunc2(float *x, float *df)
{
  df[1] = 202.0*x[1]-200.0*x[2]-2.0;
  df[2] = 200.0*x[2]-200.0*x[1];
}

void dfunc3(float *x, float *df)
{
  df[1] = -0.5*sqrt(fabs(x[1]))*cos(sqrt(fabs(x[1])))-sin(sqrt(fabs(x[1])));
  df[2] = -0.5*sqrt(fabs(x[2]))*cos(sqrt(fabs(x[2])))-sin(sqrt(fabs(x[2])));
}

void (*dfuncs[3])(float *x, float *df) = { dfunc1, dfunc2, dfunc3 };

/*
 * Prawdziwe wartosci minimow dla badanych funkcji
 */
```

```
float realmins[3][2] = { { 0.0, 1.0 },
                         { 1.0, 1.0 },
                         { 420.9687, 420.9687 } };

void main(void)
{
  int f, iter, k;
  float angle, fret, *p;

  p = vector(1, NDIM);

  for (f=0; f<3; f++)
  {
    printf("\n\nfunkcja %d\n", f+1);
    for (k=0; k<4; k++)
    {
      angle = PIO2*k/4.0;
      p[1] = realmins[f][0]+5.0*cos(angle);
      p[2] = realmins[f][1]+5.0*sin(angle);

      printf("\npunkt startowy: (%f,%f)\n", p[1], p[2]);

      frprmn(p, NDIM, FTOL, &iter, &fret, funcs[f], dfuncs[f]);

      printf("znalezione minimum: (%f,%f)      f=%f\n", p[1], p[2], fret);
      printf("ilosc iteracji: %d\n", iter);
    }
  }

  free_vector(p, 1, NDIM);
}
```

Wynik działania programu:

```
funkcja 1

punkt startowy: (5.000000,1.000000)
znalezione minimum: (0.000000,1.000000)      f=0.000000
ilosc iteracji: 2

punkt startowy: (4.619398,2.913417)
znalezione minimum: (-0.000000,1.000000)      f=0.000000
ilosc iteracji: 4

punkt startowy: (3.535534,4.535534)
znalezione minimum: (0.000000,1.000000)      f=0.000000
ilosc iteracji: 2

punkt startowy: (1.913417,5.619398)
znalezione minimum: (-0.000000,1.000000)      f=0.000000
ilosc iteracji: 4


funkcja 2

punkt startowy: (6.000000,1.000000)
znalezione minimum: (1.000000,1.000000)      f=0.000000
ilosc iteracji: 5

punkt startowy: (5.619398,2.913417)
znalezione minimum: (1.000000,1.000000)      f=0.000000
ilosc iteracji: 5

punkt startowy: (4.535534,4.535534)
znalezione minimum: (1.000000,1.000000)      f=0.000000
ilosc iteracji: 5

punkt startowy: (2.913417,5.619398)
znalezione minimum: (1.000000,1.000000)      f=0.000000
ilosc iteracji: 4


funkcja 3

punkt startowy: (425.968689,420.968689)
znalezione minimum: (420.968628,420.968750)      f=0.000025
ilosc iteracji: 2

punkt startowy: (425.588074,422.882106)
znalezione minimum: (420.968719,420.968750)      f=0.000025
```

```
ilosc iteracji: 4

punkt startowy: (424.504211,424.504223)
znalezione minimum: (420.968506,420.968506)      f=0.000025
ilosc iteracji: 2

punkt startowy: (422.882111,425.588087)
znalezione minimum: (420.968750,420.968719)      f=0.000025
ilosc iteracji: 4
```

Wyraźnie widać przewagę tej metody nad metodą simpleksu, gdyż znajduje ona minimum w znacznie krótszym czasie. Wadą jej jest natomiast to, że wymaga ‘ręcznego’ podania pochodnych cząstkowych. Ciekawym zjawiskiem w tej metodzie jest różna ilość iteracji w zależności od położenia punktu początkowego, mimo iż punkt początkowy jest oddalony od minimum o tą samą odległość.