METODY NUMERYCZNE

ZADANIE 14

Artur Guniewicz

14. Rozwiąż układ równań

$$2x^2 + y^2 = 2 \tag{13a}$$

$$\left(x - \frac{1}{2}\right)^2 + (y-1)^2 = \frac{1}{4} \tag{13b}$$

Metoda: metoda Newtona

Metoda Newtona to iteracyjna metoda znajdowania miejsc zerowych funkcji spełniającej określone założenia.

Pierwszy krok algorytmu to wybranie punktu startowego, z którego wyprowadzana jest styczna. Odcięta punktu przecięcia stycznej z osią OX jest pierwszym przybliżeniem rozwiązania. Jeżeli przybliżenie to nie jest satysfakcjonujące to wybierany jest ten punkt przecięcia i kroki algorytmu powtarzają się.

Przybliżenia te można przedstawić wzorem:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

Kod programu:

```
/*
************************************************************
*                                                          *
*               Artur Guniewicz - Zadanie 14               *
*                                                          *
************************************************************
*/

#include <iostream>
#include <iomanip>
```

```cpp
#include <stdio.h>
#include <stdlib.h>

using namespace std;

const int ile_iter = 5;

// Równania

double rownanie1(double x, double y)
{
   return (2 * x * x + y * y - 2);
}

double rownanie2(double x, double y)
{
   return ((x - 0.5) * (x - 0.5) + (y - 1) * (y - 1) - 0.25);
}


// Pochodne czastkowe

double rownanie1_po_x(double x, double y)
{
   return 4 * x;
}

double rownanie1_po_y(double x, double y)
{
   return 2 * y;
}

double rownanie2_po_x(double x, double y)
{
   return (2 * x - 1);
}

double rownanie2_po_y(double x, double y)
{
   return 2 * (y - 1);
```

```
}

void metodaNewtona(double x_0[2], double wynik[2])
{
   double Jacobian[2][2];

   double det = rownanie1_po_x(x_0[0], x_0[1]) * rownanie2_po_y(x_0[0],
x_0[1]) - (rownanie1_po_y(x_0[0], x_0[1]) * rownanie2_po_x(x_0[0],
x_0[1]));

   Jacobian[0][0] = rownanie2_po_y(x_0[0], x_0[1]);
   Jacobian[0][1] = -rownanie1_po_y(x_0[0], x_0[1]);
   Jacobian[1][0] = -rownanie2_po_x(x_0[0], x_0[1]);
   Jacobian[1][1] = rownanie1_po_x(x_0[0], x_0[1]);

   for (int i = 0; i < 2; i++)
       for (int j = 0; j < 2; j++)
           Jacobian[i][j] *= (1 / det);

   double F[2] = {rownanie1(x_0[0], x_0[1]), rownanie2(x_0[0],
x_0[1])};
   double mnozenie[2] = {Jacobian[0][0] * F[0] + Jacobian[0][1] * F[1],
Jacobian[1][0] * F[0] + Jacobian[1][1] * F[1]};

   wynik[0] = x_0[0] - mnozenie[0];
   wynik[1] = x_0[1] - mnozenie[1];
}

int main()
{
   int i = 0;
   double x_0[2] = {0.5, 1.3};
   double wynik[2];

   cout << setprecision(8) << fixed;
   cout << endl << "Dla punktu poczatkowego: (" << x_0[0] << ", " <<
x_0[1] << ")" << endl;

   while (i < ile_iter)
   {
```

```
        metodaNewtona(x_0, wynik);
        x_0[0] = wynik[0];
        x_0[1] = wynik[1];
        i++;
    }

    cout << endl << "Po iteracjach: " << i << endl;
    cout << "x = " << wynik[0] << endl;
    cout << "y = " << wynik[1] << endl << endl;

    return 0;
}
```

Kompilacja:

cd "adres pliku" && g++ Zadanie14.cpp -g -std=c++14 -ansi -pedantic -Wall -lm -o Zadanie14 && ./Zadanie14

Wyniki:

```
Dla punktu poczatkowego: (0.50000000, 1.30000000)

Po iteracjach: 5
x = 0.18639893
y = 1.38942826
```