

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Кафедра программных систем

**Задание №5**

**Метод Ньютона**

Выполнил: Мороз

Илья Олегович

Группа № K3220

Преподаватель: Иванов Сергей Евгеньевич

Санкт-Петербург

2020

**Задание:** реализовать решение системы нелинейных уравнений с помощью метода Ньютона.

**Исходный код:**

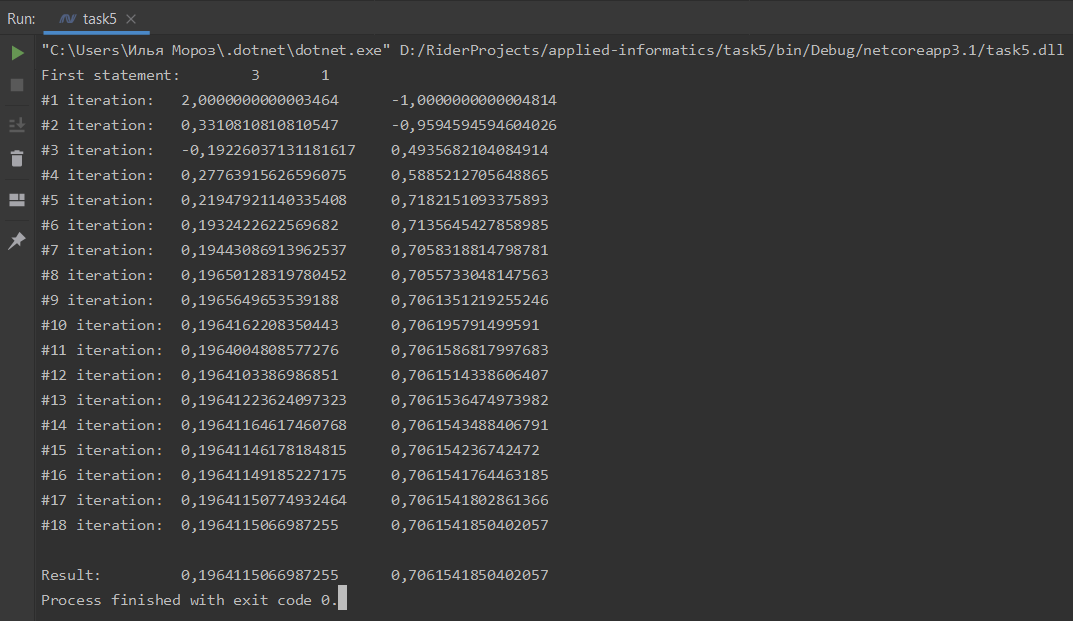
**Program.cs**

using System;  
using System.Linq;  
  
namespace task5  
{  
 class Program  
 {  
 static void Main(string[] args)  
 {  
 bool isSolved = false;  
  
 */\*  
 \* Init first statement  
 \*/* double[] x = {3, 1};  
 Console.Write("First statement: ");  
 foreach (double i in x)  
 {  
 Console.Write($"\t{i}");  
 }  
 Console.WriteLine();  
  
 int iterationCount = 0;  
 while (!isSolved)  
 {  
 double[] newX = Iterate(x);  
 isSolved = !L1NormCheck(x, newX);  
 x = newX;  
 iterationCount++;  
 Console.Write($"#{iterationCount} iteration: ");  
 foreach (double i in x)  
 {  
 Console.Write($"\t{i}");  
 }  
 Console.WriteLine();  
  
 if (iterationCount > 10000)  
 {  
 Console.WriteLine("\nTry another start statement");  
 break;  
 }  
 }  
  
 Console.Write("\nResult: ");  
 foreach (double i in x)  
 {  
 Console.Write($"\t{i}");  
 }  
 }  
  
 */\*  
 \* Init F(x)  
 \* Use two f(x) functions  
 \*/* static double[] Fx(double[] x)  
 {  
 double x1 = x[0];  
 double x2 = x[1];  
 double[] result = new double[2];  
 result[0] = 0.1 \* x1 \* x1 + x1 + 0.2 \* x2 \* x2 - 0.3;  
 result[1] = 0.2 \* x1 \* x1 + x2 - 0.1 \* x1 \* x2 - 0.7;  
 return result;  
 }  
  
 */\*  
 \* Numerically calculate the derivative f(x) with respect to the i-th variable with accuracy Ɛ  
 \*/* static double[] DFx(double[] x, int i)  
 {  
 double epsilon = 1e-3;  
 double[] xPlusEps = new double[x.Length];  
 x.CopyTo(xPlusEps, 0);  
 xPlusEps[i] += epsilon;  
 double[] xMinusEps = new double[x.Length];  
 x.CopyTo(xMinusEps, 0);  
 xMinusEps[i] -= epsilon;  
 double[] y1 = Fx(xPlusEps);  
 double[] y2 = Fx(xMinusEps);  
 double[] dVector = new double[y1.Length];  
 for (int j = 0; j < y1.Length; j++)  
 {  
 dVector[j] = (y1[j] - y2[j]) / (2 \* epsilon);  
 }  
  
 return dVector;  
 }  
  
 */\*  
 \* Initialize the system,  
 \* the solution of which will be the shifts of the initial vector of approximate roots  
 \*/* static double[][] CreateSystem(double[] x)  
 {  
 double[][] matrix = new double[x.Length][];  
 for (int i = 0; i < matrix.Length; i++)  
 {  
 */\*  
 \* Partial derivatives  
 \*/* double[] vec = DFx(x, i);  
 matrix[i] = new double[x.Length + 1];  
 for (int j = 0; j < matrix[0].Length - 1; j++)  
 {  
 matrix[i][j] = vec[j];  
 }  
  
 matrix[i][matrix[0].Length - 1] = -Fx(x)[i];  
 }  
  
 return matrix;  
 }  
  
 */\*  
 \* Check for proximity to the answer with the specified accuracy using the L1-norm  
 \*/* static bool L1NormCheck(double[] x, double[] y)  
 {  
 double[] vec = new double[x.Length];  
 for (int j = 0; j < x.Length; j++)  
 {  
 vec[j] = Math.Abs(x[j] - y[j]);  
 }  
  
 double sum = vec.Sum();  
 if (sum < 0.00000001) return false;  
  
 return true;  
 }  
  
 static double[] Iterate(double[] x)  
 {  
 */\*  
 \* Create system and solve it by Gauss method from task #4  
 \*/* double[] newX = new double[x.Length];  
 double[] systemSolution = Gauss.Solve(CreateSystem(x));  
 for (int i = 0; i < systemSolution.Length; i++)  
 {  
 newX[i] = x[i] + systemSolution[i];  
 }  
  
 */\*  
 \* Return new x (old x + delta x)  
 \*/* return newX;  
 }  
 }  
}

**Gauss.cs**

using System;  
  
namespace task5  
{  
 public class Gauss  
 {  
 */\*  
 \* Subtract row #n  
 \* Return new matrix  
 \*/* private static void SubtractRow(ref double[][] matrix, int n)  
 {  
 double m = matrix[n][n];  
 for (int i = n + 1; i < matrix.Length; i++)  
 {  
 double t = matrix[i][n] / m;  
 for (int j = n; j < matrix[0].Length; j++)  
 {  
 matrix[i][j] = matrix[i][j] - matrix[n][j] \* t;  
 if (matrix[i][j] < 0.0001 && matrix[i][j] > 0.0001) matrix[i][j] = 0;  
 }  
 }  
 }  
  
 */\*  
 \* Get Lead in column #n  
 \*/* static void SelectLeading(ref double[][] matrix, int n)  
 {  
 int indexOfRowWithMax = n;  
 for (int i = n + 1; i < matrix.Length; i++)  
 {  
 if (Math.Abs(matrix[indexOfRowWithMax][n]) < Math.Abs(matrix[i][n]))  
 {  
 indexOfRowWithMax = i;  
 }  
 }  
  
 if (indexOfRowWithMax != n)  
 {  
 for (int i = n; i < matrix[0].Length; i++)  
 {  
 matrix[n][i] += matrix[indexOfRowWithMax][i] - (matrix[indexOfRowWithMax][i] = matrix[n][i]);  
 }  
 }  
 }  
  
 */\*  
 \* Get matrix in triangular view  
 \* Return false if can't find solution  
 \*/* private static bool TriangleMatrix(ref double[][] matrix)  
 {  
 for (int i = 1; i < matrix.Length; i++)  
 {  
 SelectLeading(ref matrix, i - 1);  
 if (Math.Abs(matrix[i - 1][i - 1]) > 0.0001)  
 {  
 SubtractRow(ref matrix, i - 1);  
 }  
 else  
 {  
 return false;  
 }  
 }  
  
 return true;  
 }  
  
 */\*  
 \* Return solution for matrix by Gauss method  
 \*/* public static double[] Solve(double[][] matrix)  
 {  
 if (!TriangleMatrix(ref matrix)) return null;  
 double[] resolve = new double[matrix.Length];  
 int countOfVariables = matrix[0].Length - 1;  
 for (int i = resolve.Length - 1; i >= 0; i--)  
 {  
 double sum = 0;  
 for (int j = i + 1; j < countOfVariables; j++)  
 {  
 sum += resolve[j] \* matrix[i][j];  
 }  
  
 resolve[i] = (matrix[i][countOfVariables] - sum) / matrix[i][i];  
 }  
  
 return resolve;  
 }  
 }  
}

**Результат работы:**

****

**Вывод:** был реализовано нахождение решения системы методом Ньютона.

Исходный код также доступен на GitHub: [ilyamore88/applied-informatics/task5](https://github.com/ilyamore88/applied-informatics/tree/master/task5).