

Национальный исследовательский университет ИТМО

Факультет инфокоммуникационных технологий

**Лабораторная работа 5**

Выполнил: Орел

Даниил Максимович

Группа № K3221

Проверил: Иванов С.Е.

Санкт-Петербург

2020

**Цель работы:**

* Реализовать алгоритм решения системы уравнений методом Ньютона

**Ход работы:**

Блок-схема алгоритма:

https://algowiki-project.org/algowiki/pool/images/0/03/NewtonScheme.png

Рисунок 1 – Блок-схема метода Ньютона

1. В программу был добавлен NuGet пакет MathNet.Symbolics, позволяющий легко работать с функциями, в том числе брать производные:

using MathNet.Symbolics;

using Expr = MathNet.Symbolics.SymbolicExpression;

1. Некоторые шаги в алгоритме включают себя работу с матрицами, поэтому был добавлен класс **Matrix** из лабораторной работы 1:

using System;

namespace Lab\_05

{

static class Matrix

{

//Функция вывода матрицы дробных чисел на экран

static public void PrintMatrix(double[,] matrix)

{

int rows = matrix.GetUpperBound(0) + 1;

int columns = matrix.Length / rows;

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < columns; j++)

{

Console.Write(Math.Round(matrix[i, j], 4) + "\t");

}

Console.WriteLine();

}

}

static public double[,] MatrixAddition(double[,] matrix1, double[,] matrix2)

{

double[,] matrix;

int rows\_matrix1 = matrix1.GetUpperBound(0) + 1;

int columns\_matrix1 = matrix1.Length / rows\_matrix1;

int rows\_matrix2 = matrix2.GetUpperBound(0) + 1;

int columns\_matrix2 = matrix2.Length / rows\_matrix2;

matrix = new double[rows\_matrix1, columns\_matrix1];

if (isMatricesValidForAddition(rows\_matrix1, columns\_matrix1, rows\_matrix2, columns\_matrix2))

{

for (int i = 0; i < rows\_matrix1; i++)

for (int j = 0; j < columns\_matrix1; j++)

{

matrix[i, j] = matrix1[i, j] + matrix2[i, j];

}

}

else Console.WriteLine("Невозможно выполнить сложение матрицы, т.к. количество строк и столбцов первой матрицы не соответсвует количеству строк и столбцов второй матрицы");

return matrix;

}

static public double[,] MatrixSubtraction(double[,] matrix1, double[,] matrix2)

{

double[,] matrix;

int rows\_matrix1 = matrix1.GetUpperBound(0) + 1;

int columns\_matrix1 = matrix1.Length / rows\_matrix1;

int rows\_matrix2 = matrix2.GetUpperBound(0) + 1;

int columns\_matrix2 = matrix2.Length / rows\_matrix2;

matrix = new double[rows\_matrix1, columns\_matrix1];

if (isMatricesValidForAddition(rows\_matrix1, columns\_matrix1, rows\_matrix2, columns\_matrix2))

{

for (int i = 0; i < rows\_matrix1; i++)

for (int j = 0; j < columns\_matrix1; j++)

{

matrix[i, j] = matrix1[i, j] - matrix2[i, j];

}

}

else Console.WriteLine("Невозможно выполнить сложение матрицы, т.к. количество строк и столбцов первой матрицы не соответсвует количеству строк и столбцов второй матрицы");

return matrix;

}

//Функция, проверяющая совпадают ли размерности складываемых матриц

static private bool isMatricesValidForAddition(int rows1, int columns1, int rows2, int columns2)

{

if (rows1 == rows2 && columns1 == columns2)

return true;

else

return false;

}

static public double[,] MatrixMultiply(double[,] matrix1, double[,] matrix2)

{

double[,] matrix;

int rows\_matrix1 = matrix1.GetUpperBound(0) + 1;

int columns\_matrix1 = matrix1.Length / rows\_matrix1;

int rows\_matrix2 = matrix2.GetUpperBound(0) + 1;

int columns\_matrix2 = matrix2.Length / rows\_matrix2;

matrix = new double[rows\_matrix1, columns\_matrix2];

if (isMatricesValidForMultiply(columns\_matrix1, rows\_matrix2))

{

for (var i = 0; i < rows\_matrix1; i++)

{

for (var j = 0; j < columns\_matrix2; j++)

{

matrix[i, j] = 0;

for (var k = 0; k < columns\_matrix1; k++)

{

matrix[i, j] += matrix1[i, k] \* matrix2[k, j];

}

}

}

}

else Console.WriteLine("Умножение матриц не возможно! Количество столбцов первой матрицы не равно количеству строк второй матрицы.");

return matrix;

}

//Проверка совпдает ли количество столбцов первой матрицы с количеством строк второй матрицы:

static private bool isMatricesValidForMultiply(int columns1, int rows2)

{

if (rows2 == columns1)

return true;

else

return false;

}

static public double[,] CalculateInverseMatrix(double[,] matrix)

{

int k = 1;

double[,] algebraic\_complement\_matrix;

int rows = matrix.GetUpperBound(0) + 1;

int columns = matrix.GetUpperBound(1) + 1;

algebraic\_complement\_matrix = new double[rows, rows];

double[,] minor;

if (isSquare(rows, columns))

{

double determinant = CalculateDeterminant(matrix, rows);

if (determinant != 0)

{

minor = new double[rows - 1, rows - 1];

matrix = FindTransparentMatrix(matrix);

for (int i = 0; i < rows; i++)

for (int j = 0; j < rows; j++)

{

algebraic\_complement\_matrix[i, j] = Math.Pow(-1, (i + j)) \* CalculateDeterminant(GetMatr(matrix, minor, i, j, rows), rows - 1);

}

for (int i = 0; i < rows; i++)

for (int j = 0; j < rows; j++)

{

algebraic\_complement\_matrix[i, j] /= (double)determinant;

}

}

else Console.WriteLine("Матрица вырожденная и не имеет обратной матрицы");

}

else Console.WriteLine("Матрица не квадратная! Только квадратная матрица имеет обратную.");

return algebraic\_complement\_matrix;

}

//Проверка: является ли данная матрица квадратной?

static private bool isSquare(int rows, int columns)

{

if (rows == columns)

return true;

else return false;

}

//Вычисление определителя

static public double CalculateDeterminant(double[,] matrix, int rows)

{

double det = 0;

int k = 1;

double[,] new\_matrix = new double[rows, rows];

if (rows == 1)

return matrix[0, 0];

else if (rows == 2)

{

det = matrix[0, 0] \* matrix[1, 1] - matrix[1, 0] \* matrix[0, 1];

return det;

}

else

{

for (int i = 0; i < rows; i++)

{

det += k \* matrix[i, 0] \* CalculateDeterminant(GetMatr(matrix, new\_matrix, i, 0, rows), rows - 1);

k = -k;

}

return det;

}

}

//Получение матрицы путем вычеркивание i-ой строки и j-ого столбца

static public double[,] GetMatr(double[,] matrix, double[,] p, int i, int j, int m)

{

int ki, kj, di, dj;

di = 0;

for (ki = 0; ki < m - 1; ki++)

{ // проверка индекса строки

if (ki == i)

di = 1;

dj = 0;

for (kj = 0; kj < m - 1; kj++)

{ // проверка индекса столбца

if (kj == j) dj = 1;

p[ki, kj] = matrix[ki + di, kj + dj];

}

}

return p;

}

//Получение транспонированной матрицы

static public double[,] FindTransparentMatrix(double[,] matrix)

{

int rows = matrix.GetUpperBound(0) + 1;

int columns = matrix.Length / rows;

double temp;

for (int i = 0; i < rows; i++)

for (int j = 0; j < columns; j++)

{

if (i > j)

{

temp = matrix[i, j];

matrix[i, j] = matrix[j, i];

matrix[j, i] = temp;

}

}

return matrix;

}

}

}

1. Класс **EquationsSystem** содержит в себе систему уравнений, подающейся в виде двумерного массива.

using System;

using Expr = MathNet.Symbolics.SymbolicExpression;

namespace Lab\_05

{

class EquationsSystem

{

public Expr[,] Functions { get; set; }

public EquationsSystem(Expr[,] expr)

{

Functions = expr;

}

public void Print()

{

for (int i = 0; i < Functions.GetLength(0); i++)

{

for (int j = 0; j < Functions.GetLength(1); j++)

{

if(j != Functions.GetLength(1) - 1)

Console.Write($"({Functions[i, j]}) + ");

else

Console.Write($"({Functions[i, j]})");

}

Console.WriteLine(" = 0");

}

Console.WriteLine();

}

}

}

1. В классе **Newton** реализован метод Ньютона для решений системы нелинейных уравнений:

using System;

using System.Collections.Generic;

using System.Linq;

using MathNet.Symbolics;

using Expr = MathNet.Symbolics.SymbolicExpression;

namespace Lab\_05

{

class Newton

{

//Данная константа необходима, чтобы цикл в методе Solve

//не уходил в некоторых случаях в бесконечный цикл

const int maxCounter = 20;

private double[,] \_prev\_vector;

private double[,] \_curr\_vector;

private double[,] \_curr\_func\_value;

private double[,] \_numJac;

private double[,] \_revJacobian;

#region Properties

public EquationsSystem Equations { get; set; }

//Точность вычислений

public double Eps { get; set; }

//Словарь начальных значений для каждой переменной

public Dictionary<string, FloatingPoint> X { get; set; }

//Якобиан

public Expr[,] Jacobian { get; set; }

//Счетчик итераций

public int Counter { get; set; }

#endregion

public Newton(EquationsSystem eq, Dictionary<string, FloatingPoint> values, double e)

{

Equations = eq;

Eps = e;

X = values;

Counter = 0;

//Все свойства якобиана вычисляются при создании объекта Newton

Jacobian = FindJacobian();

}

#region Methods

private Expr[,] FindJacobian()

{

Expr[,] jac = new Expr[Equations.Functions.GetLength(0), Equations.Functions.GetLength(1)];

for (int i = 0; i < jac.GetLength(0); i++)

{

for (int j = 0; j < jac.GetLength(1); j++)

{

//Взятие производной из каждой функции

jac[i, j] = Equations.Functions[i, j].Differentiate($"x{j + 1}");

}

}

return jac;

}

private double[,] JacobianToDoubleFormat(Dictionary<string, FloatingPoint> values)

{

double[,] numJac = new double[Jacobian.GetLength(0), Jacobian.GetLength(1)];

for (int i = 0; i < Jacobian.GetLength(0); i++)

{

for (int j = 0; j < Jacobian.GetLength(1); j++)

{

//Перевод из FloatingPoint в double

numJac[i, j] = double.Parse(Jacobian[i, j].Evaluate(values).RealValue.ToString().Replace(".", ","));

}

}

return numJac;

}

private double[,] FindRevJacobian(double[,] numJac)

{

double[,] revJac = Matrix.CalculateInverseMatrix(numJac);

return revJac;

}

public void Solve()

{

double max;

//вектор, в который записывается текущее приближение

\_curr\_vector = new double[X.Values.Count, 1];

//вектор, в котором хранится предыдущее приближение

\_prev\_vector = new double[X.Values.Count, 1];

//Создание временного словаря и присвоение ему начальных значений

var temp = X;

do

{

//Увеличение счетчика итераций

Counter++;

//Составление численного Якобиана из обновленных значений перменных

\_numJac = JacobianToDoubleFormat(temp);

//Нахождение обратной матрицы Якобиана

\_revJacobian = FindRevJacobian(\_numJac);

//Обновление матрицы приближенных значений предыдущего шага

for (int i = 0; i < temp.Count; i++)

{

var elem = temp.ElementAt(i);

\_prev\_vector[i, 0] = double.Parse(elem.Value.RealValue.ToString().Replace(".", ","));

}

//Матрица числовых значений начальных функций

\_curr\_func\_value = new double[Equations.Functions.GetLength(1), 1];

//Расчет значений для матрицы функций

for (int i = 0; i < Equations.Functions.GetLength(0); i++)

{

for (int j = 0; j < Equations.Functions.GetLength(1); j++)

{

\_curr\_func\_value[i, 0] += double.Parse(Equations.Functions[i, j].Evaluate(temp).RealValue.ToString().Replace(".", ","));

}

}

//Расчет дельта X

var delta\_x = Matrix.MatrixMultiply(\_revJacobian, \_curr\_func\_value);

//Получение нового приближения

\_curr\_vector = Matrix.MatrixSubtraction(\_prev\_vector, delta\_x);

max = Math.Abs(delta\_x[0, 0]);

temp = new Dictionary<string, FloatingPoint>();

//Вычисление максимального значения в массиве delta\_x и обновление словаря temp

for (int i = 0; i < X.Count; i++)

{

if (max < Math.Abs(delta\_x[i, 0]))

max = Math.Abs(delta\_x[i, 0]);

var elem = X.ElementAt(i);

temp.Add(elem.Key, Expr.Parse(\_curr\_vector[i, 0].ToString().Replace(",", ".")).RealNumberValue);

}

} while (max > Eps && Counter != maxCounter);

Console.WriteLine($"Итоговое количество итераций: {Counter}");

Console.WriteLine("Ответ: ");

Matrix.PrintMatrix(\_curr\_vector);

Console.WriteLine("\n");

}

#endregion

}

}

1. В классе Program приведены примеры систем уравнений:

using System;

using System.Collections.Generic;

using MathNet.Symbolics;

using Expr = MathNet.Symbolics.SymbolicExpression;

namespace Lab\_05

{

class Program

{

static void Main(string[] args)

{

//Используемые переменныые

var x1 = Expr.Variable("x1");

var x2 = Expr.Variable("x2");

var x3 = Expr.Variable("x3");

//Пример 1

var values1 = new Dictionary<string, FloatingPoint>

{

{ "x1", 0.9},

{ "x2", 0.5}

};

var f1 = (x1 \* x1);

var f2 = (x2 \* x2 - 1);

var f3 = (x1 \* x1 \* x1);

var f4 = (- x2);

Expr[,] functions1 = new Expr[,] { { f1, f2 }, { f3, f4 } };

var system1 = new EquationsSystem(functions1);

system1.Print();

var newton1 = new Newton(system1, values1, 0.0001);

newton1.Solve();

//Пример 2

var values2 = new Dictionary<string, FloatingPoint>

{

{"x1", 0.5 },

{"x2", 0.5 },

{"x3", 0.5 }

};

f1 = (x1 \* x1);

f2 = (x2 \* x2);

f3 = (x3 \* x3 - 1);

f4 = (2 \* x1 \* x1);

var f5 = f2;

var f6 = (-4 \* x3);

var f7 = (3 \* x1 \* x1);

var f8 = (-4 \* x2);

var f9 = (x3 \* x3);

Expr[,] functions2 = new Expr[,] { { f1, f2, f3 }, { f4, f5, f6 }, {f7, f8, f9 } };

var system2 = new EquationsSystem(functions2);

system2.Print();

var newton2 = new Newton(system2, values2, 0.005);

newton2.Solve();

Console.ReadLine();

}

}

}

1. Результат работы программы:

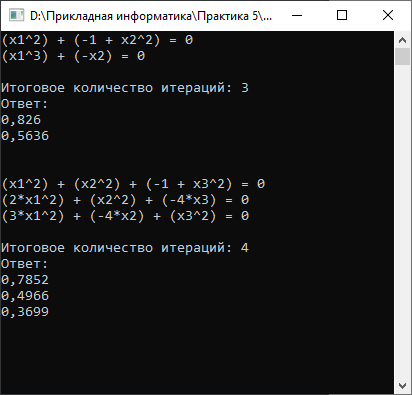


Рисунок 2 – Решение систем, заданных в программе

**Вывод:**

В ходе данной лабораторной работы был реализован метод Ньютона для решения системы нелинейных уравнений.