

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Системы автоматизированного проектирования | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Разработка программных систем | | |
| Курс | II | | | Группа | 423 |

Отчёт по контрольной работе №3

Вариант № 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 423 |  |  |  | Ефремов Иван Андреевич |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Макарук Роман Валерьевич |
|  |  |  |  | Федин Алексей Константинович |

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc134647499)

[2 Исходные данные 3](#_Toc134647500)

[3 Особые ситуации 3](#_Toc134647501)

[4 Математические методы и алгоритмы решения задач 4](#_Toc134647502)

[5 Форматы представления данных 4](#_Toc134647503)

[6 Структура программы 5](#_Toc134647504)

[7 Блок-схема алгоритма решения задачи 9](#_Toc134647505)

[8 Описание хода выполнения 13](#_Toc134647506)

[9 Результаты работы программы 14](#_Toc134647507)

[10 Выводы по заданию 16](#_Toc134647508)

[11 Исходный код полученного программного решения 16](#_Toc134647509)

## 1 Постановка задачи

Необходимо написать приложение с использованием технологии WinForms для построения графика функции и вывода таблицы значений функции. Пользователь задает правую и левую границу, шаг, коэффициенты (при их наличии). При невозможности построить график функции в заданном интервале пользователю выдается предупреждение об этом с предложением сменить границы построения. Если график функции из-за коэффициентов вырождается в точку или не может быть построен пользователь также видит предупреждение.

Заданная функция:

## 2 Исходные данные

Исходные данные состоят из границ построения функции, коэффициентов функции и шага, введенных пользователем с клавиатуры в форму.

## 3 Особые ситуации

Необходимо рассмотреть следующие особые ситуации:

-если при запросе числа пользователь ввёл другие символы, то программа укажет на необходимость ввода целых чисел и попросит повторить ввод.

## 4 Математические методы и алгоритмы решения задач

Согласно постановке задачи, для получения необходимых результатов будут использоваться операторы “>”, “≥”, “<”, “≤”, “=”, “+”.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 Форматы представления данных Таблица 1 - Основные переменные, используемые в программе   |  |  |  | | --- | --- | --- | | Имя | Тип | Описание | | x | double[] | Массив аргументов функции | | y | double[] | Массив значений функции при соответствующих аргументах | | points | Point[] | Точки функции, преобразованные для отображения в графике |    6 Структура программы Таблица 2 - Модули программы   |  |  | | --- | --- | | Название модуля | Описание | | Program | Запуск основной формы приложения | | MainForm | Логика и структура основной формы | | TableForm | Логика и структура формы, содержащей таблицу значений | | FunctionPoints | Вычисление точек функции | |  |

Таблица 3 – Основные функции модуля MainForm

|  |  |
| --- | --- |
| Имя | Описание |
| DrawAxises | Процедура, рисующая оси и координатные отрезки на плоскости |
| DrawGraph | Процедура, рисующая график функции из преобразованных точек |
| CalculateGraphPoints | Функция, преобразовывающая аргументы и значения функции, в точки, пригодные для отрисовки на плоскости |
| ValidateValues | Функция, проверяющая корректность введенных значений коэффициентов, шага и границ |

Таблица 4 – Функции модуля TableForm

|  |  |
| --- | --- |
| Имя | Описание |
| TableForm\_Load | Процедура, заполняющая таблицу в форме аргументами и значениями функции |

Таблица 5 – Функции модуля FunctionPoints

|  |  |
| --- | --- |
| Имя | Описание |
| GenerateX | Функция, генерирующая аргументы функции, согласно введенным границам и шагу |
| GenerateY | Функция, генерирующая значения функции, по заданному массиву аргументов и коэффициентам |

## 7 Блок-схема алгоритма решения задачи

На рисунке 1 представлена блок-схема работы программы.



Рисунок 1 – Блок-схема работы программы

## 8 Описание хода выполнения

Во время работы, программа запросит у пользователя ввести коэффициенты функции, границы вычисления и шаг. Затем пользователь должен нажать на кнопку «Draw Graph». После этого на плоскости в правой части формы построится график функции, согласно введенным значениям. Следом, пользователь может нажать на кнопку «Show coordinates», которая откроет форму с таблицей аргументов и значений функции.

## 9 Результаты работы программы

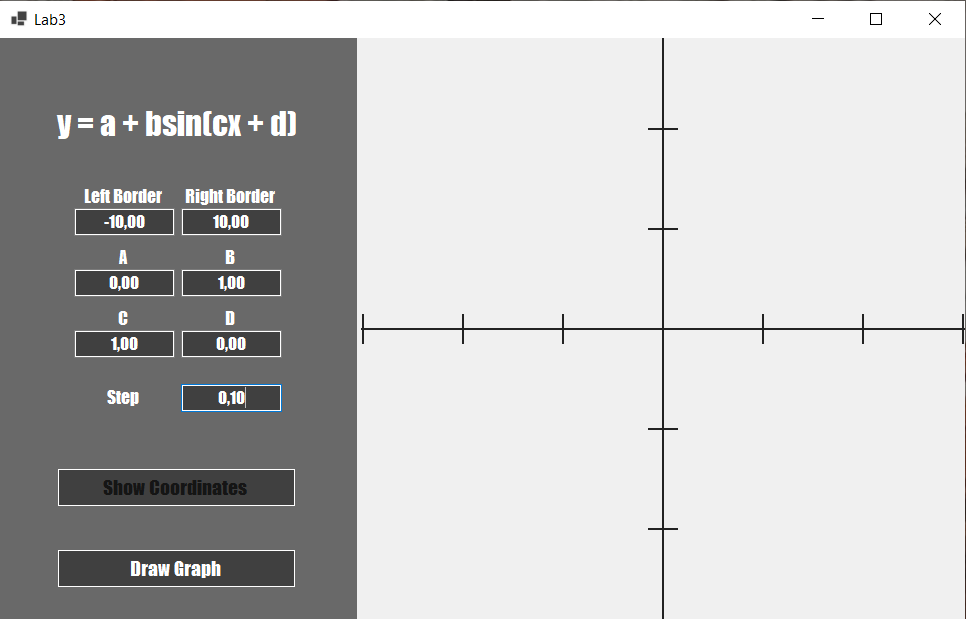


Рисунок 2 – Экранная копия окна программы сразу после запуска

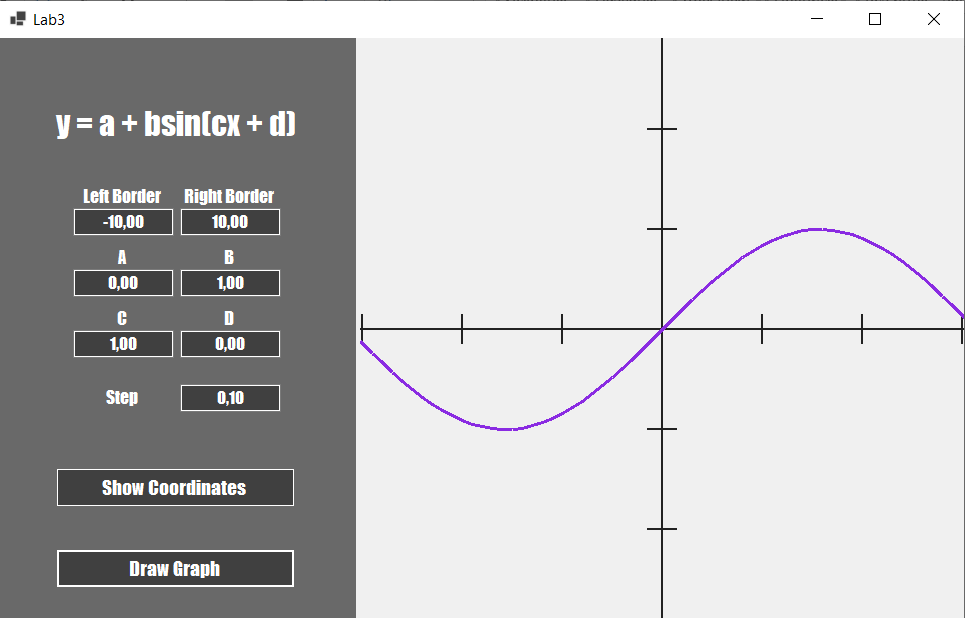


Рисунок 3 – Экранная копия окна программы после построения графика со стандартными значениями

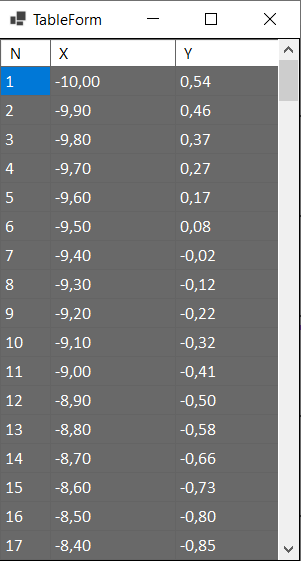


Рисунок 4 – Экранная копия окна формы с таблицей аргументов и значений функции

## 10 Выводы по заданию

В ходе выполнения задания были изучены основы работы с языком С# и работы с системой WinForms. Были получены навыки создания приложений с полноценным графическим интерфейсом.

## 11 Исходный код полученного программного решения

Файл Program.cs:

namespace Lab\_3

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

// To customize application configuration such as set high DPI settings or default font,

// see https://aka.ms/applicationconfiguration.

ApplicationConfiguration.Initialize();

Application.Run(new GraphForm());

}

}

}

Файл FunctionPoints.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_3

{

public static class FunctionPoints

{

public static double[] GenerateX(double step, double leftBorder, double rightBorder)

{

int size = Convert.ToInt32((rightBorder-leftBorder) / step) + 1;

double[] abscissa = new double[size];

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

abscissa[i] = leftBorder + i \* step;

return abscissa;

}

public static double[] GenerateY(double[] x, double[] odds)

{

double[] ordinate = new double[x.Length];

for(int i = 0; i < x.Length; i++)

ordinate[i] = Math.Round(odds[0] + odds[1] \* Math.Sin(odds[2] \* x[i] + odds[3]), 2);

return ordinate;

}

}

}

Файл MainForm.cs:

namespace Lab\_3

{

public partial class GraphForm : Form

{

public GraphForm()

{

InitializeComponent();

}

private const int pixelsPerSection = 100;

private double step = 0.1;

private double a = 0;

private double b = 1;

private double c = 1;

private double d = 0;

private double leftBorderVal = -10;

private double rightBorderVal = 10;

private bool isGraphShown = false;

private bool isValidData = true;

double[] x = new double[0];

double[] y = new double[0];

protected void DrawAxises()

{

graphPlane.Refresh();

Graphics graphic = graphic = graphPlane.CreateGraphics();

Pen pen = new Pen(Color.FromArgb(34, 34, 34), 2f);

int height = graphPlane.Height;

int width = graphPlane.Width;

int x = 0;

int y = graphPlane.Size.Height / 2;

graphic.DrawLine(pen, x, y, width, y);

graphic.DrawLine(pen, width / 2, 0, width / 2, height);

for (int i = (width / 2) % pixelsPerSection; i < width; i += pixelsPerSection)

graphic.DrawLine(pen, i, y + (pixelsPerSection / 10 + 5), i, y - (pixelsPerSection / 10 + 5));

for (int i = y % pixelsPerSection; i < height; i += pixelsPerSection)

graphic.DrawLine(pen, width / 2 + (pixelsPerSection / 10 + 5), i, width / 2 - (pixelsPerSection / 10 + 5), i);

}

protected void DrawGraph()

{

int xPrev = Math.Max(0, graphPlane.Width / 2 + Convert.ToInt32(leftBorderVal \* pixelsPerSection));

int yPrev = graphPlane.Height / 2 - Convert.ToInt32(

a \* pixelsPerSection + b \* pixelsPerSection \* Math.Sin(

c \* (xPrev - graphPlane.Width / 2) / Convert.ToDouble(pixelsPerSection) + d)

);

Graphics graphic = graphic = graphPlane.CreateGraphics();

Pen pen = new Pen(Color.BlueViolet, 3f);

Point[] points = CalculateGraphPoints();

for (int i = 0; i < points.Length && xPrev < Math.Min(graphPlane.Width, graphPlane.Width / 2 + Convert.ToInt32(rightBorderVal \* pixelsPerSection)); i++)

{

if (points[i].X <= xPrev) continue;

graphic.DrawLine(pen, xPrev, yPrev, points[i].X, points[i].Y);

xPrev = points[i].X;

yPrev = points[i].Y;

}

}

private Point[] CalculateGraphPoints()

{

x = FunctionPoints.GenerateX(step, leftBorderVal, rightBorderVal);

y = FunctionPoints.GenerateY(x, [a, b, c, d]);

Point[] points = new Point[x.Length];

for (int i = 0; i < x.Length; i++)

{

points[i].X = graphPlane.Width / 2 + Convert.ToInt32(x[i] \* pixelsPerSection);

points[i].Y = graphPlane.Height / 2 - Convert.ToInt32(y[i] \* pixelsPerSection);

}

return points;

}

private void grahPlane\_Resize(object sender, EventArgs e)

{

DrawAxises();

if (isGraphShown) DrawGraph();

}

private void GraphForm\_Shown(object sender, EventArgs e)

{

DrawAxises();

}

private void drawGraphBtn\_Click(object sender, EventArgs e)

{

DrawAxises();

if (CheckGraphOdds())

{

DrawGraph();

isGraphShown = true;

showCoordinatesBtn.Enabled = true;

}

}

private string FormatTextField(string text)

{

if (text.IndexOf(',') == -1)

return text + ",00";

if (text.Length - 1 - text.IndexOf(',') < 2)

return text + '0';

return text;

}

private bool CheckGraphOdds()

{

isGraphShown = false;

if (!isValidData) return false;

if (Math.Abs(a) > graphPlane.Height / 2 / step)

{

MessageBox.Show("Change \"A\" value, either way graph won't be seen with current window size", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return false;

}

if (leftBorderVal > graphPlane.Width / 2 / step)

{

MessageBox.Show("Make \"Left Border\" value less, either way graph won't be seen with current window size", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return false;

}

if (rightBorderVal < -graphPlane.Width / 2 / step)

{

MessageBox.Show("Make \"Right Border\" value greater, either way graph won't be seen with current window size", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return false;

}

return true;

}

private int ValidateValues(string? text, ref double value)

{

if (string.IsNullOrEmpty(text))

return 0;

else if (!double.TryParse(text, out value))

return -1;

else if (text.IndexOf(',') != -1 && text.Length - 1 - text.IndexOf(',') > 2)

return 2;

return 1;

}

private void leftBorderValue\_Check(object sender, EventArgs e)

{

switch (ValidateValues(leftBorderValue.Text, ref leftBorderVal))

{

case 0:

MessageBox.Show("Field must not be empty!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case -1:

MessageBox.Show("Invalid data!\nNumbers only!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case 2:

MessageBox.Show("Programm accepts values up to 2 decimal places!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

default:

if (leftBorderVal < rightBorderVal)

{

leftBorderValue.Text = FormatTextField(leftBorderValue.Text);

isValidData = true;

return;

}

MessageBox.Show("Left border must be less than right!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

}

isValidData = false;

leftBorderValue.Focus();

leftBorderValue.Select(0, leftBorderValue.Text.Length);

}

private void rightBorderValue\_Check(object sender, EventArgs e)

{

switch (ValidateValues(rightBorderValue.Text, ref rightBorderVal))

{

case 0:

MessageBox.Show("Field must not be empty!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case -1:

MessageBox.Show("Invalid data!\nIntegers only!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case 2:

MessageBox.Show("Programm accepts values up to 2 decimal places!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

default:

if (rightBorderVal > leftBorderVal)

{

rightBorderValue.Text = FormatTextField(rightBorderValue.Text);

isValidData = true;

return;

}

MessageBox.Show("Right border must be greater than left!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

}

isValidData = false;

rightBorderValue.Focus();

rightBorderValue.Select(0, rightBorderValue.Text.Length);

}

private void aValue\_Check(object sender, EventArgs e)

{

switch (ValidateValues(aValue.Text, ref a))

{

case 0:

MessageBox.Show("Field must not be empty!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case -1:

MessageBox.Show("Invalid data!\nNumbers only!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case 2:

MessageBox.Show("Programm accepts values up to 2 decimal places!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

default:

aValue.Text = FormatTextField(aValue.Text);

isValidData = true;

return;

}

isValidData = false;

aValue.Focus();

aValue.Select(0, aValue.Text.Length);

}

private void bValue\_Check(object sender, EventArgs e)

{

switch (ValidateValues(bValue.Text, ref b))

{

case 0:

MessageBox.Show("Field must not be empty!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case -1:

MessageBox.Show("Invalid data!\nNumbers only!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case 2:

MessageBox.Show("Programm accepts values up to 2 decimal places!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

default:

bValue.Text = FormatTextField(bValue.Text);

isValidData = true;

return;

}

isValidData = false;

bValue.Focus();

bValue.Select(0, bValue.Text.Length);

}

private void cValue\_Check(object sender, EventArgs e)

{

switch (ValidateValues(cValue.Text, ref c))

{

case 0:

MessageBox.Show("Field must not be empty!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case -1:

MessageBox.Show("Invalid data!\nNumbers only!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case 2:

MessageBox.Show("Programm accepts values up to 2 decimal places!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

default:

cValue.Text = FormatTextField(cValue.Text);

isValidData = true;

return;

}

isValidData = false;

cValue.Focus();

cValue.Select(0, cValue.Text.Length);

}

private void dValue\_Check(object sender, EventArgs e)

{

switch (ValidateValues(dValue.Text, ref d))

{

case 0:

MessageBox.Show("Field must not be empty!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case -1:

MessageBox.Show("Invalid data!\nNumbers only!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case 2:

MessageBox.Show("Programm accepts values up to 2 decimal places!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

default:

dValue.Text = FormatTextField(dValue.Text);

isValidData = true;

return;

}

isValidData = false;

dValue.Focus();

dValue.Select(0, dValue.Text.Length);

}

private void stepValue\_Check(object sender, EventArgs e)

{

switch (ValidateValues(stepValue.Text, ref step))

{

case 0:

MessageBox.Show("Field must not be empty!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case -1:

MessageBox.Show("Invalid data!\nNumbers only!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

case 2:

MessageBox.Show("Programm accepts values up to 2 decimal places!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

break;

default:

if (step > 1)

MessageBox.Show("Step value can't be greater than 1!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

else if (step <= 0)

MessageBox.Show("Step value can't be negative or 0!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);

else

{

stepValue.Text = FormatTextField(stepValue.Text);

isValidData = true;

return;

}

break;

}

isValidData = false;

stepValue.Focus();

stepValue.Select(0, stepValue.Text.Length);

}

private void showCoordinatesBtn\_Click(object sender, EventArgs e)

{

Form table = new TableForm(x, y);

table.ShowDialog();

}

}

}

Файл TableForm.cs:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab\_3

{

public partial class TableForm : Form

{

public TableForm()

{

InitializeComponent();

}

public TableForm(double[] x, double[] y)

{

InitializeComponent();

abscisse = x;

ordinate = y;

}

public double[] abscisse = new double[0];

public double[] ordinate = new double[0];

private void TableForm\_Load(object sender, EventArgs e)

{

coordinateTable.Rows.Clear();

int size = abscisse.Length;

coordinateTable.RowCount = size;

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

{

coordinateTable.Rows[i].Cells[0].Value = i + 1;

coordinateTable.Rows[i].Cells[1].Value = abscisse[i];

coordinateTable.Rows[i].Cells[2].Value = ordinate[i];

}

}

}

}

Файл Lab3Tests.cs:

using Lab\_3;

namespace Lab3\_Tests

{

[TestClass]

public class Lab3Tests

{

private bool CompareArrays(double[] expected, double[] actual)

{

const double epsilon = 0.01;

for (int i = 0; i < expected.Length; i++)

{

if (Math.Abs(expected[i]) - Math.Abs(actual[i]) >= epsilon) return false;

}

return true;

}

double[] odds = new double[4];

double[] x = FunctionPoints.GenerateX(0.5, 0, 10);

double[] y = new double[21];

double[] expected = new double[21];

[TestMethod]

/\*

Тестирование с шагом 0.5 и коэффициентами:

A = 3; B = 1.5; C = 0.78; D = 2

\*/

public void Test1()

{

odds = [3, 1.5, 0.78, 2];

expected = [4.36, 4.02, 3.53, 2.96, 2.39, 1.92, 1.6, 1.5, 1.62, 1.95, 2.44, 3.01, 3.58, 4.06, 4.39, 4.5, 4.39, 4.07, 3.59, 3.02, 2.45];

y = FunctionPoints.GenerateY(x, odds);

Assert.IsTrue(CompareArrays(expected, y));

}

[TestMethod]

/\*

Тестирование с шагом 0.5 и коэффициентами:

A = 5; B = 2; C = -1.04; D = -3

\*/

public void Test2()

{

odds = [5, 2, -1.04, -3];

expected = [4.72, 5.74, 6.56, 6.98, 6.87, 6.26, 5.32, 4.3, 3.46, 3.03, 3.12, 3.7, 4.63, 5.66, 6.51, 6.96, 6.9, 6.33, 5.41, 4.38, 3.52];

y = FunctionPoints.GenerateY(x, odds);

Assert.IsTrue(CompareArrays(expected, y));

}

[TestMethod]

/\*

Тестирование с шагом 0.5 и коэффициентами:

A = 2; B = 3.14; C = 0.52; D = 1

\*/

public void Test3()

{

odds = [2, 3.14, 0.52, 1];

expected = [4.64, 4.99, 5.14, 5.07, 4.8, 4.34, 3.72, 2.99, 2.19, 1.38, 0.61, -0.07, -0.6, -0.97, -1.13, -1.08, -0.83, -0.39, 0.22, 0.94, 1.74];

y = FunctionPoints.GenerateY(x, odds);

Assert.IsTrue(CompareArrays(expected, y));

}

[TestMethod]

/\*

Тестирование с шагом 0.5 и коэффициентами:

A = 4; B = 0.78; C = 1.57; D = 0

\*/

public void Test4()

{

odds = [4, 0.78, 1.57, 0];

expected = [4.0, 4.55, 4.78, 4.55, 4.0, 3.45, 3.22, 3.45, 4.0, 4.55, 4.78, 4.55, 4.0, 3.45, 3.22, 3.45, 4.0, 4.55, 4.78, 4.56, 4.01];

y = FunctionPoints.GenerateY(x, odds);

Assert.IsTrue(CompareArrays(expected, y));

}

[TestMethod]

/\*

Тестирование с шагом 0.5 и коэффициентами:

A = 6; B = 1.04; C = 0; D = 4

\*/

public void Test5()

{

odds = [6, 1.04, 0, 4];

expected = [5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21, 5.21];

y = FunctionPoints.GenerateY(x, odds);

Assert.IsTrue(CompareArrays(expected, y));

}

}

}