# Содержание

[Содержание 3](#_Toc10281805)

[Введение 4](#_Toc10281806)

[1. Математическая формулировка задачи 5](#_Toc10281807)

[2. Описание программы 6](#_Toc10281808)

[2.1 Структурная схема программы 6](#_Toc10281809)

[2.2 Описание разработанного класса 6](#_Toc10281810)

[2.3 Основные возможности программы 7](#_Toc10281811)

[3. Руководство пользователя 8](#_Toc10281812)

[4. Методика испытаний 10](#_Toc10281813)

[Заключение 12](#_Toc10281814)

[Литература 13](#_Toc10281815)

[Приложение А. 14](#_Toc10281816)

[SplashForm.cs 14](#_Toc10281817)

[Form1.cs 14](#_Toc10281818)

[Client.cs 18](#_Toc10281819)

[Painter.cs 19](#_Toc10281820)

[Server.cs 20](#_Toc10281821)

# Введение

Ма́трица — математический объект, записываемый в виде прямоугольной таблицы элементов кольца или поля (например, целых, действительных или комплексных чисел), которая представляет собой совокупность строк и столбцов, на пересечении которых находятся её элементы. Количество строк и столбцов задает размер матрицы. Хотя исторически рассматривались, например, треугольные матрицы, в настоящее время говорят исключительно о матрицах прямоугольной формы, так как они являются наиболее удобными и общими.

Матрицы широко применяются в математике для компактной записи систем линейных алгебраических или дифференциальных уравнений. В этом случае количество строк матрицы соответствует числу уравнений, а количество столбцов — количеству неизвестных. В результате решение систем линейных уравнений сводится к операциям над матрицами.

Для матрицы определены следующие алгебраические операции:

* сложение матриц, имеющих один и тот же размер;
* умножение матриц подходящего размера (матрицу, имеющую n столбцов, можно умножить справа на матрицу, имеющую n строк);
* в том числе умножение на матрицу вектора (по обычному правилу матричного умножения; вектор является в этом смысле частным случаем матрицы);
* умножение матрицы на элемент основного кольца или поля (то есть скаляр).

Относительно сложения матрицы образуют абелеву группу; если же рассматривать ещё и умножение на скаляр, то матрицы образуют модуль над соответствующим кольцом (векторное пространство над полем). Множество квадратных матриц замкнуто относительно матричного умножения, поэтому квадратные матрицы одного размера образуют ассоциативное кольцо с единицей относительно матричного сложения и матричного умножения.

Доказано, что каждому линейному оператору, действующему в n-мерном линейном пространстве, можно сопоставить единственную квадратную матрицу порядка n; и обратно — каждой квадратной матрице порядка n может быть сопоставлен единственный линейный оператор, действующий в этом пространстве. Свойства матрицы соответствуют свойствам линейного оператора. В частности, собственные числа матрицы — это собственные числа оператора, отвечающие соответствующим собственным векторам.

То же можно сказать о представлении матрицами билинейных (квадратичных) форм.

В математике рассматривается множество различных типов и видов матриц. Таковы, например, единичная, симметричная, кососимметричная, верхнетреугольная (нижнетреугольная) и т. п. матрицы.

Особое значение в теории матриц занимают всевозможные нормальные формы, то есть канонический вид, к которому можно привести матрицу заменой координат. Наиболее важной (в теоретическом значении) и проработанной является теория жордановых нормальных форм. На практике, однако, используются такие нормальные формы, которые обладают дополнительными свойствами, например, устойчивостью.

Определи́тель (или детермина́нт) — одно из основных понятий линейной алгебры. Определитель квадратной матрицы A размеров n\*n, заданной над коммутативным кольцом R, является элементом кольца R, вычисляемым по формуле, приведённой ниже.

Он «определяет» свойства матрицы A. В частности, матрица A обратима тогда и только тогда, когда её определитель является обратимым элементом кольца R.

В случае, когда R — поле, определитель матрицы A равен нулю тогда и только тогда, когда ранг матрицы A меньше n или когда системы строк и столбцов матрицы A являются линейно зависимыми.

Определитель матрицы А обозначается как det(A), |A| или .

# 1. Математическая формулировка задачи

Во многих алгоритмах необходимо приводить матрицу к треугольному виду (выполнять триангуляцию матрицы) путем эквивалентных преобразований. К таким преобразованиям относятся:

* перестановка строк;
* прибавление к элементам одной строки соответствующих элементов другой строки, домноженной на константу.

Выполнение над матрицей эквивалентных преобразований не изменит значение определителя матрицы, а также корни системы уравнений, заданной матрицей. Построение треугольной матрицы является частью алгоритма Гаусса-Жордана решения СЛАУ, а также вычисления определителя методом Гаусса.

Для «обнуления» элементов i-того столбца матрицы достаточно ко всем строкам с номерами j = i+1, ... n прибавить i-тую строку, домноженную на -a[j][i]/a[i][i]. При выполнении такой операции может возникать деление на ноль если элемент на главной диагонали окажется равным нулю.

Такой алгоритм можно изобразить в виде следующей блок-схемы:.

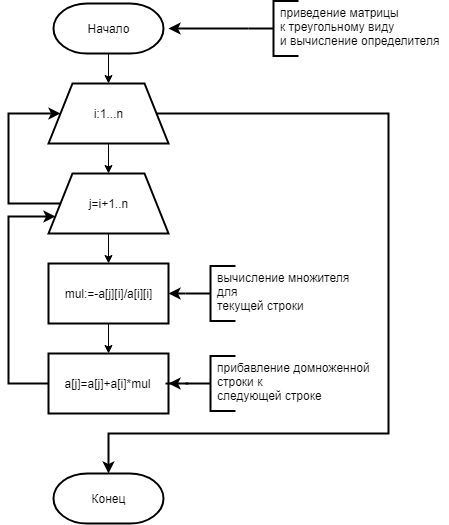


Рисунок 1.1 – Алгоритм вычисления определителся матрицы

# 2. Описание программы

## 2.1 Структурная схема программы

Структурно программа состоит из главного модуля, модулей описания вспомогательных форм, динамических библиотек, файлов помощи и обращений к другим программам. Связь модулей приведена на рисунке 2.1.

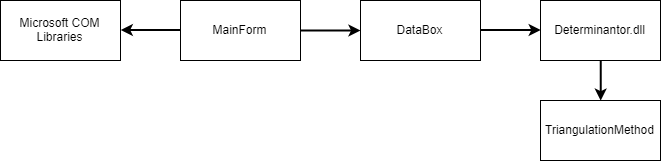


Рисунок 2.1 - Структурная схема связей модулей программы

## 2.2 Описание разработанного класса

В результате анализа предметной области для её описания и моделирования разработан класс пользовательский компонент DataBox, инкапсулирующий поля, методы и свойства, применимые к рассматриваемому объекту:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Windows.Forms;

using DeterminantCalculator;

namespace Main\_solution

{

public partial class DataBox : UserControl

{

public delegate void DeterminantStateHandler(double? message);

public event DeterminantStateHandler DeterminantCalculated;

private static ManualResetEvent \_event;

private TriangulationMethod \_calculator;

private TextBox \_texBox;

private Thread \_myThread;

private double? \_determinant;

public int Steps { get; private set; }

public DataBox()

{

InitializeComponent();

dataGridView.EditingControlShowing +=

DataGridView\_EditingControlShowing;

\_event = new ManualResetEvent(true);

}

public void ConnectTextBox(ref TextBox textBox)

{

\_texBox = textBox;

}

private static void DataGridView\_EditingControlShowing(object sender,

DataGridViewEditingControlShowingEventArgs e)

{

e.Control.KeyPress -= Column\_KeyPress;

if (e.Control is TextBox tb) tb.KeyPress += Column\_KeyPress;

}

private static void Column\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar)

&& e.KeyChar != '.'

&& e.KeyChar != '-')

e.Handled = true;

if (e.KeyChar == '.'

&& ((TextBox)sender).Text.IndexOf('.') > -1)

e.Handled = true;

if (e.KeyChar == '-'

&& ((TextBox)sender).Text.IndexOf('-') == 0)

e.Handled = true;

}

private void RestartCalculating()

{

progressBar1.Value = progressBar1.Minimum;

if (\_texBox == null)

{

\_calculator = new TriangulationMethod(ref dataGridView);

}

else

{

\_calculator = new TriangulationMethod(ref dataGridView, ref \_texBox);

}

\_calculator.MatrixChanged += UpdateDataGrid;

\_myThread = new Thread(Calculate)

{

Priority = ThreadPriority.Highest

};

\_myThread.Start();

}

public double CalcAll(int waitTime)

{

progressBar1.Value = progressBar1.Minimum;

\_calculator = new TriangulationMethod(ref dataGridView)

{

AutomaticMode = true

};

\_calculator.MatrixChanged += UpdateDataGrid;

\_determinant = \_calculator?.Calc(\_event,waitTime);

if (\_determinant != null)

{

DeterminantCalculated?.Invoke(\_determinant);

}

return (double)\_determinant;

}

public void NextStep()

{

if (\_myThread == null)

{

RestartCalculating();

}

else

{

if (!\_myThread.IsAlive)

{

\_myThread.Abort();

RestartCalculating();

}

}

MakeStep();

}

private static void MakeStep()

{

\_event?.Set();

}

public void SetSize(int size)

{

Random rnd = new Random();

dataGridView.UseWaitCursor = false;

dataGridView.Rows.Clear();

dataGridView.RowCount = size;

dataGridView.ColumnCount = size;

for (var str = 0; str < size; str++)

{

dataGridView.Rows[str].HeaderCell.Value = (str + 1).ToString();

for (var col = 0; col < size; col++)

{

dataGridView.Columns[col].HeaderText = (col + 1).ToString();

dataGridView[col, str].Value = rnd.Next(); ;

}

}

progressBar1.Minimum = 0;

progressBar1.Value = progressBar1.Minimum;

Steps = CountSteps(size);

progressBar1.Maximum = Steps;

}

private static int CountSteps(int size)

{

var counter = 0;

for (var str = 0; str < size; str++) counter += str;

return counter;

}

private void UpdateDataGrid(double[][] matrix)

{

if (matrix == null) return;

for (var row = 0; row < dataGridView.RowCount; row++)

for (var col = 0; col < dataGridView.ColumnCount; col++)

dataGridView[col, row].Value = Math.Round(matrix[row][col], 3);

dataGridView.Refresh();

++progressBar1.Value;

}

public void CopyToDataGrid(DataBox dataBox)

{

dataBox.dataGridView.RowCount = dataGridView.RowCount;

dataBox.dataGridView.ColumnCount = dataGridView.ColumnCount;

for (var row = 0; row < dataGridView.RowCount; row++)

{

dataBox.dataGridView.Rows[row].HeaderCell.Value = (row + 1).ToString();

for (var col = 0; col < dataGridView.ColumnCount; col++)

{

dataBox.dataGridView.Columns[col].HeaderText = (col + 1).ToString();

dataBox.dataGridView[col, row].Value = dataGridView[col, row].Value;

}

}

dataBox.progressBar1.Minimum = progressBar1.Minimum;

dataBox.progressBar1.Value = progressBar1.Minimum;

dataBox.Steps = Steps;

dataBox.progressBar1.Maximum = Steps;

}

private void Calculate()

{

CheckForIllegalCrossThreadCalls = false;

progressBar1.Value = progressBar1.Minimum;

dataGridView.UseWaitCursor = true;

dataGridView.ReadOnly = true;

if (\_event == null) return;

\_determinant = \_calculator?.Calc(\_event);

if (\_determinant != null)

{

DeterminantCalculated?.Invoke(\_determinant);

}

dataGridView.ReadOnly = false;

dataGridView.UseWaitCursor = false;

MessageBox.Show("Вычисление завершено.");

}

public double[][] CopyToArray()

{

var localMatrix = new LinkedList<double[]>();

for (var str = 0; str < dataGridView.Rows.Count; str++)

{

var rowValues = new LinkedList<double>();

for (var col = 0; col < dataGridView.Rows[str].Cells.Count; col++)

rowValues.AddLast(double.Parse(dataGridView.Rows[str].Cells[col].Value.ToString()));

localMatrix.AddLast(rowValues.ToArray());

}

return localMatrix.ToArray();

}

}

}

Описание и реализация класса структурно оформлено в классе. Используемые секции public и private определяют области видимости элементов класса. Так, видно, что все расчетные поля класса являются полностью закрытыми, а доступ к ним организуется с помощью методов и свойств.

Так же в нём реализованы события, которые могут обрабатывается выше по иерархии вызовов.

Ввод в матрицу защищён от некорректных данных, а так же оптимизирована за грузка больших таблиц данных.

## 2.3 Основные возможности программы

Программа начинается с отображением окна приветствия. После него появляется главное окно программы, на котором и выполняются все манипуляции:

* Запустить программу (начнётся отрисовка вычисление определителя);
* Запустить программу (в отлеьном потоке, что бы пошагово ознакомиться с вычислением);
* Открытие Helper;
* Запись данных в Word;
* Запись данных в Excel;
* Открытие PowerPoint;
* Сохранять данные после закрытия и загружать их после открытия.

Диалог с пользователем поддерживается с помощью главного и выпадающего из панели инструментов списка кнопок, обычных кнопок и других интерфейсных элементов.

Основная информация, а также помощь по использованию программы содержится в окне Helper. Всю необходимую литературу, теорию и помощь можно получить в данном приложении к программе.

# 3. Руководство пользователя

Для запуска нашего приложения необходимо запустить файл «Main solution.exe». При этом на экране появляется сначала окно загрузки программы, показанное на рисунке 3.1.

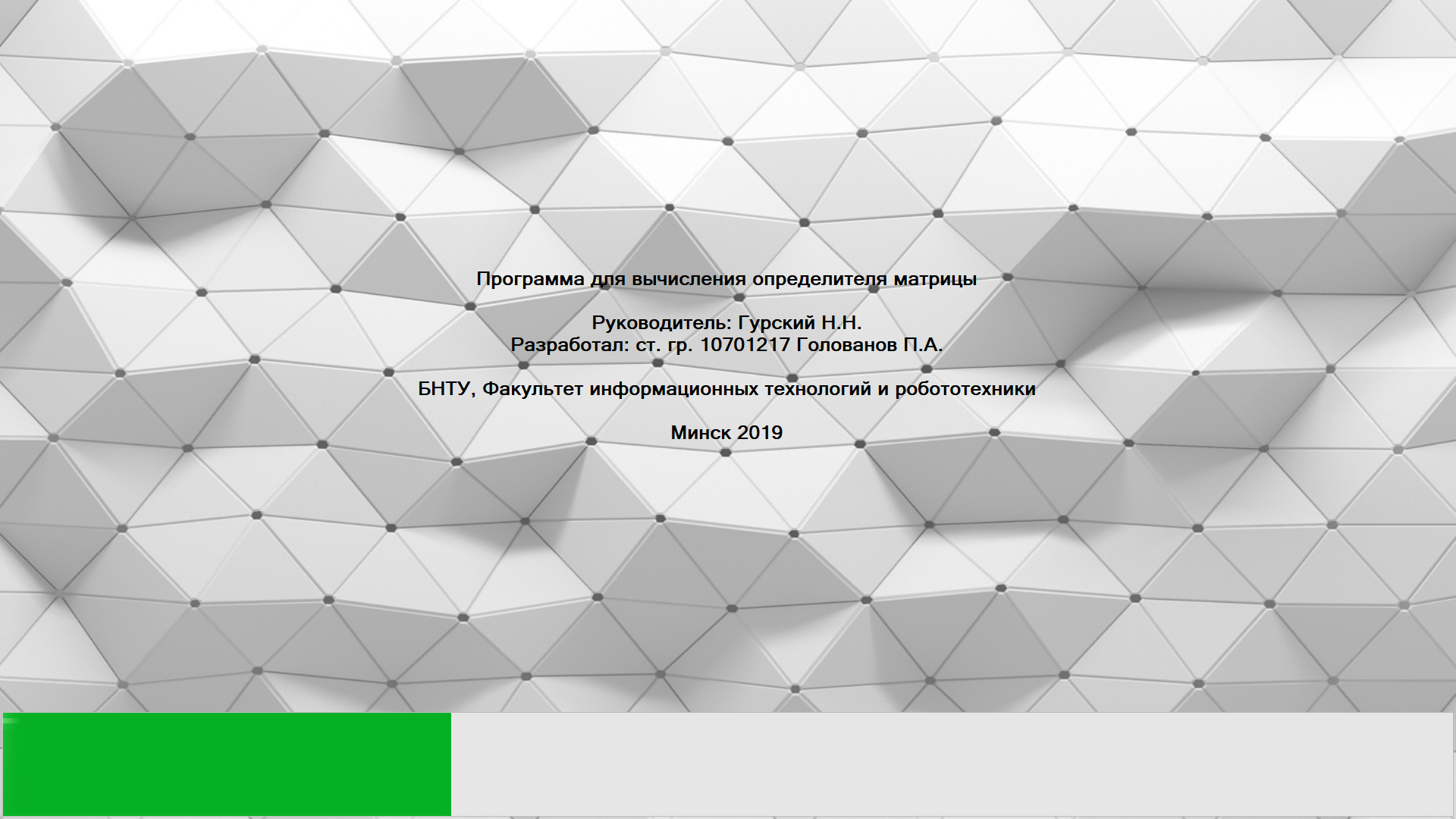


Рисунок 3.1 – Окно загрузки

После будет запущено главное окно, в котоом уже можно задать размер матицы и ввести в неё заначения и запустить вычисление автоматическое или пошаговое (рисунок 3.2):

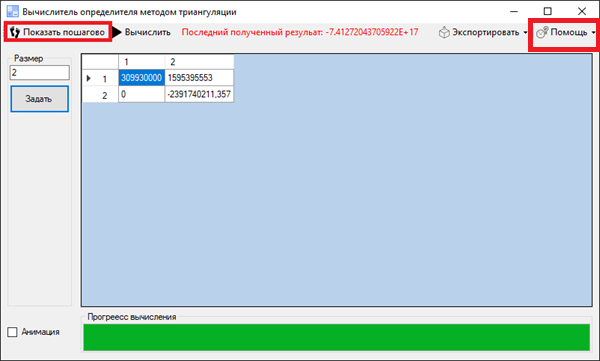


Рисунок 3.2 – «Главный экран приложения»

При нажатии на клавишы справки можно узнать о приложении подробнее (рисунок 3.3):

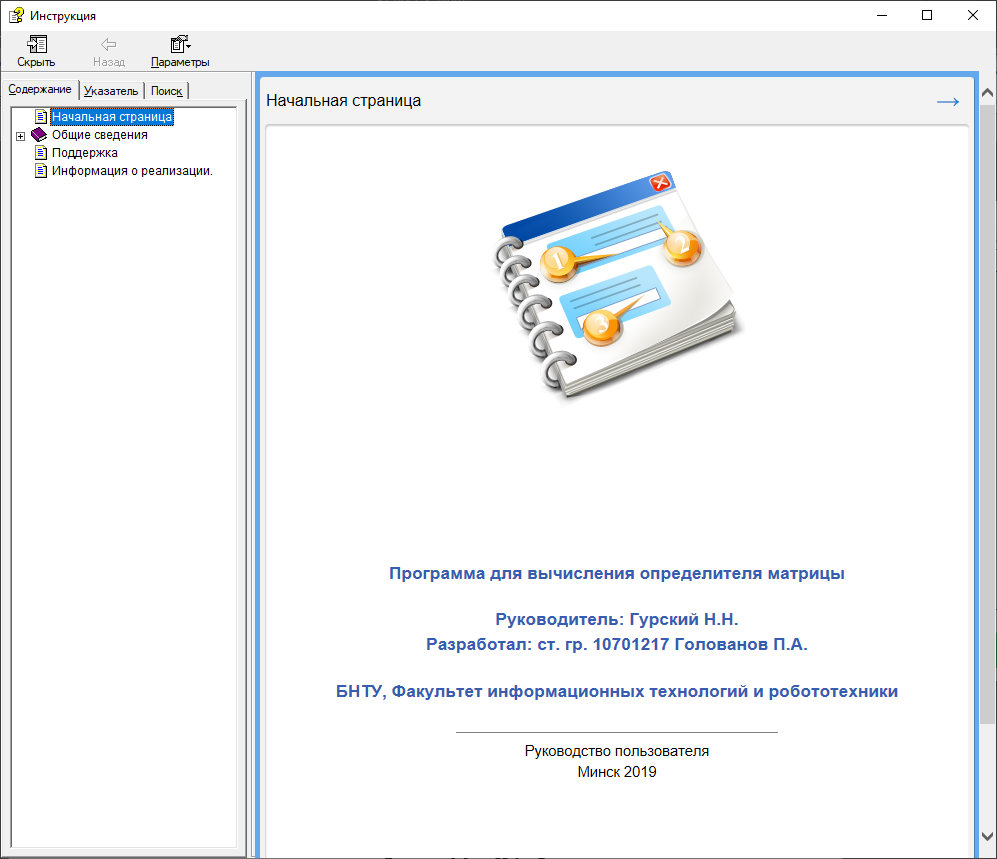


Рисунок 3.3 – «Инструкция»

Выход из программы осущестляется зыкрытием главного окна.

# 4. Методика испытаний

Целью проведения испытаний является проверка работоспособности (надежности) программы при различных условиях ее функционирования. Программа должна обеспечивать корректность ввода исходных данных (путем осуществления соответствующих проверок и информирования пользователя о возникших неточностях в работе), а также получение данных в виде ответа сервера. Для демонстрации работоспособности программы необходимо провести ряд испытаний с различными начальными условиями.

Испытания выполнялись путем увеличения обьёма входных данных.

Первое испытание проведем с максимально простыми начальными данными.

Результаты мы видим на рисунке 4.1.

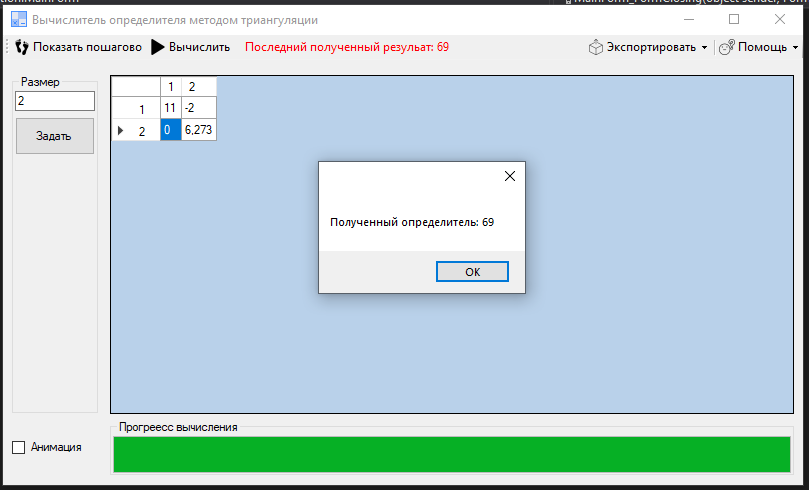


Рисунок 4.1 - Результат работы с начальными исходными данными

Теперь попробуем запустить изначально провальный тест:

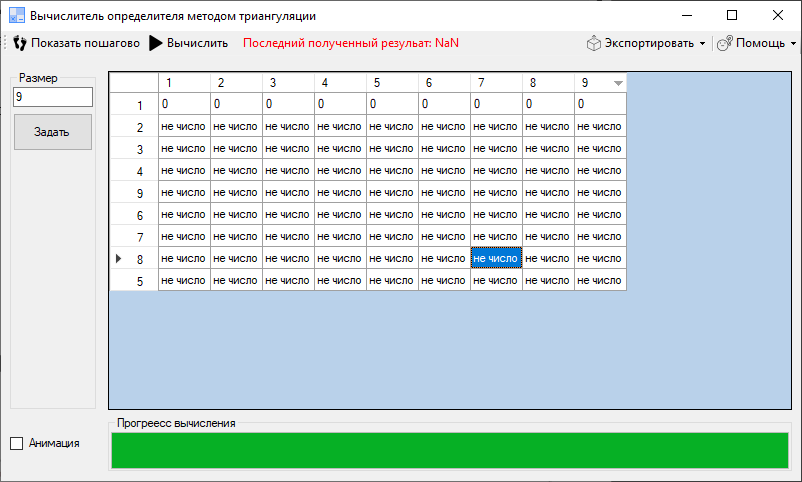


Рисунок 4.2 - Тестирование при других исходных данных

Таким образом, проведенное тестирование программы не выявило сбойных ситуаций и некорректностей в ее работе. Следует считать, что в целом программа протестирована, отвечает поставленным требованиям и вполне работоспособна.

# Заключение

В ходе выполнения курсовой работы:

- Были закреплены знания по курсу “Разработка приложений в визуальных средах”;

- Приобретен опыт при разработке объектно-ориентированных программ;

- Изучены принципы создания динамических библиотек;

- Изучены и реализованы способы создания клиент-серверных приложений на базе COM-технологий;

-Освоен механизм работы и создания событий.

- Освоены методы передачи данных между приложениями;

- Изучены возможности создания справочной системы.

При разработке программы использовался модульный принцип программирования. Такая структура программы позволяет в дальнейшем модифицировать отдельные её части без нарушения работоспособности и потери функциональности приложения. Программа имеет минималистичный, понятый и дружелюбный пользовательский интерфейс, обеспечивающий удобство в работе и обеспечивающий получение необходимой пользователю информации с наименьшими затратами времени.

# Литература

1. Албахари Б. и Д.. C# 6.0. Справочник. Полное описание языка. 2016 – 1040 стр.;
2. Кристиан Нейгел. C# 5.0 и платформа .NET 4.5 для профессионалов 2008 – 1440стр.;

# Приложение

## MainForm.cs

using System;

using System.Diagnostics;

using System.Drawing;

using System.Globalization;

using System.Threading;

using System.Windows.Forms;

namespace Main\_solution

{

public partial class MainForm : Form

{

private bool \_firstLoop;

private int \_waitTime;

private double? \_lastCalculatedResult;

public MainForm()

{

var prev = new Preview();

prev.ShowDialog();

InitializeComponent();

CheckForIllegalCrossThreadCalls = false;

\_firstLoop = true;

\_lastCalculatedResult = null;

animationTimer.Start();

\_waitTime = 0;

}

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

{

if (dataBox1 == null) return;

dataBox1.DeterminantCalculated += DrawResult;

var res = dataBox1.CalcAll(\_waitTime);

toolStripDropDownButton1.Enabled = true;

MessageBox.Show($"Полученный определитель: {res}");

}

private void DrawResult(double? determinant)

{

if (determinant != null)

{

ResultLabel.Visible = true;

toolStripDropDownButton1.Enabled = true;

\_lastCalculatedResult = determinant;

ResultLabel.Text = " Последний полученный резульат: " +

determinant.Value.ToString(CultureInfo.InvariantCulture);

}

dataBox1.DeterminantCalculated -= DrawResult;

}

private void Setsize()

{

if (!\_firstLoop)

{

if (MessageBox.Show("Вы действительно хотите сбросить данные в таблице?",

"Важный вопрос!", MessageBoxButtons.YesNo) != DialogResult.Yes) return;

}

if (int.TryParse(size.Text, out var value))

{

if (value > 0)

{

\_firstLoop = false;

simpleCalcButton.Enabled = true;

showDetailedStripButton.Enabled = true;

dataBox1.SetSize(value);

}

else MessageBox.Show("РАЗМЕР НЕ МОЖЕТ БЫТЬ МЕНЬШЕ ЕДЕНИЦЫ");

}

else

{

MessageBox.Show("Введённый размер должен быть целочисленным!");

}

}

private void Size\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode != Keys.Enter) return;

Setsize();

}

private void CreateNewStepsForm()

{

var x = new StepsView(dataBox1);

x.ShowDialog();

}

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

{

new Thread(CreateNewStepsForm).Start();

}

private void AnimatedCheck\_CheckedChanged(object sender, EventArgs e)

{

\_waitTime = animatedCheck.Checked ? 1000 : 0;

}

private void AnimationTimer\_Tick(object sender, EventArgs e)

{

if (ResultLabel.ForeColor == Color.Red)

{

ResultLabel.ForeColor = Color.Black;

}

else { ResultLabel.ForeColor = Color.Red; }

}

private void setSize\_Button\_Click(object sender, EventArgs e)

{

Setsize();

}

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

{

var excel = new MExcel();

if (\_lastCalculatedResult != null)

excel.DisplayInExcel(dataBox1.CopyToArray(), (double) \_lastCalculatedResult);

}

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

{

if (\_lastCalculatedResult != null)

MWord.CreateWordDoc(dataBox1.CopyToArray(), (double) \_lastCalculatedResult);

}

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

{

var ppoint = new Ppoint();

ppoint.ShowPresentation();

}

private void MainForm\_FormClosing(object sender, FormClosingEventArgs e)

{

Properties.Settings.Default.size = size.Text;

Properties.Settings.Default.aChecked = animatedCheck.Checked;

Properties.Settings.Default.Save();

}

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

{

size.Text = Properties.Settings.Default.size;

animatedCheck.Checked = Properties.Settings.Default.aChecked;

}

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

{

new About().ShowDialog();

}

private void helpToolStripMenuItem1\_Click(object sender, EventArgs e)

{

try

{

Process SysInfo = new Process();

SysInfo.StartInfo.ErrorDialog = true;

SysInfo.StartInfo.FileName = "Help.chm";

SysInfo.Start();

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

}

}

## DataBox.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading;

using System.Windows.Forms;

using DeterminantCalculator;

namespace Main\_solution

{

public partial class DataBox : UserControl

{

public delegate void DeterminantStateHandler(double? message);

public event DeterminantStateHandler DeterminantCalculated;

private static ManualResetEvent \_event;

private TriangulationMethod \_calculator;

private TextBox \_texBox;

private Thread \_myThread;

private double? \_determinant;

public int Steps { get; private set; }

public DataBox()

{

InitializeComponent();

dataGridView.EditingControlShowing +=

DataGridView\_EditingControlShowing;

\_event = new ManualResetEvent(true);

}

public void ConnectTextBox(ref TextBox textBox)

{

\_texBox = textBox;

}

private static void DataGridView\_EditingControlShowing(object sender,

DataGridViewEditingControlShowingEventArgs e)

{

e.Control.KeyPress -= Column\_KeyPress;

if (e.Control is TextBox tb) tb.KeyPress += Column\_KeyPress;

}

private static void Column\_KeyPress(object sender, KeyPressEventArgs e)

{

if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar)

&& e.KeyChar != '.'

&& e.KeyChar != '-')

e.Handled = true;

if (e.KeyChar == '.'

&& ((TextBox)sender).Text.IndexOf('.') > -1)

e.Handled = true;

if (e.KeyChar == '-'

&& ((TextBox)sender).Text.IndexOf('-') == 0)

e.Handled = true;

}

private void RestartCalculating()

{

progressBar1.Value = progressBar1.Minimum;

if (\_texBox == null)

{

\_calculator = new TriangulationMethod(ref dataGridView);

}

else

{

\_calculator = new TriangulationMethod(ref dataGridView, ref \_texBox);

}

\_calculator.MatrixChanged += UpdateDataGrid;

\_myThread = new Thread(Calculate)

{

Priority = ThreadPriority.Highest

};

\_myThread.Start();

}

public double CalcAll(int waitTime)

{

progressBar1.Value = progressBar1.Minimum;

\_calculator = new TriangulationMethod(ref dataGridView)

{

AutomaticMode = true

};

\_calculator.MatrixChanged += UpdateDataGrid;

\_determinant = \_calculator?.Calc(\_event,waitTime);

if (\_determinant != null)

{

DeterminantCalculated?.Invoke(\_determinant);

}

return (double)\_determinant;

}

public void NextStep()

{

if (\_myThread == null)

{

RestartCalculating();

}

else

{

if (!\_myThread.IsAlive)

{

\_myThread.Abort();

RestartCalculating();

}

}

MakeStep();

}

private static void MakeStep()

{

\_event?.Set();

}

public void SetSize(int size)

{

Random rnd = new Random();

dataGridView.UseWaitCursor = false;

dataGridView.Rows.Clear();

dataGridView.RowCount = size;

dataGridView.ColumnCount = size;

for (var str = 0; str < size; str++)

{

dataGridView.Rows[str].HeaderCell.Value = (str + 1).ToString();

for (var col = 0; col < size; col++)

{

dataGridView.Columns[col].HeaderText = (col + 1).ToString();

dataGridView[col, str].Value = rnd.Next(); ;

}

}

progressBar1.Minimum = 0;

progressBar1.Value = progressBar1.Minimum;

Steps = CountSteps(size);

progressBar1.Maximum = Steps;

}

private static int CountSteps(int size)

{

var counter = 0;

for (var str = 0; str < size; str++) counter += str;

return counter;

}

private void UpdateDataGrid(double[][] matrix)

{

if (matrix == null) return;

for (var row = 0; row < dataGridView.RowCount; row++)

for (var col = 0; col < dataGridView.ColumnCount; col++)

dataGridView[col, row].Value = Math.Round(matrix[row][col], 3);

dataGridView.Refresh();

++progressBar1.Value;

}

public void CopyToDataGrid(DataBox dataBox)

{

dataBox.dataGridView.RowCount = dataGridView.RowCount;

dataBox.dataGridView.ColumnCount = dataGridView.ColumnCount;

for (var row = 0; row < dataGridView.RowCount; row++)

{

dataBox.dataGridView.Rows[row].HeaderCell.Value = (row + 1).ToString();

for (var col = 0; col < dataGridView.ColumnCount; col++)

{

dataBox.dataGridView.Columns[col].HeaderText = (col + 1).ToString();

dataBox.dataGridView[col, row].Value = dataGridView[col, row].Value;

}

}

dataBox.progressBar1.Minimum = progressBar1.Minimum;

dataBox.progressBar1.Value = progressBar1.Minimum;

dataBox.Steps = Steps;

dataBox.progressBar1.Maximum = Steps;

}

private void Calculate()

{

CheckForIllegalCrossThreadCalls = false;

progressBar1.Value = progressBar1.Minimum;

dataGridView.UseWaitCursor = true;

dataGridView.ReadOnly = true;

if (\_event == null) return;

\_determinant = \_calculator?.Calc(\_event);

if (\_determinant != null)

{

DeterminantCalculated?.Invoke(\_determinant);

}

dataGridView.ReadOnly = false;

dataGridView.UseWaitCursor = false;

MessageBox.Show("Вычисление завершено.");

}

public double[][] CopyToArray()

{

var localMatrix = new LinkedList<double[]>();

for (var str = 0; str < dataGridView.Rows.Count; str++)

{

var rowValues = new LinkedList<double>();

for (var col = 0; col < dataGridView.Rows[str].Cells.Count; col++)

rowValues.AddLast(double.Parse(dataGridView.Rows[str].Cells[col].Value.ToString()));

localMatrix.AddLast(rowValues.ToArray());

}

return localMatrix.ToArray();

}

}

## About.cs

using System.Windows.Forms;

namespace Main\_solution

{

public partial class About : Form

{

public About()

{

InitializeComponent();

}

}

}

## MExcel.cs

using Excel = Microsoft.Office.Interop.Excel;

namespace Main\_solution

{

class MExcel

{

public void DisplayInExcel(double[][] data, double result)

{

var excelApp = new Excel.Application

{

Visible = true

};

excelApp.Workbooks.Add();

Excel.\_Worksheet workSheet = (Excel.Worksheet)excelApp.ActiveSheet;

var laststr = data.Length;

for (int str = 0; str < data.Length; str++)

for (var col = 0; col < data[str].Length; col++)

{

var value = data[str][col].ToString();

if (value == "65535")

{

value = "не число";

}

workSheet.Cells[str+1, col+1].Value = value;

}

var res = result.ToString();

if (res == "65535")

{

res = "не число";

}

workSheet.Cells[laststr+1, 1].Value = "Определитель = ";

workSheet.Cells[laststr+1, 2].Value = result;

for(int col =1;col<=data[0].Length;col++)

workSheet.Columns[col].AutoFit();

}

}

}

## MWord.cs

using Microsoft.Office.Interop.Word;

using System.Globalization;

namespace Main\_solution

{

internal class MWord

{

public static void CreateWordDoc(double[][] data, double result)

{

object oEndOfDoc = "\\endofdoc";

\_Application oWord = new Application

{

Visible = true

};

\_Document oDoc = oWord.Documents.Add();

var res = result.ToString();

if (res == "65535")

{

res = "не число";

}

oDoc.Content.Text = "Определитель = "+res;

Table oTable;

var wrdRng = oDoc.Bookmarks.get\_Item(ref oEndOfDoc).Range;

oTable = oDoc.Tables.Add(wrdRng, data.Length, data[0].Length);

oTable.Borders.Enable = 3;

for (var str = 0; str < data.Length; str++)

for (var col = 0; col < data[str].Length; col++)

{

var value = data[str][col].ToString(CultureInfo.InvariantCulture);

if (value == "65535")

{

value = "не число";

}

oTable.Cell(str + 1, col + 1).Range.Text = value;

}

}

}

}

## Ppoint.cs

using System.Windows.Forms;

using PowerPoint = Microsoft.Office.Interop.PowerPoint;

namespace Main\_solution

{

class Ppoint

{

PowerPoint.Application ppApp;

PowerPoint.Presentations objPresSet;

PowerPoint.SlideShowWindows objSSWs;

PowerPoint.SlideShowSettings objSSS;

public void ShowPresentation()//открытие презентации

{

ppApp = new PowerPoint.Application();

string path = Application.StartupPath + @"\presentation.ppt";

ppApp.Visible = Microsoft.Office.Core.MsoTriState.msoTrue;

objPresSet = ppApp.Presentations;

PowerPoint.\_Presentation oPres = objPresSet.Open(path ,

Microsoft.Office.Core.MsoTriState.msoFalse, Microsoft.Office.Core.MsoTriState.msoFalse,

Microsoft.Office.Core.MsoTriState.msoTrue);

}

}

}

## Preview.cs

using System;

using System.Windows.Forms;

namespace Main\_solution

{

public partial class Preview : Form

{

public Preview()

{

InitializeComponent();

timer1.Start();

}

private void timer1\_Tick(object sender, EventArgs e)

{

if (progressBar1.Value < progressBar1.Maximum) progressBar1.PerformStep();

else Close();

}

}

}

## StepsView.cs

using System;

using System.Windows.Forms;

namespace Main\_solution

{

public partial class StepsView : Form

{

public StepsView(DataBox dataBox)

{

InitializeComponent();

dataBox.CopyToDataGrid(dataBox1);

dataBox1.ConnectTextBox(ref textBox1);

}

private void Step\_Button\_Click(object sender, EventArgs e)

{

dataBox1?.NextStep();

}

}

}

## TriangulationMethod.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Windows.Forms;

namespace DeterminantCalculator

{

public class TriangulationMethod

{

public delegate void MatrixStateHandler(double[][] matrix);

public event MatrixStateHandler MatrixChanged;

private bool \_automaticMode;

private const int WidthOfTextbox = 52;

private TextBox \_textBox;

private double[][] \_matrix;

public bool AutomaticMode { get => \_automaticMode; set => \_automaticMode = value; }

public TriangulationMethod(ref DataGridViewEx dataGrid)

{

\_automaticMode = false;

var localMatrix = new LinkedList<double[]>();

for (var str = 0; str < dataGrid.Rows.Count; str++)

{

var rowValues = new LinkedList<double>();

for (var col = 0; col < dataGrid.Rows[str].Cells.Count; col++)

rowValues.AddLast(double.Parse(dataGrid.Rows[str].Cells[col].Value.ToString()));

localMatrix.AddLast(rowValues.ToArray());

}

\_matrix = localMatrix.ToArray();

}

public TriangulationMethod(ref DataGridViewEx dataGrid, ref TextBox textBox) :

this(ref dataGrid)

{

\_textBox = textBox;

}

private void CleanTextBox()

{

if (\_textBox == null) return;

\_textBox.Text = "";

}

private void Print(string text)

{

if (\_textBox == null) return;

if (\_textBox.Text.Length > 0)

\_textBox.Text += text;

else

\_textBox.Text = text;

}

private void PrintLine(string text)

{

if (\_textBox == null) return;

if (\_textBox.Text.Length > 0)

\_textBox.Text += text + "\r\n";

else

\_textBox.Text = text + "\r\n";

}

private static string MatrixToStr(IEnumerable<double[]> matrix)

{

var sb = new StringBuilder();

foreach (var str in matrix)

{

Array.ForEach(str, i => sb.Append(i + " "));

sb.Append("\r\n");

}

return sb.ToString();

}

private static string ArrayToString(double[] array)

{

var sb = new StringBuilder();

Array.ForEach(array, i => sb.Append(i + " "));

return sb.ToString();

}

public double Calc(ManualResetEvent resetEvent, int waitTime = 1)

{

CleanTextBox();

var size = \_matrix.Length;

int stepsCounter = 1;

for (var str = 0; str < \_matrix.Length; str++)

{

for (var j = str + 1; j < size; j++)

{

if (\_automaticMode)

{

Thread.Sleep(waitTime);

}

else

{

resetEvent.WaitOne();

}

PrintLine($"Шаг: {stepsCounter++}");

PrintLine(new string('˅', WidthOfTextbox));

var mul = -(\_matrix[j][str] / \_matrix[str][str]);

PrintLine($"-({\_matrix[j][str]})" +

$"\r\n/\r\n" +

$"({\_matrix[str][str]})" +

$"\r\n=\r\n" +

$"{mul}");

PrintLine(new string('˅', WidthOfTextbox));

var mul2 = MultiplyMas(\_matrix[str], mul);

PrintLine($"({ArrayToString(\_matrix[str])})" +

$"\r\n\*\r\n" +

$"({mul})" +

$"\r\n=\r\n" +

$"{ArrayToString(mul2)}");

PrintLine(new string('˅', WidthOfTextbox));

var newStr = FoldMasvs(\_matrix[j], mul2);

PrintLine($"({ArrayToString(\_matrix[j])})" +

$"\r\n+\r\n" +

$"({ArrayToString(mul2)})" +

$"\r\n=\r\n" +

$"{ArrayToString(newStr)}");

PrintLine(new string('˅', WidthOfTextbox));

\_matrix[j] = newStr;

Print(MatrixToStr(\_matrix));

PrintLine(new string('=', WidthOfTextbox));

if(!\_automaticMode) resetEvent.Reset();

MatrixChanged?.Invoke(\_matrix);

}

}

double determinant = 1;

for (int i = 0, j = 0; i < size; i++, j++) determinant \*= \_matrix[i][j];

PrintLine($"Определитель = {determinant}");

return determinant;

}

private double[] FoldMasvs(double[] mas1, IReadOnlyList<double> mas2)

{

var newMas = new double[mas1.Length];

Array.Copy(mas1, newMas, mas1.Length);

for (var i = 0; i < mas2.Count; i++)

newMas[i] += mas2[i];

return newMas;

}

private double[] MultiplyMas(double[] mas, double value)

{

var newMas = new double[mas.Length];

Array.Copy(mas, newMas, mas.Length);

for (var i = 0; i < mas.Length; i++) newMas[i] \*= value;

return newMas;

}

}

}

## DataGridViewEx.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace DeterminantCalculator

{

public class DataGridViewEx : DataGridView

{

public DataGridViewEx()

{

SetStyle(ControlStyles.OptimizedDoubleBuffer | ControlStyles.AllPaintingInWmPaint, true);

}

}

}