Московский Авиационный Институт (Национальный исследовательский университет) Факультет прикладной математики и информационных технологий

**Курсовая работа по дисциплине «Объектно-ориентированное программирование».**

Выполнил студент: Жилов Андрей А.

Группы : М8О-205Б-21

Руководитель: Кузнецова С.В

Оценка:

Дата:

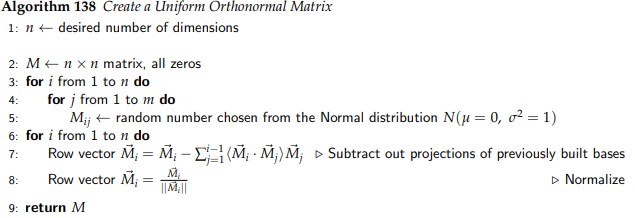
Москва 2022

**Алгоритм №138”Создание ортогональной матрицы”**

**Постановка задачи**

Реализация алгоритма проблемы на С# и обеспечение визуализации выполнения алгоритма. Данный алгоритм используется для нахождения корня уравнения на отрезке. Делает она это выбирая начальную точку и троя касательную к ней которая пересекает ось ОХ в следующей точке дальше алгоритм запускается по новой. Алгоритм работает пока не даст заданное заранее приближение к корню функции.

**Описание алгоритма**



Сначала мы создаем матрицу размеров nxn и заполняем её случайными числами. Затем с помощью метода Грамма Шмидта находим ортогональный вектор и нормируем его, добавляем в конечную матрицу и возвращаем итоговую матрицу

**Реализация алгоритма С#**

**Controller**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace kyrs138

{

internal class Controller

{

private Model model;

public Controller(Model model\_in)

{

this.model = model\_in;

}

public double[,] getRandomMatrix()

{

return model.enterRandomNumbers();

}

public void SetSizeOfMatrix(int n)

{

model.putSizeOfMatrix(n);

}

//public double[,] getOrthonormalMatrix()

//{

// return model.OrthonormalMatrix();

//}

public void getOrthonormalMatrix()

{

model.OrthonormalMatrix();

return;

}

public double Check()

{

return model.checkMatrix();

}

public void SetMatrix(double[,] M)

{

model.getMatrix(M);

}

}

}

**Model**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

using System.Threading;

using System.Collections;

namespace kyrs138

{

public class Model

{

public delegate void MatrixCallback(double[,] Matrix);

public event MatrixCallback OnUpdate;

private ArrayList Listeners;

double[,] M;

Random rand;

public int n;

double[,] B;

public Model()

{

rand = new Random();

n = 0;

Listeners = new ArrayList();

}

public void putSizeOfMatrix(int size)

{

n = size;

M = new double[n, n];

}

public void register(Observer listener)

{

Listeners.Add(listener);

//OnUpdate += listener.OnUpdate;

}

public void UpdateObservers()

{

foreach(Observer observer in Listeners)

{

observer.OnUpdate(B);

}

}

public double checkMatrix()//проверка матрицы на ортогональность

{

double[] A = new double[n];

double[] B = new double[n];

double max = 0;

double c=0;

for (int i = 0; i < n - 1; i++)

{

for (int j = i + 1; j < n; j++)

{

A = getRow(M, i);

B = getRow(M, j);

c = scalarPr(A, B);

if (c > max)

max = c;

}

}

return c;

}

public double[,] enterRandomNumbers()//заполнение матрицы рандомными числами

{

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

{

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

{

if (rand.Next(1, 2) == 1)

{

M[i, j] = 1.00 / rand.Next(-20, -1);

}

else

{

M[i, j] = 1.00 / rand.Next(1, 20);

}

}

}

return M;

}

public void getMatrix(double[,] a)

{

M = a;

}

public double scalarPr(double[] A, double[] B)//Скалярное произведение векторов

{

double c = 0;

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

{

c += A[i] \* B[i];

}

return c;

}

public double[] vectorSub(double[] A, double[] B)//разность векторов

{

double[] C = new double[n];

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

{

C[i] = A[i] - B[i];

}

return C;

}

public double[] getRow(double[,] A, int index)//получение строки

{

double[] C = new double[n];

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

{

C[j] = A[index, j];

}

return C;

}

public double[] getColl(double[,] A, int index)//получение столбца

{

double[] C = new double[n];

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

{

C[j] = A[j, index];

}

return C;

}

public double[] vectorSum(double[] A, double[] B)//сумма векторов

{

double[] C = new double[n];

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

{

C[i] = A[i] + B[i];

}

return C;

}

public double[] Multiplication(double A, double[] B)//произведение вектрора на число

{

double[] C = new double[n];

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

{

C[i] = B[i] \* A;

}

return C;

}

public double[] vectorMultiplication(double[] A, double[] B)//произведение векторов

{

double[] C = new double[n];

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

{

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

{

C[i] += A[i] \* B[j];

}

}

return C;

}

public double[] sumOfscalar(int i, double[,] B)//сумма проекций векторов

{

double[] C = new double[n];

for (int j = 0; j < i; j++)//summ=scalarproisv(mixmj)xmj

{

C = vectorSum(C, Multiplication(scalarPr(getRow(M, i), getRow(B, j)) / scalarPr(getRow(B, j), getRow(B, j)), getRow(B, j)));// an-сумма проекция an на bj j=1 -> n-1

}

return C;

}

public double[] normalization(double[] A)

{

double c = 0;

c = Math.Sqrt(scalarPr(A, A));

if (c != 0)

{

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

{

A[i] = A[i] / c;

}

}

return A;

}

public void OrthonormalMatrix()//вычисление с помошбю грамма-шмидта

{

double[] C = new double[n];

B = new double[n, n];

//double[,] X = new double[n, n];

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

{

C = vectorSub(getRow(M, i), sumOfscalar(i, B));//mi-sum

C=normalization(C);

//dataGridView1.Rows[i] = vectorSubtraction(dataGridView1.Rows[i], sumOfscalar(scalarPr(scalarPr()) , i));

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

{

//dataGridView2.Rows[j].Cells[i].Value = C[j];

// X[j, i] = C[j];

B[i, j] = C[j];

//OnUpdate?.Invoke(B);

UpdateObservers();

}

Thread.Sleep(1000);

}

M = B;

return;

}

}

}

**Реализация пользовательского режима**

**Form1**

using System;

using System.CodeDom;

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;

using static System.Net.Mime.MediaTypeNames;

namespace kyrs138

{

public partial class Form1 : Form,Observer

{

public int n;

private Model model;

private Controller controller;

public Form1(Model model\_l)

{

n = 0;

InitializeComponent();

this.model = model\_l;

model.register(this);

controller = new Controller(model);

}

public void MatrixInTable1(double[,]M)//v

{

//A = new double [n,n];

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

{

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

{

dataGridView1.Rows[j].Cells[i].Value = M[i, j];

}

}

}

public void MatrixInTable2(double[,] M)//v

{

//A = new double [n,n];

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

{

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

{

dataGridView2.Rows[j].Cells[i].Value = M[i, j];

}

}

}

//delegate void MatrixCallback(double[,] Matrix);

//public void OnUpdate(double[,] B)

//{

// if (this.dataGridView2.InvokeRequired)

// {

// MatrixCallback d = new MatrixCallback(OnUpdate);

// this.Invoke(d, new object[] { B });

// }

// else

// {

// label5.Text = B.ToString();

// MatrixInTable2(B);

// }

//}

public void OnUpdate(double[,] b)

{

MatrixInTable2(b);

this.dataGridView2.Refresh();

}

public double[,] Table1ToMatrix()

{

double[,] A = new double[n, n];

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

{

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

{

A[i, j] =Convert.ToDouble( dataGridView1.Rows[j].Cells[i].Value);

}

}

return A;

}

private void button1\_Click\_1(object sender, EventArgs e)

{

n = Convert.ToInt32(this.numericUpDown1.Value);

if(n<=0)

{

MessageBox.Show("Введите размер матрицы больший нуля");

return;

}

dataGridView1.RowCount = n;

dataGridView1.ColumnCount = n;

dataGridView2.RowCount = n;

dataGridView2.ColumnCount = n;

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

{

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

{

dataGridView1.Rows[i].Cells[j].Value = 0;

}

}

MatrixInTable1(controller.getRandomMatrix());

}

private void button2\_Click\_1(object sender, EventArgs e)

{

controller.getOrthonormalMatrix();

textBox1.Text = controller.Check().ToString("F4");

}

private void numericUpDown1\_ValueChanged(object sender, EventArgs e)

{

n = Convert.ToInt32(this.numericUpDown1.Value);

dataGridView1.RowCount = n;

dataGridView1.ColumnCount = n;

dataGridView2.RowCount = n;

dataGridView2.ColumnCount = n;

controller.SetSizeOfMatrix(n);

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

{

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

{

dataGridView1.Rows[i].Cells[j].Value = 0;

dataGridView2.Rows[i].Cells[j].Value = 0;

}

}

}

private void button3\_Click(object sender, EventArgs e)

{

try

{

controller.SetMatrix(Table1ToMatrix());

MessageBox.Show("Данные введены");

}

catch

{

MessageBox.Show("Некоректные данные");

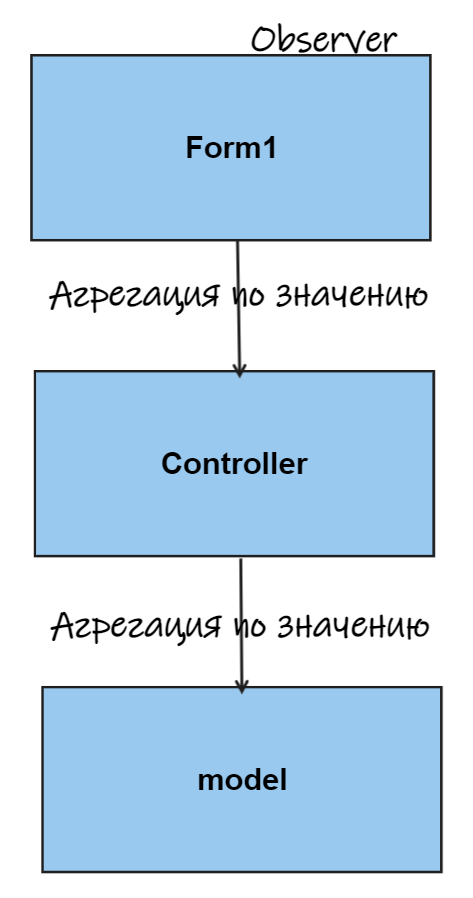
}

}

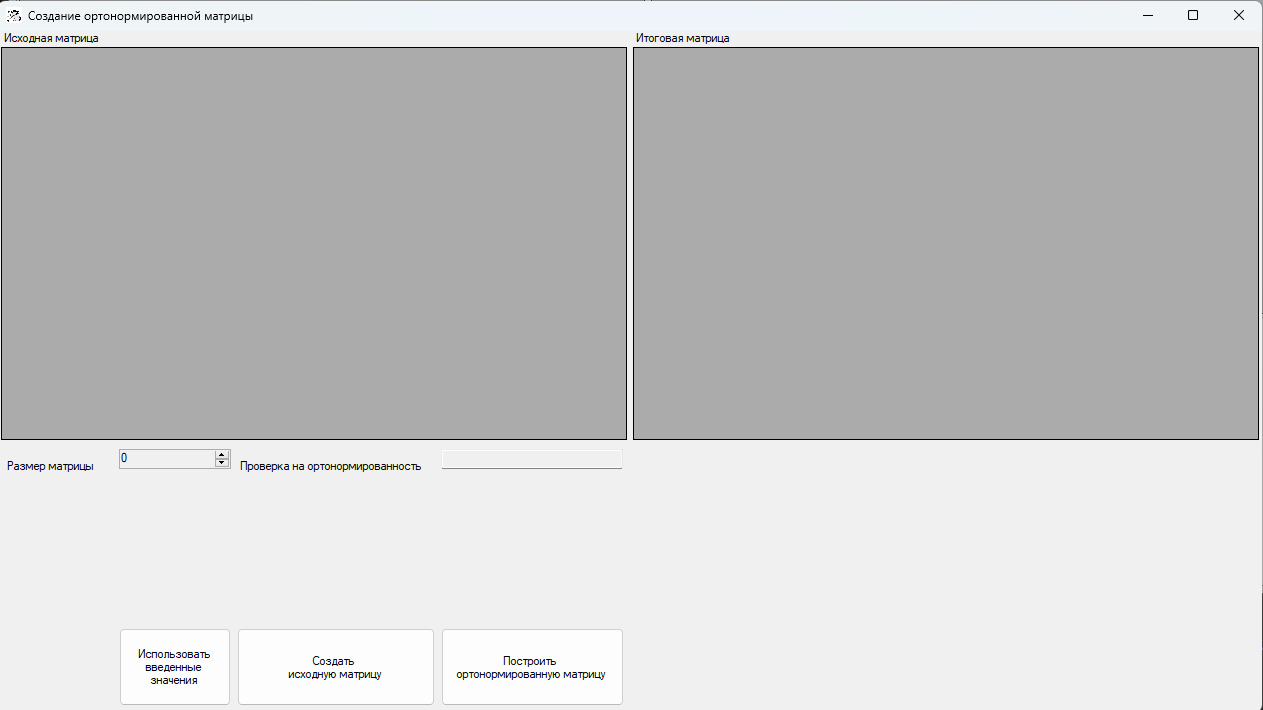
}

}

**Диаграмма класса:**

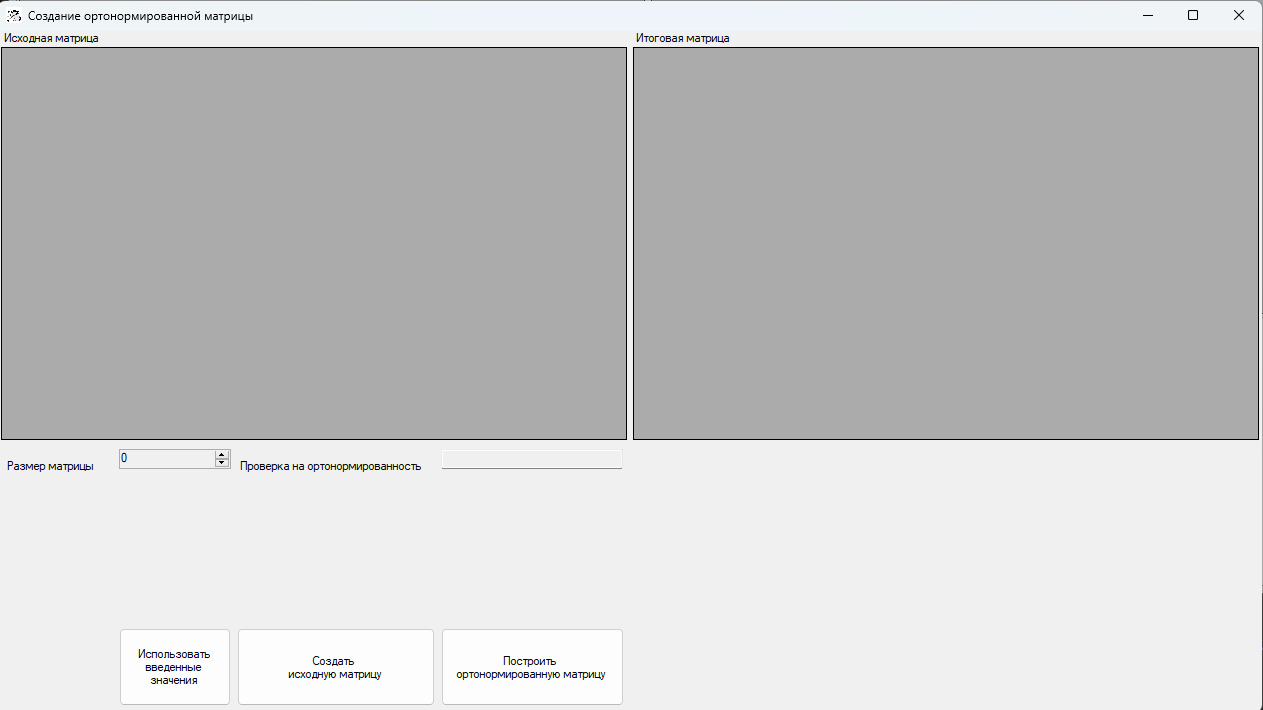
****

**Демонстрация функциональных возможностей приложения**

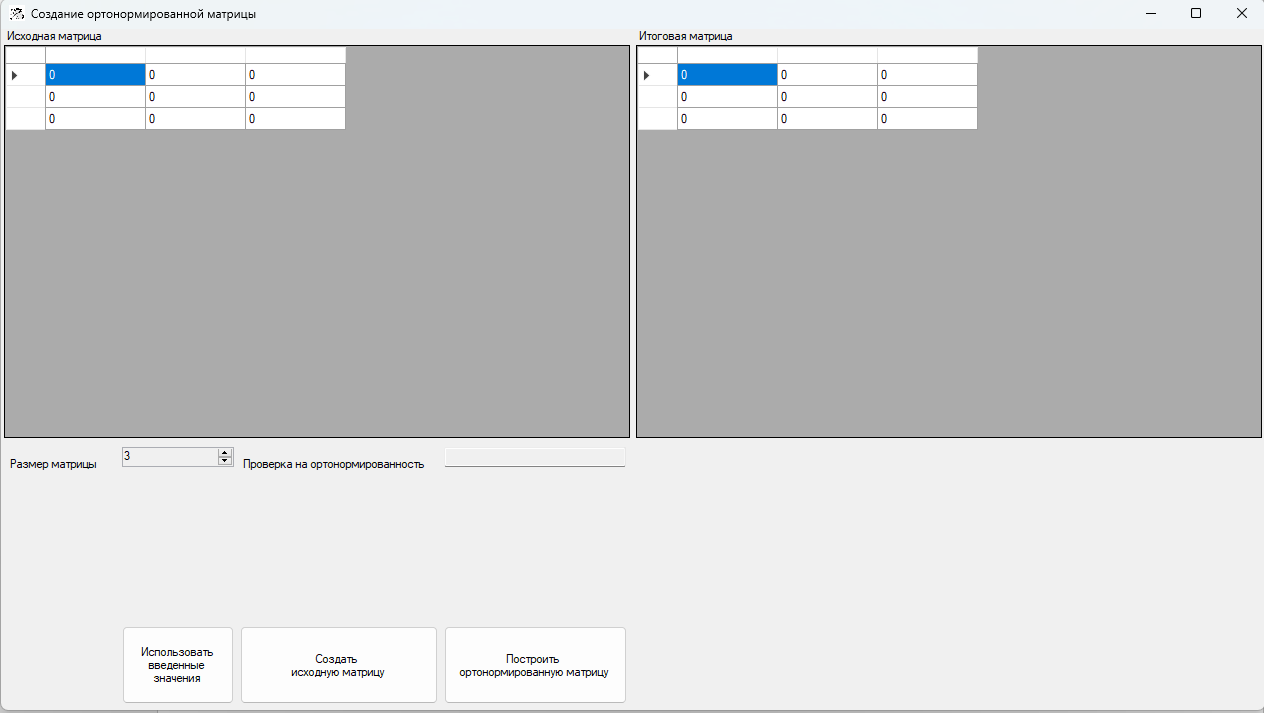


**Тесты. Протокол работы.**

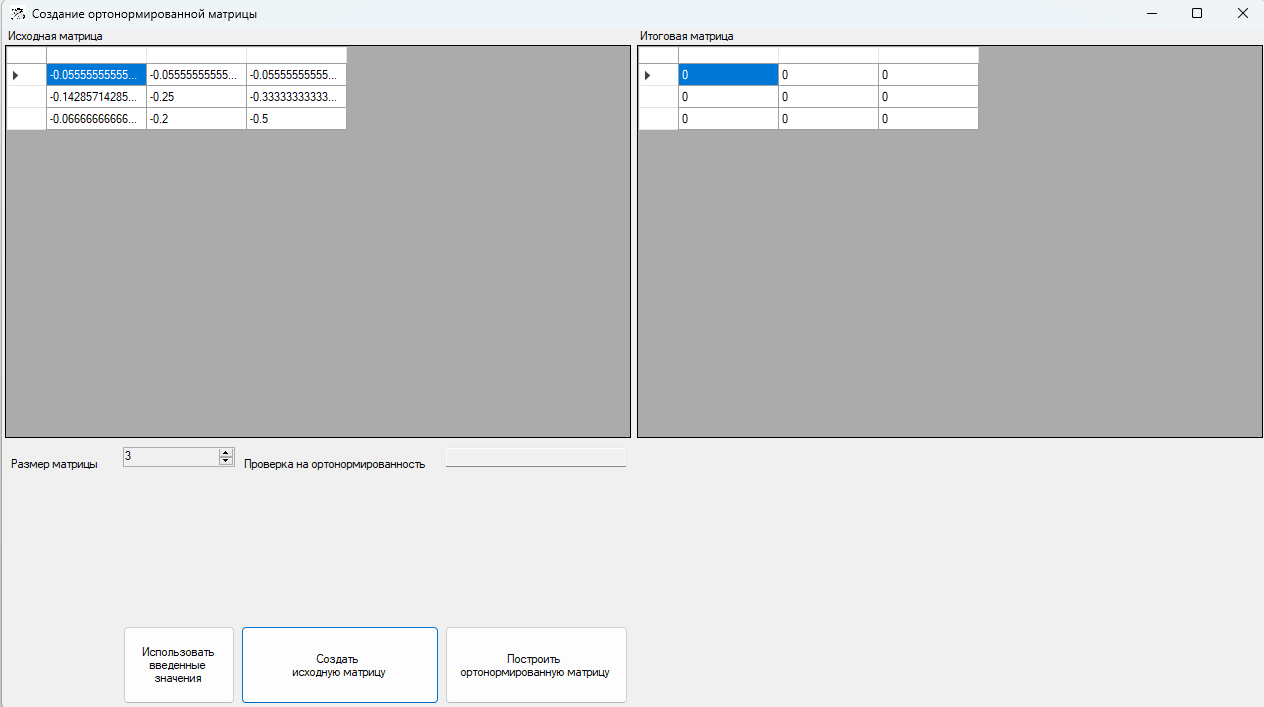
Начальный вид формы



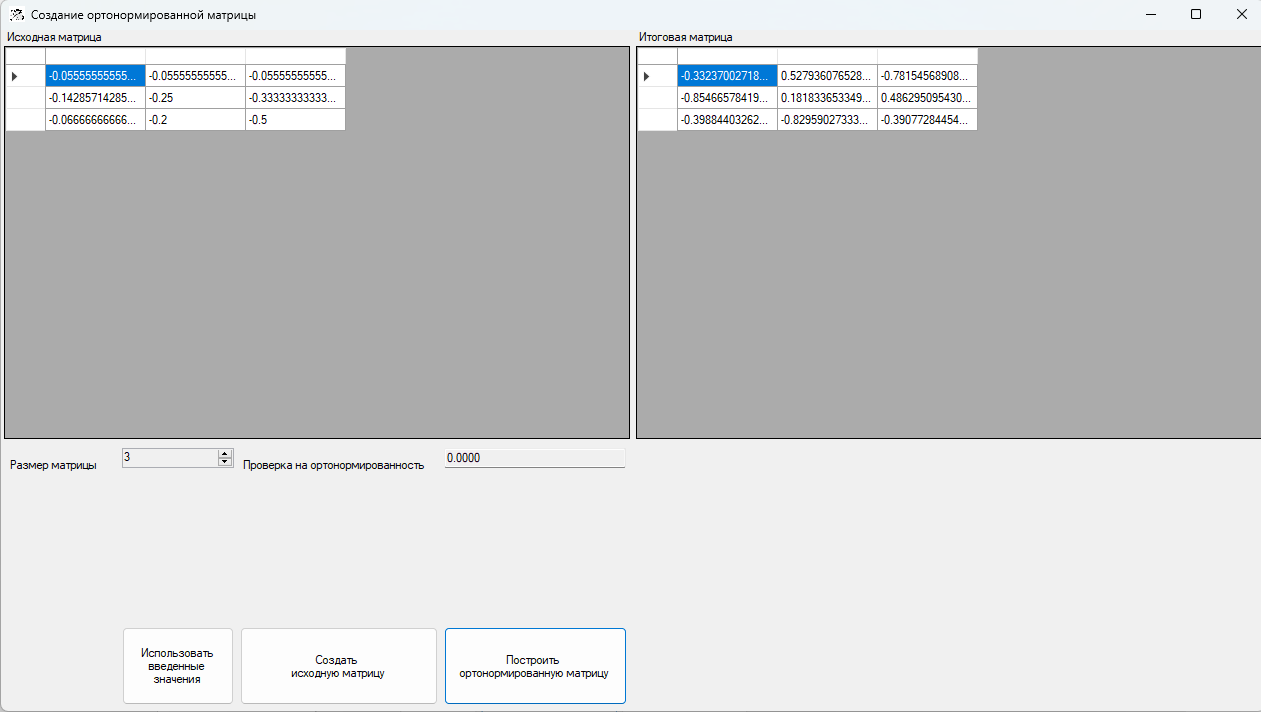
С помощью numbericUpDown можно изменить размер матрицы, сразу после изменения значения исходная матрица заполняется нулями.



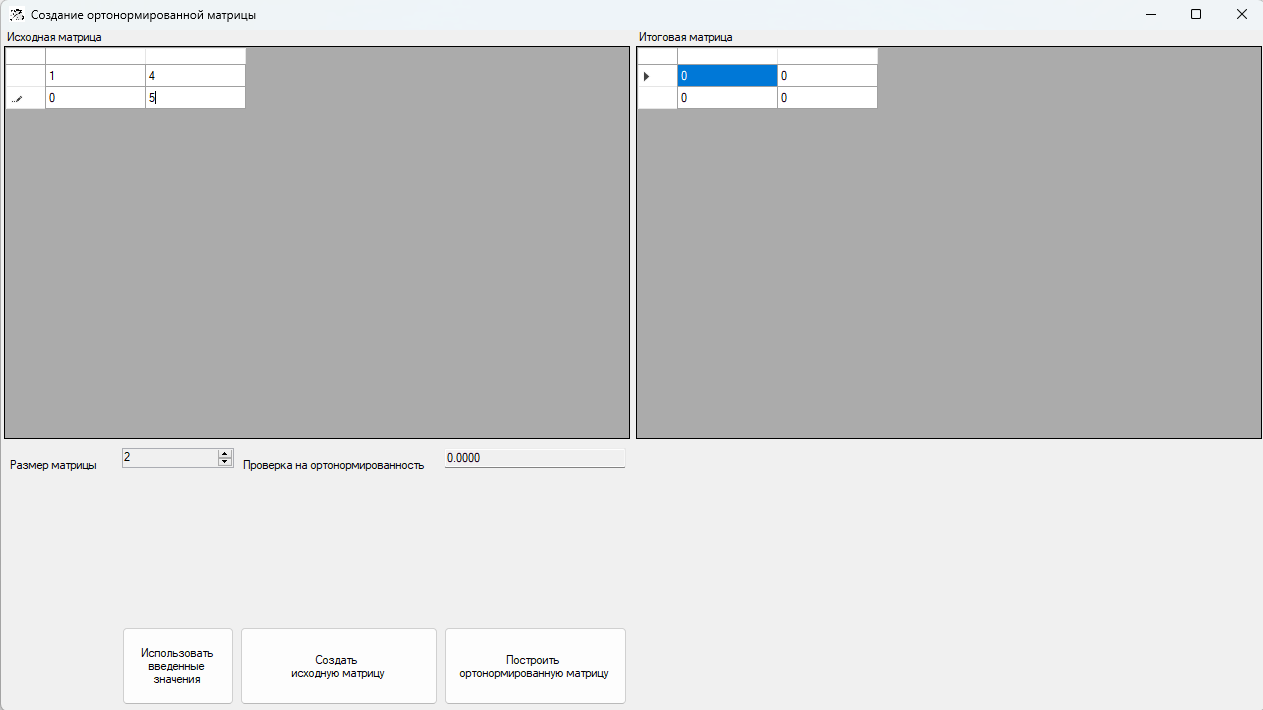
Затем с помощью button(Создать исходную матрицу) мы заполняем исходную матрицу случайными значениями.



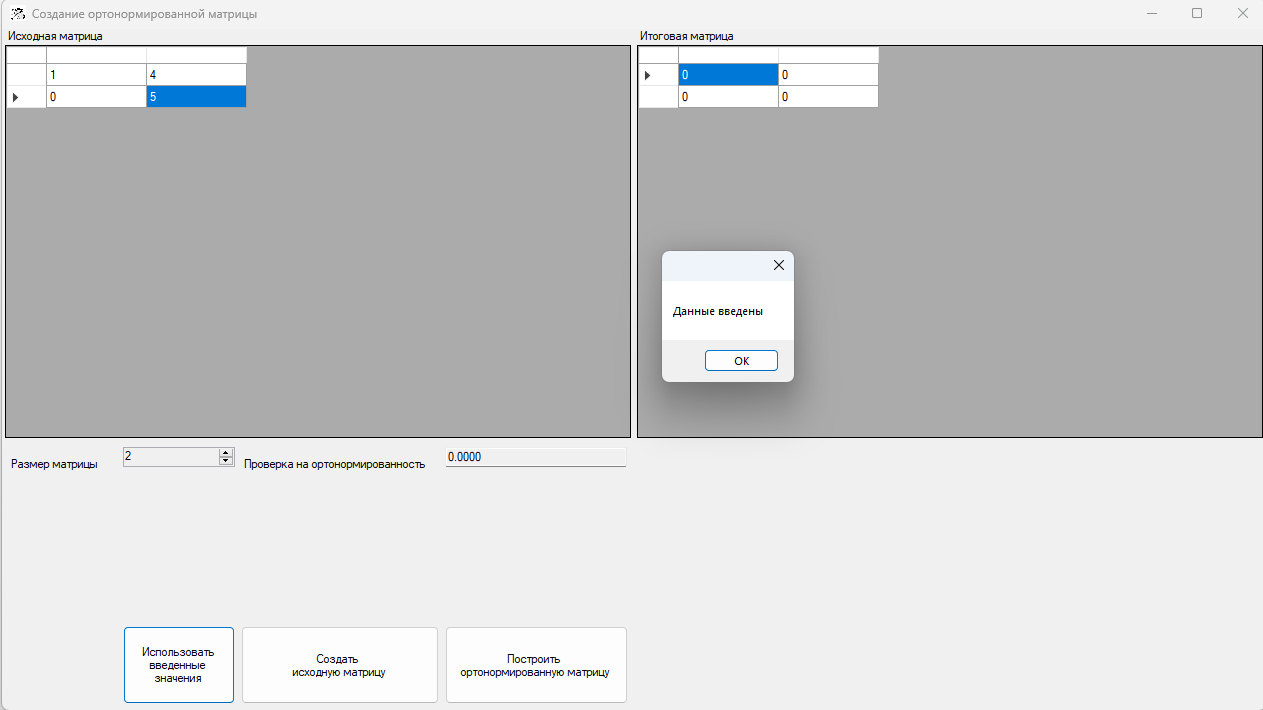
С помощью button(Построить ортонормированную матрицу выводится конечная матрица) выводится ортонормированная матрица.



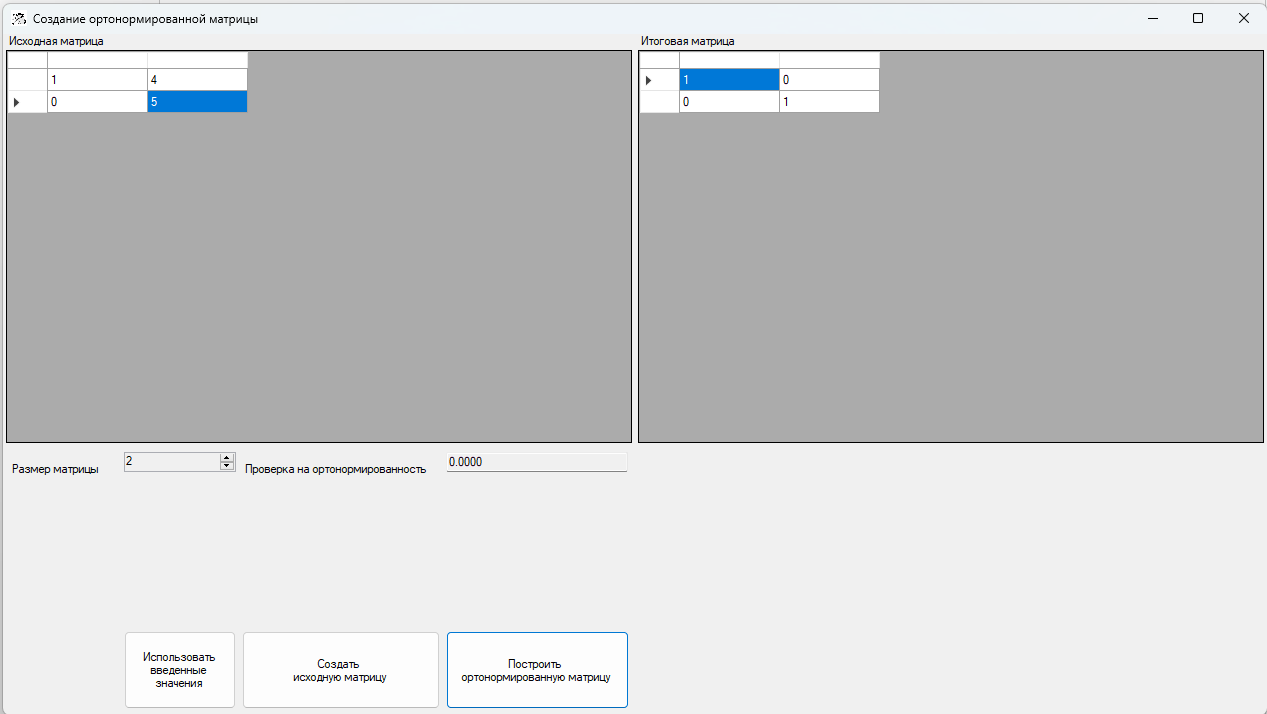
Также мы можес ввести собственные значения в исходную матрицу.



Нажав button(Использовать введенные значения) введённые данные отправляются в модель.



И с помощью button(Построить ортонормированную матрицу выводится конечная матрица)



**Необходимое программное обеспечение:** Для данного курсового проекта использовалась операционная система Windows 11 и среда разработки Microsoft Visual Studio 2022.

**Вывод:** В ходе выполнения данного курсового проекта была разработана программа создающая ортонормированная матрица с помощью метода Грамма-Шмидта. Для осуществления визуализации и пользовательского интерфейса была использована встроенная в среду библиотека Windows.Forms, а также паттерн MVC.