Функциональные возможности программного комплекса

1.1 Описание программного комплекса

Программный комплекс лабораторных работ разработанный в рамках моей работы представляет из себя Windows Form приложение, включающее в себя семь лабораторных работ по предмету «Численные методы».

Включены следующие численные методы, распределенные по лабораторным работам:

1. Решение систем линейных алгебраических уравнений точными методами: метод Гаусса и метод квадратных корней (Холецкого).
2. Решение систем линейных алгебраических уравнений итерационными методами: метод Якоби, метод Зейделя и метод верхней релаксации (обобщенный метод Зейделя).
3. Решение плохо обусловленных систем линейных алгебраических уравнений: метод регуляризации и метод вращения (Гивенса).
4. Решение нелинейных уравнений и систем нелинейных уравнений: метод простых итераций и метод Ньютона.
5. Решение проблемы собственных значений и собственных векторов: метод Леверрье, метод Фадеева и метод Крылова.
6. Решение проблемы собственных значений и собственных векторов: QR-алгоритм.
7. Приближение функций: интерполяционный полином Лагранжа, приближение полиномами Ньютона, интерполирование функций с помощью кубического сплайна и аппроксимация функций методом наименьших квадратов многочленом второй степени.

Каждая отдельная лабораторная работа содержится в своем классе (lab1 – лабораторная №1, lab2 – лабораторная №2 и т.д.). Все лабораторные работы выполнены в графическом интерфейсе. Исключением являются некоторые из лабораторных, которым требуется изменять исходный код для изменения входных данных.

1.2 Описание графического интерфейса

Лабораторная работа №1-2. Обе лабораторные работы исследуют численные методы решения систем линейных алгебраических уравнений. Графический интерфейс содержит в себе таблицу ввода исходной матрицы 3x3 и таблицу ввода исходного вектора (рис. 3.1).

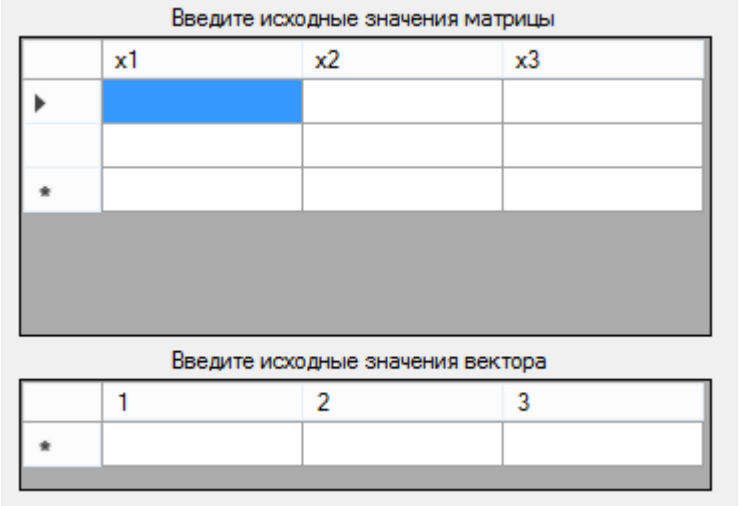


Рис. 3.1 – Элементы для ввода исходных данных

Лабораторная работа №3. Данная работа исследует численные методы решения систем линейных алгебраических уравнений. Графический интерфейс содержит в себе таблицу ввода исходной матрицы 2x2 и таблицу ввода исходного вектора (рис. 3.2).

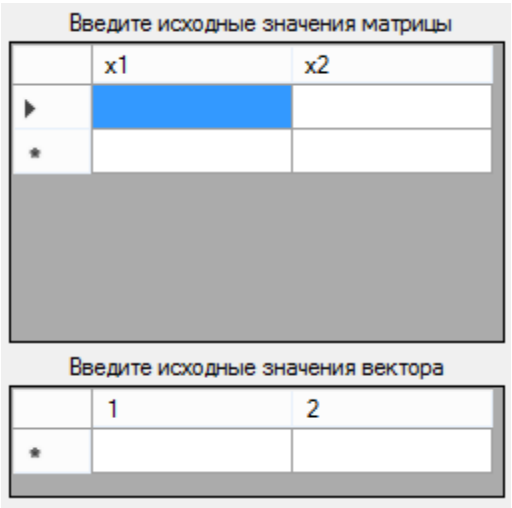


Рис. 3.2 – Элементы для ввода исходных данных

Лабораторная работа №4. Данная работа исследует численные методы решения нелинейных уравнений и систем нелинейных уравнений.

Графический интерфейс содержит в себе изображение, которое указывает на исходные данные, кнопки вызова методов и таблица с решением (рис.3.3).

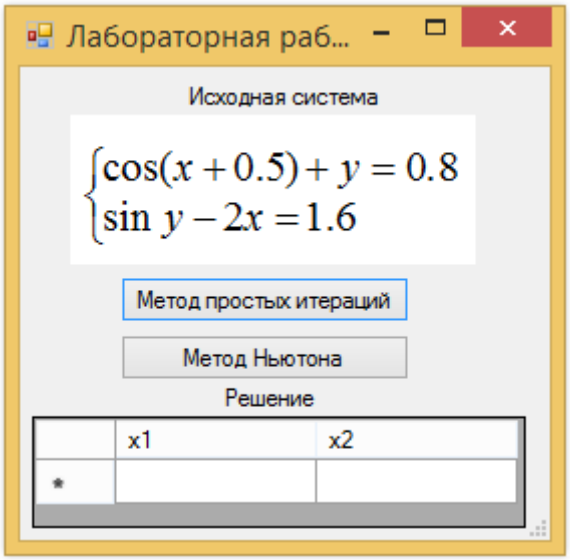


Рис. 3.3 – Графический интерфейс лабораторной работы

Для изменения исходных данных требуется изменить исходный код следующих функций:

- double func(double[] x, int i);
- double MatrJacobi(double[] x, int i, int j);
- double jacobian(double[] x, int i, int j);
- double func2(double[] x, int i).

Лабораторная работа №5. Данная работа исследует численные методы решения проблемы собственных значений и собственных векторов. Графический интерфейс содержит в себе таблицу ввода исходной матрицы 4x4 (рис. 3.4).

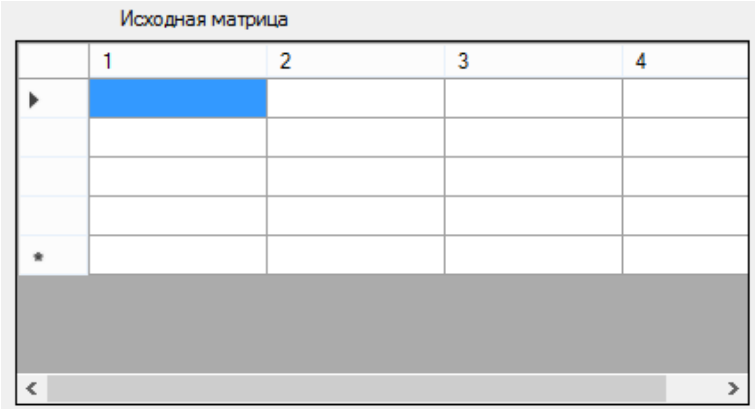


Рис. 3.4 – Графический интерфейс пятой лабораторной работы

Лабораторная работа №6. Данная работа исследует численные методы решения проблемы собственных. Графический интерфейс содержит в себе таблицу ввода исходной матрицы 3x3 (рис. 3.5).

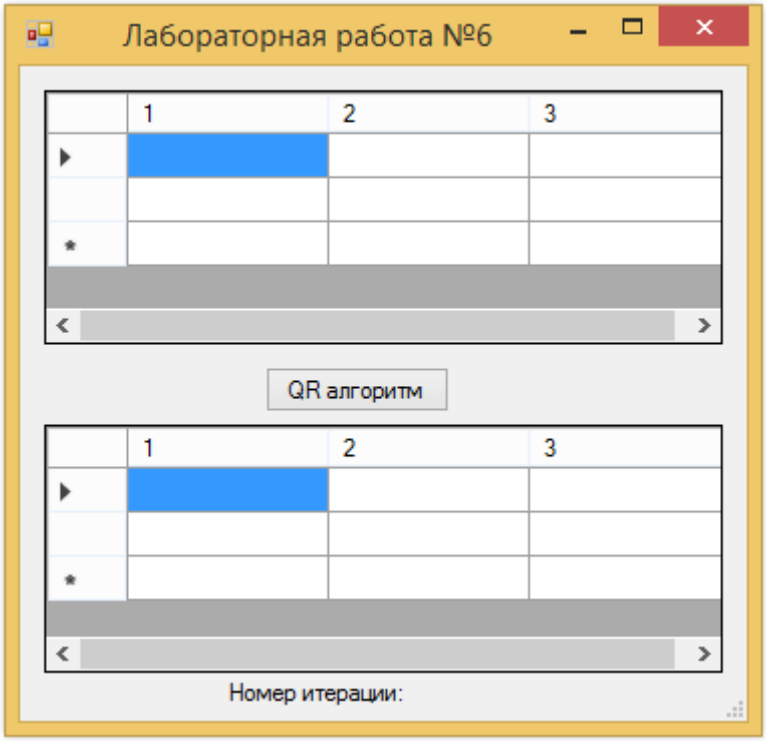


Рис. 3.5 – Графический интерфейс шестой лабораторной работы

Лабораторная работа №7. Данная работа исследует численные методы приближения функция. Графический интерфейс содержит в себе таблицу ввода исходных значений x и y полученные от решения функции, а так же поле для ввода значения, относительно которого нужен результат (рис. 3.6).

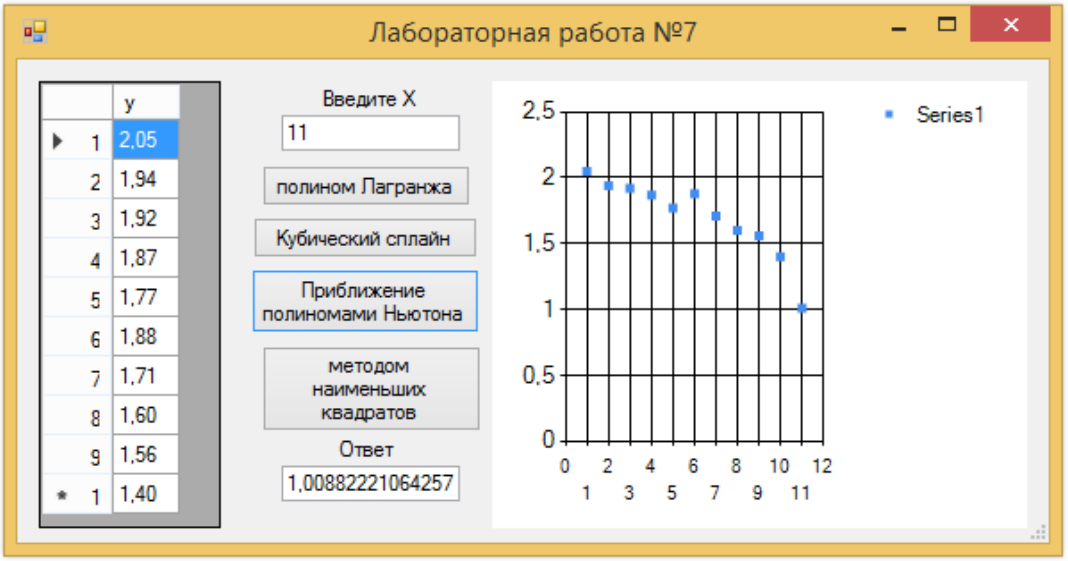


Рис. 3.6 – Графический интерфейс седьмой лабораторной работы

1.3 Примеры решения численных методов

Лабораторная работа №1-2. Рассмотрим процесс работы лабораторной на примере решения системы линейных алгебраических уравнений, указанной на рис. 3.7.

$$\begin{cases} x_1+x_2+x_3=6 \\ x_1-x_3=-2 \\ x_1+2x_2+x_3=8 \end{cases}$$

Рис. 3.7 – Система линейных алгебраических уравнений

Введем коэффициенты левой части системы в таблицу «исходной матрицы», а в правой части в таблицу «исходного вектора». Входные данные будут иметь вид, как на рис. 3.8.

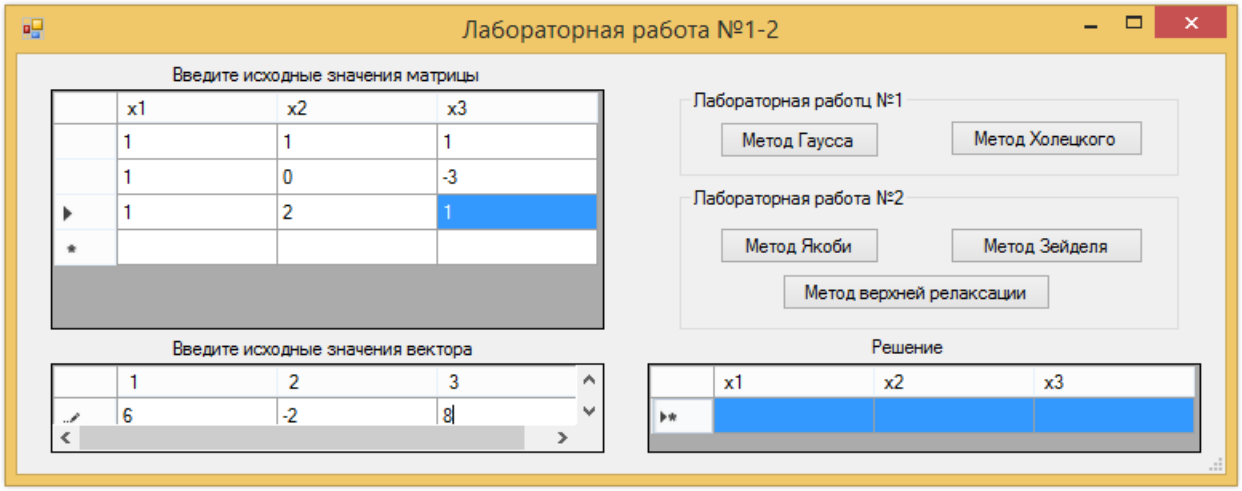


Рис. 3.8 – Входные данные лабораторной работы

Для получения решения одним из методов выберите соответствующую кнопку. Например, решим методом Гаусса рис. 3.9.

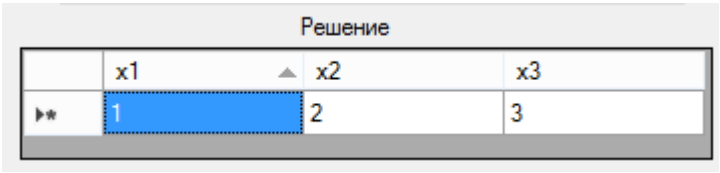


Рис. 3.9 – Полученное решение методом Гаусса

Лабораторная работа №3. Рассмотрим процесс работы лабораторной на примере решения системы линейных алгебраических уравнений, указанной на рис. 3.10.

$$\begin{cases} 1.03x_1 + 0.991x_2 = 2.51 \\ 0.991x_1 + 0.943x_2 = 2.41 \end{cases}$$

Рис. 3.10 – Система линейных алгебраических уравнений

Введем коэффициенты левой части системы в таблицу «исходной матрицы», а в правой части в таблицу «исходного вектора». Входные данные будут иметь вид, как на рис. 3.11.

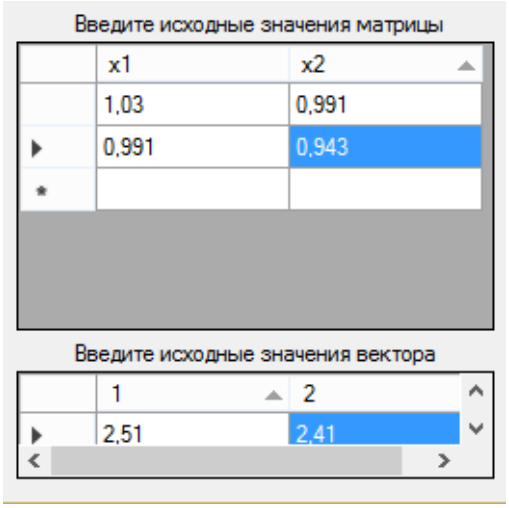


Рис. 3.11 – Входные данные лабораторной работы

Для получения решения одним из методов выберите соответствующую кнопку. Например, решим методом Регуляризации рис. 3.12.

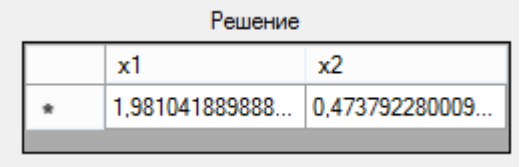


Рис. 3.12 – Решение системы методом Регуляризации

Лабораторная работа №4. Рассмотрим решение на примере не линейной системы уравнений, представленной на рис. 3.13.

$$\begin{cases} \cos(x + 0.5) + y = 0.8 \\ \sin y - 2x = 1.6 \end{cases}$$

Рис. 3.13 – Система нелинейных уравнений

Для получения решения одним из методов выберите соответствующую кнопку. Например, решим методом простых итераций на рис. 3.14.

Решение

	x1	x2
*	-0,86658080752...	-0,13355832610...

Рис. 3.14 – Решение системы методом Регуляризации

Лабораторная работа №5. Рассмотрим процесс работы лабораторной на примере решения системы, указанной на рис. 3.15.

$$\begin{pmatrix} 1 & -1 & -1 & 2 \\ 2 & 3 & 0 & -4 \\ 1 & 1 & -2 & -2 \\ 1 & 1 & 0 & -1 \end{pmatrix}$$

Рис. 3.15 – Исходная система

Введем значения системы в таблицу «исходной матрицы». Входные данные будут иметь вид, как на рис. 3.16.

	1	2	3	4
▶	1	-1	-1	2
	2	3	0	-4
	1	1	-2	-2
	1	1	0	-1
*				

Рис. 3.16 – Входные данные лабораторной работы

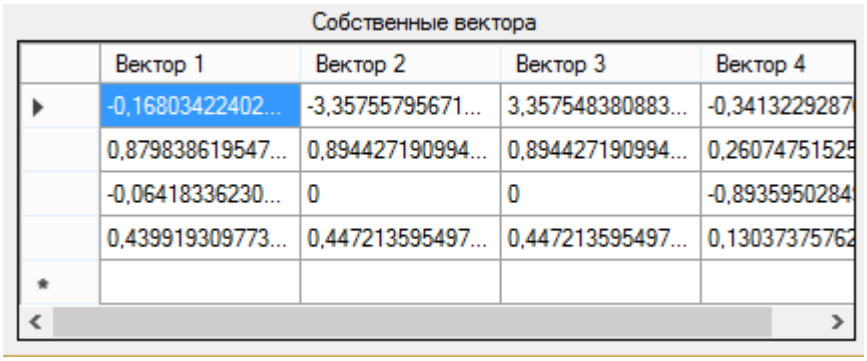
Для получения собственных значений одним из методов выберите соответствующую кнопку. Например, решим методом Фадеева на рис. 3.17.

Собственные значения

	Собственные значения
▶	0,618033988749...
	0,999992492272...
	1,000007507706...
	-1,61803398872...
*	

Рис. 3.17 – Собственные значения методом Фадеева

Для получения собственных векторов одним из методов выберите соответствующую кнопку. Например, решим методом Фадеева на рис. 3.18


Собственные вектора

	Вектор 1	Вектор 2	Вектор 3	Вектор 4
▶	-0,16803422402...	-3,35755795671...	3,357548380883...	-0,3413229287
	0,879838619547...	0,894427190994...	0,894427190994...	0,26074751525
	-0,06418336230...	0	0	-0,8935950284
	0,439919309773...	0,447213595497...	0,447213595497...	0,13037375762
*				

Рис. 3.18 – Собственные вектора методом Фадеева

Лабораторная работа №6. Рассмотрим процесс работы лабораторной на примере решения системы, указанной на рис. 3.19.

$$\begin{pmatrix} 2 & 2 & -2 \\ 2 & 5 & -4 \\ -2 & -4 & 5 \end{pmatrix}$$

Рис. 3.19 – Исходная система

Введем значения системы в таблицу «исходной матрицы». Входные данные будут иметь вид, как на рис. 3.20.

	1	2	3
	2	2	-2
	2	5	-4
▶	-2	-4	5

Рис. 3.21 – Входные данные лабораторной работы

Для получения собственных значений одним из методов выберите соответствующую кнопку. Найдем решение QR-алгоритмом на рис. 3.22.

1	2	3
9,999999999999...	1,045225604807...	-3,48408567145...
1,045225484423...	0,999999999999...	5,551115123125...
-3,48408494807...	1,110223024625...	1

Рис. 3.22 – Собственные значения, полученные QR-алгоритмом

Лабораторная работа №7. Рассмотрим процесс работы лабораторной на примере исходный данных, указанных на рис. 3.23.

x_i	y_i	x_i	y_i
1	2,05	6	1,88
2	1,94	7	1,71
3	1,92	8	1,60
4	1,87	9	1,56
5	1,77	10	1,40

Рис. 3.23 – Исходные данные

Введем значения в таблицу. Входные данные будут иметь вид, как на рис. 3.24.

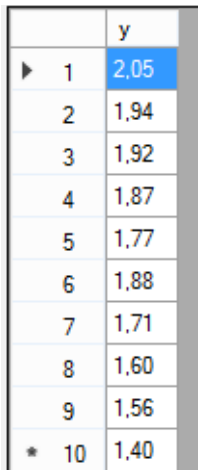


Рис. 3.24 – Входные данные лабораторной работы

Для получения Y от X одним из методов выберите соответствующую кнопку. Например, найдем Y от X методом приближения методами Ньютона (рис. 3.25).

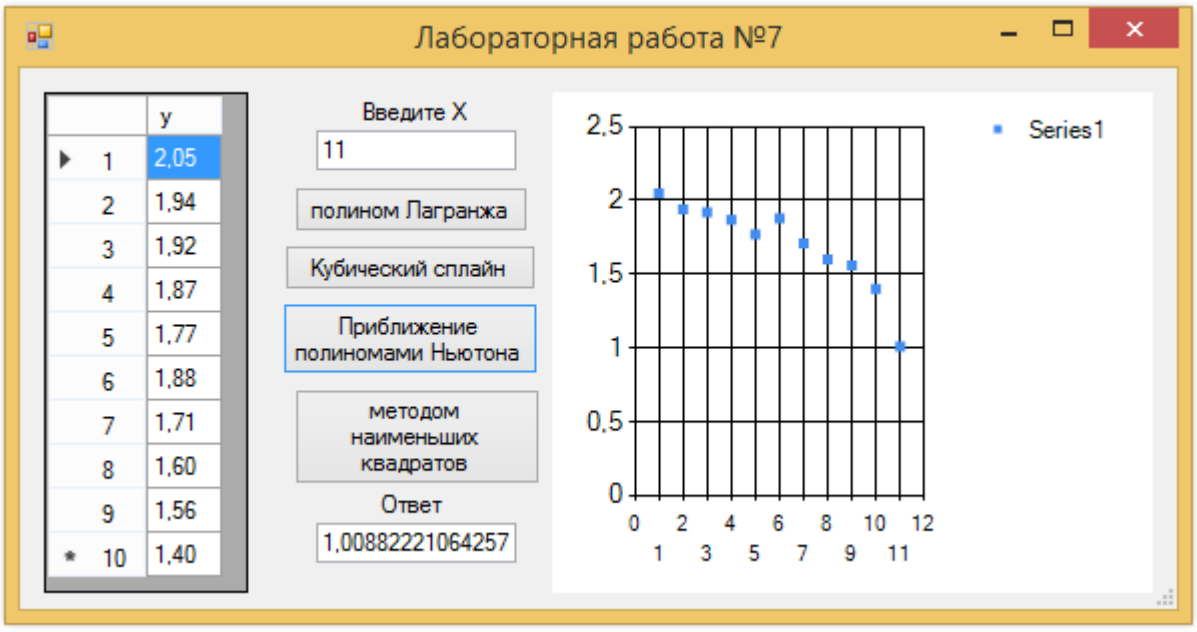


Рис. 3.25 – Y от X методом приближения Ньютона

Заключение

Разработка программного обеспечения – актуальность этого направления сейчас очень велика. Количество языков программирования и операционных систем, под которые пишут программное обеспечение великое множество, поэтому практикуясь разрабатывать программное обеспечение в рамках бакалаврской работы я обобщаю знания, полученные мною по специальности «Информатика и вычислительная техника».

В ходе выполнения бакалаврской работы, на тему «Программирование численных методов. Часть 1», было проделано следующее:

1) проведен теоретический обзор по технологии разработки программного обеспечения. Это сделано для расширения кругозора и подготовки мыслительного процесса по проектированию своей системы. В результате стали понятны цели и задачи, которые требуется поставить перед собой;

2) разработано ПО, которое подходит для применения в лабораторном комплексе по предмету «Численные методы» на кафедре ПОУТС;

3) в качестве среды разработки была выбрана MS Visual Studio, а язык программирования C#;

4) была проведена верификация данных полученных старым исходным кодом и новым, результаты совпали;

5) разработан графический интерфейс, который позволил отказаться от «консоли» и сконцентрировать внимание пользователя на результатах;

6) запрограммированы численные методы с первой по седьмую лабораторную работу;

7) описаны основные возможности программного обеспечения, такие как примеры решения численных методов и описание графического интерфейса. Это позволит студентам, которые хотят начать работать с данным лабораторным комплексом быстрее разобраться в его основах.

Список использованных источников

1. Амосов, А.А. Вычислительные методы для инженеров [Текст] / А.А. Амосов, Ю.А. Дубинский, Н.В. Копченова. – М.: Высшая школа, 1994 – 544 с.
2. Бахвалов, Н.С. Численные методы [Текст] / Н.С. Бахвалов. – М.: Наука, 1973 –631 с.
3. Вержбицкий, В.М. Численные методы. Математический анализ и обыкновенные дифференциальные уравнения [Текст] / В.М. Вержбицкий. – М.: Высшая школа, 2001 –400 с.
4. Волков, Е.А. Численные методы [Текст] / Е.А. Волков. – СПб.: Лань, 2004 –256 с.
5. Воробьева, Г.Н. Практика по численным методам [Текст] / Г.Н. Воробьева, А.Н., Данилова. – М.: Высшая школа, 1979 –184 с.
6. Демидович, Б.П. Численные методы анализа [Текст] / Б.П. Демидович, И.А. Марон, Э.З. Шувалова. – М.: Наука, 1967 –368 с.
7. Ильин, В.П. Численный анализ, Часть 1 [Текст] / В.П. Ильин. – Новосибирск : ИВМ и МГСО РАН, 2004 –335с.
8. Калиткин, Н.Н. Численные методы [Текст] / Н.Н. Калиткин. – М.: Наука, 1978 –512 с.
9. Копченова, Н.В. Вычислительная математика в примерах и задачах [Текст] / Н.В. Копченова, И.А. Марон. – М.: Наука, 1972 –246 с.
10. Костомаров, Д.П. Вводные лекции по численным методам [Текст] / Д.П. Костомаров, А.П. Фаворский. – М.: Логос, 2004 –184с.

Приложение А

Исходный код лабораторной работы №1

```
double[] Gauss(int Row, int Colum, double[] B, double[,] A)
        {
            RightPart = new double[Row];
            Answer = new double[Row];
            Matrix = new double[Row][];
            for (int i = 0; i < Row; i++)
                Matrix[i] = new double[Colum];
            RowCount = Row;
            ColumCount = Colum;
            //обнулим массив
            for (int i = 0; i < Row; i++)
            {
                Answer[i] = 0;
                RightPart[i] = 0;
                for (int j = 0; j < Colum; j++)
                    Matrix[i][j] = 0;
            }
            ReturnVal = new double[Row];
            //заполняем правую часть
            for (int i = 0; i < Row; i++)
            {
                RightPart[i] = B[i];
            }
            for (int i = 0; i < Row; i++)
            {
                for (int j = 0; j < Row; j++)
                {
                    Matrix[j][i] = A[j, i];
                }
            }
            for (int i = 0; i < RowCount - 1; i++)
            {
                SortRows(i);
                for (int j = i + 1; j < RowCount; j++)
                {
                    if (Matrix[i][i] != 0) //если главный элемент не 0, то производим
вычисления
                    {
                        double MultElement = Matrix[j][i] / Matrix[i][i];
                        for (int k = i; k < ColumCount; k++)
                            Matrix[j][k] -= Matrix[i][k] * MultElement;
                        RightPart[j] -= RightPart[i] * MultElement;
                    }
                    //для нулевого главного элемента просто пропускаем данный шаг
                }
            }

            //ищем решение
            for (int i = (int)(RowCount - 1); i >= 0; i--)
            {
                Answer[i] = RightPart[i];

                for (int j = (int)(RowCount - 1); j > i; j--)
                    Answer[i] -= Matrix[i][j] * Answer[j];

                Answer[i] /= Matrix[i][i];

            }
            return Answer;
           }
```

```
double[] Holetskiy(int N, double[,] A1, double[] B1)
        {
            double summ;
            double[,] c = new double[N, N + 1], L = new double[N, N + 1];
            double[] y = new double[N];
            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N + 1; j++)
                {
                    c[i, j] = 0;
                    L[i, j] = 0;
                    y[i] = 0;
                }
            }
            //Умножение матрицы на транспонированную
            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    summ = 0.0;
                    for (int t = 0; t < N; t++)
                    {
                        summ = A1[t, j] * A1[t, i] + summ;
                    }
                    c[i, j] = summ;
                    //Console.WriteLine("c[" + i + "," + j + "] = " + c[i,j]);
                }
            }
            //{умножение правой части на транспонированную м-цу}
            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    y[i] = A1[j, i] * B1[j] + y[i];
                }
            }
            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
                    A1[i, j] = c[i, j];
                    B1[i] = y[i];
                }
            }
            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j <= i; j++)
                {
                    summ = 0;
                    for (int t = 0; t <= j - 1; t++)
                    {
                        summ = summ + L[i, t] * L[j, t];
                    }
                    if (i != j)
                    {
                        L[i, j] = (A1[i, j] - summ) / L[j, j];
                    }
                    else
                    {
                        L[i, i] = Math.Sqrt(A1[i, i] - summ);
                    }
                }
            }
            for (int i = 0; i < N; i++)
```

```
{
    L[i, N] = B1[i];
}
B1[0] = L[0, N] / L[0, 0];
for (int i = 1; i < N; i++)
{
    for (int j = 0; j <= i - 1; j++)
    {
        L[i, N] = L[i, N] - L[i, j] * B1[j];
    }
    B1[i] = L[i, N] / L[i, i];
}
for (int i = 0; i < N; i++)
{
    for (int j = i + 1; j < N; j++)
    {
        L[i, j] = L[j, i];
        L[j, i] = 0;
    }
    L[i, N] = B1[i];
}
B1[N - 1] = L[N - 1, N] / L[N - 1, N - 1];
for (int i = N - 1 - 1; i >= 0; i--)
{
    for (int j = i + 1; j < N; j++)
    {
        L[i, N] = L[i, N] - L[i, j] * B1[j];
    }
    B1[i] = L[i, N] / L[i, i];
}
return B1;
   }
```

Приложение Б

Исходный код лабораторной работы №2

```
double[] Jacobi(double[,] A, double[] B)
        {
            int N = 3;
            double[] X = new double[N];
            X[0] = 1; X[1] = 1; X[2] = 1;
            double[] TempX = new double[N];
            double norm; // норма, определяемая как наибольшая разность компонент столбца
иксов соседних итераций.
            do
            {
                for (int i = 0; i < N; i++)
                {
                    TempX[i] = -B[i];
                    for (int g = 0; g < N; g++)
                    {
                        if (i != g)
                            TempX[i] += A[i, g] * X[g];
                    }
                    TempX[i] /= -A[i, i];
                }
                norm = Math.Abs(X[0] - TempX[0]);
                for (int h = 0; h < N; h++)
                {
                    if (Math.Abs(X[h] - TempX[h]) > norm)
                        norm = Math.Abs(X[h] - TempX[h]);
                    X[h] = TempX[h];
                }
            } while (norm > eps);
            return X;
               }
double[] Zejdel(double[,] A, double[] B)
        {
            int n = 3;
            double[] z = new double[n];

            double[] x = new double[n];
            int i, j;
            for (i = 0; i < n; ++i)
            {
                B[i] /= A[i, i];
                z[i] = A[i, i];
                for (j = 0; j < n; ++j)
                    A[i, j] /= z[i];
            }
            x[1] = x[2] = 0;
            i ^= i;
            do
            {
                x[0] = B[0] - A[0, 1] * x[1] - A[0, 2] * x[2];
                x[1] = B[1] - A[1, 0] * x[0] - A[1, 2] * x[2];
                x[2] = B[2] - A[2, 0] * x[0] - A[2, 1] * x[1];
            } while (!Equal(x[0], x[1], x[2]));
            return x;
        }
        double[] ZejdelEx(double[,] A, double[] B)
        {
            int n = 3;
            double[] x0 = new double[n];
            double[] x = new double[n];
            int step = 0;
```

```
double e = 0;
//Параметр релаксации
double w = 0.2;
eps = 0.0001;
for (int i = 0; i < n; i++)
{
    x0[i] = B[i] / A[i, i];
}
do
{
    for (int i = 0; i < n; i++)
    {
        x[i] = w * B[i] / A[i, i] + (1 - w) * x0[i];

        for (int j = 0; j <= i - 1; j++)
        {
            x[i] = x[i] - w * A[i, j] * x[j] / A[i, i];
        }
        for (int j = i + 1; j < n; j++)
        {
            x[i] = x[i] - w * A[i, j] * x0[j] / A[i, i];
        }

    }
    e = 0;
    for (int i = 0; i < n; i++)
    {
        if (Math.Abs(x[i] - x0[i]) > e) { e = Math.Abs(x[i] - x0[i]); }
        x0[i] = x[i];
    }
    step++;
}
while (e >= eps);
return x;
   }
```

Приложение В

Исходный код лабораторной работы №3

```
double[] Givens(double[,] A, double[] B)
        {
            int Nn = 2;
            double[] x = new double[Nn];
            //for (int i = 0; i < Nn; i++)
            //{
            //    A[i,0] = B[i];
            //}
            double A_0_1 = A[0, 1];
            double M = 0.0;
            double L, R;
            for (int i = 0; i < Nn - 1; i++)
            {
                for (int k = i + 1; k < Nn; k++)
                {
                    M = Math.Sqrt(A[i, i] * A[i, i] + A[k, i] * A[k, i]);
                    L = A[k, i] / M; //Вычислили A12
                    M = A[i, i] / M; //Вычислили B12
                    for (int j = 0; j < Nn; j++)
                    {
                        R = A[i, j];
                        A[i, j] = M * A[i, j] + L * A[k, j]; // {получили a1j}
                        A[k, j] = M * A[k, j] - L * R;      //     {получили a2j}
                    }
                    R = B[i];
                    B[i] = M * B[i] + L * B[k];
                    B[k] = M * B[k] - L * R;
                }
            }
            Console.WriteLine("Матрица приняла вид после вращения Гивенса:");
            for (int i = 0; i < Nn; i++)
            {
                for (int j = 0; j < Nn; j++)
                {
                    Console.Write("a[" + i + "," + j + "]=" + A[i, j] + "     ");
                }
                Console.WriteLine("b[" + i + "]=" + B[i]);
            }
            x[1] = B[1] / A[1, 1];
            B[0] = 2.51;
            A[0, 0] = 1.03;
            x[0] = (B[0] - A_0_1 * x[1]) / A[0, 0];
            return x;
        }
        double[] regul(int n, double[,] a, double[] b)
        {
            double[] result;
            double[,] a1 = new double[n, n], a2 = new double[n, n];
            double[] b1 = new double[n], x0 = new double[n];
            double eps = 0.005;
            double s;
            int k;
            for (int i = 0; i < n; i++)
            {
                for (k = 0; k < n; k++)
                {
                    s = 0;
                    for (int j = 0; j < n; j++)
                    {
                        s = s + a[j, i] * a[j, k];
```

```
                }
                a1[i, k] = s;
            }
        }
        for (int i = 0; i < n; i++)
        {
            s = 0;
            for (int j = 0; j < n; j++)
            {
                s = s + a[j, i] * b[j];
            }
            b1[i] = s;
        }
        double alfa = 0;
        k = 0;
        double[] b2 = new double[n];
        b2 = vozm(n, eps, b2);
        double max;
        do
        {
            alfa = alfa + 0.00000001;
            a2 = a1;
            for (int i = 0; i < n; i++)
            {
                a2[i, i] = a1[i, i] + alfa;
                b2[i] = b1[i] + alfa * x0[i];
            }
            a1 = a2; b1 = b2;
            Lab1 lb1 = new Lab1();
            b2 = lb1.StartGauss(a2, b2, n);
            a2 = a1; result = b2; x0 = result; b2 = b1;
            b2 = lb1.StartGauss(a2, b2, n);
            max = Math.Abs(b2[1] - result[1]);
            for (int i = 1; i < n; i++)
            {
                if ((Math.Abs(b2[1] - result[1])) > max)
                {
                    max = Math.Abs(b2[i] - result[i]);
                }
            }
        } while (max > eps);
        return result;
    }
    double[] vozm(int n, double eps, double[] b)
    {
        double[] b2 = new double[n];
        for (int i = 0; i < n; i++)
        {
            b2[i] = b2[i] + eps;
        }
        return b2;
                                                }
```

Приложение Г

Исходный код лабораторной работы №4

```
double Norma(double[,] a, int n)
        {
            double res = 0;
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    res += a[i, j] * a[i, j];
                }
            }
            res = Math.Sqrt(res);
            return res;
        }
        double func(double[] x, int i)
        {
            double res = 0;
            switch (i)
            {
                case 0: res = (Math.Sin(x[1]) - 1.6) / 2;
                    break;
                case 1:
                    res = 0.8 - Math.Cos((x[0]) + 0.5);
                    break;
            }
            return res;
        }
        double MatrJacobi(double[] x, int i, int j)
        {
            double res = 0;
            switch (i)
            {
                case 0:
                    switch (j)
                    {
                        case 0:
                            res = Math.Sin(x[0] + 0.5);
                            break;
                        case 1:
                            res = 0;
                            break;
                    }
                    break;
                case 1:
                    switch (j)
                    {
                        case 0:
                            res = 0;
                            break;
                        case 1:
                            res = 0.5 * Math.Cos(x[1]);
                            break;
                    }
                    break;
            }
            return res;
        }
        void vivod_vectr(double[] vect)
        {
            int n = 2;
            for (int i = 0; i < n; i++)
```

```
    {
        dataGridView3.Rows[0].Cells[i].Value = vect[i];
    }
}
public void simpte_iter()
{
    int n = 2;
    int iter = 0;
    double[] x0 = new double[n];
    double[] x = new double[n];
    double[,] a = new double[n, n];
    x0[0] = 0; x0[1] = 0;
    double max, eps = 1e-4;
    do
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                a[i, j] = MatrJacobi(x0, i, j);
            }
        }
        vivod_vectr(x0);
        //Console.WriteLine("Norma = " + Norma(a, n));
        //Console.WriteLine("Nomer iterazii = " + iter);
        //Console.WriteLine("=============");
        for (int i = 0; i < n; i++)
        {
            x[i] = func(x0, i);
        }
        max = Math.Abs(x[0] - x0[0]);
        for (int i = 1; i < n; i++)
        {
            if (Math.Abs(x[i] - x0[i]) > max)
            {
                max = Math.Abs(x[i] - x0[i]);
            }
        }
        x0 = x;
        iter++;

        //Console.ReadLine();
    }
    while ((max > eps) || (iter < 20));
    //Console.ReadLine();
}

double jacobian(double[] x, int i, int j)
{
    double res = 0;
    switch (i)
    {
        case 0:
            switch (j)
            {
                case 0:
                    res = -Math.Sin(x[0] + 0.5);
                    break;
                case 1:
                    res = 1;
                    break;
            }
            break;
        case 1:
```

```
            switch (j)
            {
                case 0:
                    res = -2;
                    break;
                case 1:
                    res = Math.Cos(x[1]);
                    break;
            }
            break;
    }
    return res;
}
double func2(double[] x, int i)
{
    double res = 0;
    switch (i)
    {
        case 0: res = Math.Cos(x[0] + 0.5) + x[1] - 0.8;
            break;
        case 1:
            res = Math.Sin(x[1]) - 2 * x[0] - 1.6;
            break;
    }
    return res;
}
public void Nyuton()
{
    int n = 2;
    int iter = 0;
    double[] dx = new double[n];
    double[] x = new double[n];
    double[] f = new double[n];
    double[,] a = new double[n, n];
    x[0] = 0; x[1] = 0;
    double max, eps = 1e-4;
    Lab1 lb1;
    do
    {
        vivod_vectr(x);
        //Console.WriteLine("Nomer iterazii = " + iter);
        //Console.WriteLine("=============");
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                a[i, j] = jacobian(x, i, j);
            }
        }
        for (int i = 0; i < n; i++)
        {
            f[i] = -1 * func2(x, i);
        }
        lb1 = new Lab1();
        dx = lb1.StartGauss(a, f, n);
        max = Math.Abs(dx[0]);
        for (int i = 1; i < n; i++)
        {
            if (Math.Abs(dx[i]) > max)
            {
                max = Math.Abs(dx[i]);
            }
        }
        for (int i = 0; i < n; i++)
```

```
    {
        x[i] = x[i] + dx[i];
    }
    iter++;

    //Console.ReadLine();
}
while (max > eps);
//Console.ReadLine();
                                                }
```

Приложение Д

Исходный код лабораторной работы №5

```
//проверяет входит ли в массив зна-чение Znach,
        //используется при вычислении определителя
        bool Vkl(int[] Per, int Znach, int Kol)
        {
            bool result = false;
            for (int i = 0; i <= Kol - 1; i++)
            {
                if (Per[i] == Znach)
                {
                    result = true;
                }
            }
            return result;
        }
        //для определителя указывает знак с каким входит
        //в сумму очередное слагаемое
        bool Perestanovka(int[] Per, int n)
        {
            int kol = 0;
            for (int i = 0; i <= n - 2; i++)
            {
                for (int j = i + 1; j <= n - 1; j++)
                {
                    if (Per[i] > Per[j])
                    {
                        kol++;
                    }
                }
            }

            if (kol % 2 == 0)
            {
                return false;
            }
            return true;
        }
        //формирует очеред-ное слагаемое в определителе
        double SumMatrToPer(double[,] Matr, int[] Per, int n)
        {
            double result = 1;
            for (int i = 0; i <= n - 1; i++)
            {
                result *= Matr[i, Per[i]];
            }
            if (Perestanovka(Per, n))
            {
                result *= -1;
            }
            return result;
        }
        //рекурсивно формирует перестановки и ищет определитель
        double DetRec(double[,] Matr, int n, int[] Per, int n0)
        {
            double result = 0;
            for (int i = 0; i <= n - 1; i++)
            {
                if (Vkl(Per, i, n0))
                {
                    continue;
                }
```

```
        else
        {
            Per[n0] = i;
            if (n0 == n - 1)
            {
                result = SumMatrToPer(Matr, Per, n);
            }
            else
            {
                result += DetRec(Matr, n, Per, n0 + 1);
            }
        }
    }
    return result;
}
// подготавливает массив и запускает ре-курсию
//для нахождения определителя
double Det(double[,] Matr, int n)
{
    double result = 0;
    int[] Per = new int[n];
    Per[0] = 1;
    result = DetRec(Matr, n, Per, 0);
    return result;
}
//возводит в  степень B число A
double SQRN(double a, int b)
{
    double result = 0;
    if (a > 0)
    {
        result = Math.Exp(b * Math.Log(a));
    }
    else
    {
        if (a != 0)
        {
            if (b % 2 != 0)
            {
                result = -Math.Exp(b * Math.Log(Math.Abs(a), Math.E));
            }
            else
            {
                result = Math.Exp(b * Math.Log(Math.Abs(a), Math.E));
            }
        }
        else
        {
            result = 0;
        }
    }
    return result;
}
//перемножение матриц
double[,] MatrUmn(double[,] a, double[,] b, int n, int m, int k)
{
    double[,] c = new double[n, k];
    double s;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < k; j++)
        {
            s = 0;
            for (int l = 0; l < m; l++)
```

```
            {
                s += a[i, l] * b[l, j];
            }
            c[i, j] = s;
        }
    }
    return c;
}
// считает производную многочлена, переданного в массиве
double[] Proizv(double[] xar)
{
    double[] result;
    double[] proizv = new double[xar.Length - 1];
    for (int i = 0; i < xar.Length - 2; i++)
    {
        proizv[i] = xar[i] * (xar.Length - i - 1);
    }
    proizv[xar.Length - 1 - 1] = xar[xar.Length - 1 - 1];
    result = proizv;
    return result;
}
// делит многочлен на одночлен (корень), тем самым уменьшая его степень
double[] Delenie(double[] f, double koren)
{
    double[] result;
    double[] otv = new double[f.Length];
    otv[0] = f[0];
    for (int i = 1; i < f.Length; i++)
    {
        otv[i] = (koren * otv[i - 1]) + f[i];
    }
    result = otv;
    return result;
}
//подставляет число в многочлен
double Podstanovka(double[] xar, double kor)
{
    double result = 0;
    for (int i = 0; i < xar.Length - 1; i++)
    {
        result += SQRN(kor, xar.Length - 1 - i) * xar[i];
    }
    result = result + xar[xar.Length - 1];
    return result;
}
int High(double[] xar)
{
    return xar.Length;
}
int High(double[,] xar)
{
    return xar.Length / 2;
}
//находит решение многочлена
double[] Resh(double[] xar)
{
    double[] result;
    int p;
    double xn, dx, xn1;
    double[] f1, otv = new double[xar.Length - 1];
    p = xar.Length;
    for (int i = 1; i < p; i++)
    {
        xn = 0.00001;
```

```
            f1 = (Proizv(xar));
            do
            {
                dx = -(Podstanovka(xar, xn)) / (Podstanovka(f1, xn));
                Console.WriteLine("dx = " + dx);
                xn1 = dx + xn;
                xn = xn1;
            } while (Math.Abs(dx) > 0.00001);
            //} while ((Math.Abs(dx) > 0.00001) || (Podstanovka(xar, xn1) != 0));
            xar = Delenie(xar, xn1);
            //SetLength(xar, High(xar));
            double[] xar2 = new double[xar.Length - 1];
            for (int l = 0; l < xar2.Length; l++)
            {
                xar2[l] = xar[l];
            }
            xar = xar2;
            otv[i - 1] = xn1;
            //Заполняем таблицу

            dataGridView2.Rows[i - 1].Cells[0].Value = xn1.ToString();
            //Form1.StringGrid2.Cells[1,i]:=FloatToStrF(xn1,ffExponent,6,13);
        }
        result = otv;
        return result;
    }
    //находит значение очередного не-известного, считая сумму
    //последующих элементов и деля её на элемент на главной диа-гонали
    double sum(double[,] Matr, double[] Mas, int p)
    {
        double result = 0;
        for (int i = p + 1; i < dataGridView1.ColumnCount; i++)
        {
            result += Matr[p, i] * Mas[i];
        }
        result = -result / Matr[p, p];
        return result;
    }
    bool Perest(double[,] Matr, int p, int i)
    {
        bool result = false;
        double rec;
        for (int u = p + 1; u < dataGridView1.ColumnCount; i++)
        {
            if (Matr[u, i] != 0)
            {
                for (int l = 0; l < dataGridView1.ColumnCount; l++)
                {
                    rec = Matr[p, l];
                    Matr[p, l] = Matr[u, l];
                    Matr[u, l] = rec;
                }
                result = true;
                break;
            }
        }
        return result;
    }
    //заменят все элементы в матрице меньше 0.0001 на 0
    double[,] Minim(double[,] Matr)
    {
        for (int i = 0; i < dataGridView1.ColumnCount; i++)
        {
            for (int j = 0; j < dataGridView1.ColumnCount; j++)
```

```
        {
            if (Math.Abs(Matr[i, j]) < 0.0001)
            {
                Matr[i, j] = 0;
            }
        }
    }
    return Matr;
}
// делается проверка, если решение до этого было выбрано любое,
//а теперь выясняется что оно не подходит, то оно заменяет-ся 0
void Prov(double[,] Matr, double[] b1, int k, int l)
{
    for (int i = l + 1; i < dataGridView1.ColumnCount; i++)
    {
        if (Matr[k, i] != 0)
        {
            b1[i] = 0;
        }
    }
}
//приводим матрицу к ступенчатому виду и нахо-дим любое частное решение
double[] Stup(double[,] Matr)
{
    double[] b1;
    double b;
    for (int i = 0; i < dataGridView1.ColumnCount - 1; i++)
    {
        for (int k = i + 1; k < dataGridView1.ColumnCount; k++)
        {
            if (Math.Abs(Matr[i, i]) == 0)
            {
                if (!Perest(Matr, i, i))
                {
                    break;
                }
            }
            b = -Matr[k, i] / Matr[i, i];
            for (int j = 0; j < dataGridView1.ColumnCount; j++)
            {
                double i_j = Matr[i, j];
                double k_j = Matr[k, j];
                //test = Matr[i, j] * b + Matr[k, j];
                Matr[k, j] = Matr[i, j] * b + Matr[k, j];
            }
            Matr = Minim(Matr);
        }
    }
    b1 = new double[dataGridView1.ColumnCount];
    for (int i = dataGridView1.ColumnCount - 1; i >= 0; i--)
    {
        if (Math.Abs(Matr[i, i]) == 0)
        {
            b1[i] = 1;
            Prov(Matr, b1, i, i);
        }
        else
        {
            b1[i] = sum(Matr, b1, i);
        }
    }
    return b1;
}
//копируем матрицу
```

```
        double[,] Copy1(double[,] Matr)
        {
            double[,] Matr1 = new double[dataGridView1.ColumnCount,
dataGridView1.ColumnCount];
            for (int i = 0; i < dataGridView1.ColumnCount; i++)
            {
                for (int j = 0; j < dataGridView1.ColumnCount; j++)
                {
                    Matr1[i, j] = Matr[i, j];
                }
            }
            return Matr1;
        }
        //копируем массив
        double[] CopyMas(double[] Mas)
        {
            double[] result = new double[Mas.Length + 1];
            for (int i = 0; i <= High(Mas); i++)
            {
                result[i] = Mas[i];
            }
            return Mas;
        }
        //очищаем массив
        double[] Clear1(double[] Mas)
        {
            double[] result = new double[Mas.Length + 1];
            for (int i = 0; i < Mas.Length; i++)
            {
                result[i] = 0;
            }
            return result;
        }
        //нормализуем массив
        void OutPut(double[] otv1, int p)
        {
            double s = 0;
            for (int i = 0; i < otv1.Length; i++)
            {
                s += Math.Pow(otv1[i], 2);
            }
            s = Math.Sqrt(s);

            for (int i = 0; i < otv1.Length; i++)
            {
                otv1[i] = otv1[i] / s;
                //Form1.StringGrid3.Cells[p,i+1]:=FloatToStrF(otv1[i],ffExponent,6,13);
                dataGridView3.Rows[i].Cells[p - 1].Value = otv1[i].ToString();
            }
        }
        //находим собственные вектора для соб-ственных значений
        void SobVect(double[,] Matr, double[] otv)
        {
            double[,] Matr1;
            double[] otv1 = new double[dataGridView1.ColumnCount + 1];
            for (int k = 0; k < otv.Length; k++)
            {
                Matr1 = Copy1(Matr);
                for (int i = 0; i < otv.Length; i++)
                {
                    Matr1[i, i] = Matr[i, i] - otv[k];
                }
                Matr1 = Minim(Matr1);
                //otv1 = Clear1(otv1);
```

```
                otv1 = Stup(Matr1);
                OutPut(otv1, k + 1);
            }
        }
        void Leverre(double[,] Matr)
        {
            double[,] Matr1;
            double[] s = new double[dataGridView1.ColumnCount], p = new
double[dataGridView1.ColumnCount + 1];
            Matr1 = Copy1(Matr);
            for (int i = 0; i < s.Length; i++)
            {
                s[i] = 0;
                if (i != 0)
                {
                    Matr1 = MatrUmn(Matr1, Matr, s.Length, s.Length, s.Length);
                }
                for (int j = 0; j < s.Length; j++)
                {
                    s[i] = s[i] + Matr1[j, j];
                }
            }
            for (int i = 0; i < s.Length; i++)
            {
                p[i + 1] = s[i];
                for (int j = 0; j <= i - 1; j++)
                {
                    p[i + 1] = p[i + 1] + p[j + 1] * s[i - j - 1];
                }
                p[i + 1] *= (-1.0 / (i + 1));
                Console.WriteLine("-1/(i+1) = " + (-1 / (i + 1)));
            }
            p[0] = 1;
            p = Resh(p);
            SobVect(Matr, p);
        }
        void Fadeev(double[,] Matr)
        {
            double[,] Matr1, Matr2;
            double[] s = new double[dataGridView1.ColumnCount], p = new
double[dataGridView1.ColumnCount + 1];
            Matr1 = Copy1(Matr);
            Matr2 = Copy1(Matr);
            for (int i = 0; i < dataGridView1.ColumnCount; i++)
            {
                s[i] = 0;
                for (int j = 0; j < s.Length; j++)
                {
                    s[i] = s[i] + Matr2[j, j];
                }
                s[i] = s[i] / (i + 1);
                Matr1 = Copy1(Matr2);
                for (int j = 0; j < dataGridView1.ColumnCount; j++)
                {
                    Matr1[j, j] = Matr2[j, j] - s[i];
                }
                Matr2 = MatrUmn(Matr, Matr1, dataGridView1.ColumnCount,
dataGridView1.ColumnCount, dataGridView1.ColumnCount);
            }
            for (int i = 0; i < s.Length; i++)
            {
                p[i + 1] = -s[i];
            }
            p[0] = 1;
```

```
            p = Resh(p);
            SobVect(Matr, p);
        }
        // находим произведение матрицы на массив
        double[] MatrUmnMas(double[,] a, double[] b, int n, int m, int k)
        {
            double s;
            double[] c = new double[n];
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < k; j++)
                {
                    s = 0;
                    for (int l = 0; l < m; l++)
                    {
                        s += a[i, l] * b[l];
                    }
                    c[i] = s;
                }
            }
            return c;
        }
        //копируем и транспонируем матрицу, удаляя последнюю строчку
        double[,] CopyTrans(double[,] a)
        {
            double[,] result = new double[dataGridView1.ColumnCount,
dataGridView1.ColumnCount];
            for (int i = 0; i < dataGridView1.ColumnCount; i++)
            {
                for (int j = 0; j < dataGridView1.ColumnCount; j++)
                {
                    result[i, j] = a[j + 1, i];
                }
            }
            return result;
        }
        void Krilov(double[,] Matr)
        {
            double[] p = new double[dataGridView1.ColumnCount + 1];
            int l = 1;
            double[,] a = new double[dataGridView1.ColumnCount - 1,
dataGridView1.ColumnCount - 1], y = new double[dataGridView1.ColumnCount + 1,
dataGridView1.ColumnCount];
            double[] b = new double[dataGridView1.ColumnCount + 1];
            do
            {
                a = Copy1(Matr);
                Random rand = new Random();
                for (int j = 0; j < dataGridView1.ColumnCount; j++)
                {

                    do
                    {
                        y[dataGridView1.ColumnCount, j] = Math.Truncate(l * 10 *
Convert.ToDouble(rand.Next(100)) / 100);
                    } while (y[dataGridView1.ColumnCount, j] < 5);
                }
                l++;
                double[] temp = new double[dataGridView1.ColumnCount];
                for (int i = 1; i <= dataGridView1.ColumnCount; i++)
                {
                    for (int j = 0; j < dataGridView1.ColumnCount; j++)
                    {
                        temp[j] = y[dataGridView1.ColumnCount - i + 1, j];
```

```
                    }
                    temp = MatrUmnMas(a, temp, dataGridView1.ColumnCount,
dataGridView1.ColumnCount, dataGridView1.ColumnCount);
                    for (int j = 0; j < dataGridView1.ColumnCount; j++)
                    {
                        y[dataGridView1.ColumnCount - i, j] = temp[j];
                    }
                }
                for (int j = 0; j < dataGridView1.ColumnCount; j++)
                {
                    b[j] = y[0, j];
                }
                a = CopyTrans(y);
            } while (Det(a, dataGridView1.ColumnCount) == 0);
            double[,] a2 = new double[dataGridView1.ColumnCount,
dataGridView1.ColumnCount + 1];
            for (int k = 0; k < dataGridView1.ColumnCount; k++)
            {
                for (int j = 0; j < dataGridView1.ColumnCount; j++)
                {
                    a2[k, j] = a[k, j];
                }
            }
            a = a2;
            Lab1 lb1 = new Lab1();
            b = lb1.StartGauss(a, b, b.Length - 1);
            for (int i = 1; i < p.Length; i++)
            {
                p[i] = -1.0 * b[i - 1];
            }
            p[0] = 1;
            p = Resh(p);
            SobVect(Matr, p);
        }
        double[,] SetLength(double[,] ar, int n, int k)
        {
            double[,] new_ar = new double[n, k];
            for (int i = 0; i <= n - 1; i++)
            {
                for (int j = 0; j <= k - 1; j++)
                {
                    new_ar[i, j] = ar[i, j];
                }

            }
            return new_ar;
        }
        double[] SetLength(double[] ar, int n)
        {
            double[] new_ar = new double[n];
            for (int i = 0; i <= ar.Length; i++)
            {
                new_ar[i] = ar[i];
            }
            return new_ar;
                                                    }
```

Приложение Е

Исходный код лабораторной работы №6

```
void QR(int n, double[,] A)
        {
            double[,] Q = new double[n, n];
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    if (i == j)
                    {
                        Q[i, j] = 1;
                    }
                    else
                    {
                        Q[i, j] = 0;
                    }
                }
            }

            double[,] QH = new double[n, n];
            double[,] R = new double[n, n];

            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    R[i, j] = A[i, j];
                }
            }

            double[,] QR_A = new double[n, n];
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    QR_A[i, j] = A[i, j];
                }
            }


            int QRk = 0;
            int k = 0;

            while (QRk < 10)
            {

                for (int i = 0; i < n; i++)
                {
                    for (int j = 0; j < n; j++)
                    {
                        R[i, j] = QR_A[i, j];
                    }
                }

                k = 0;
                for (int i = 0; i < n; i++)
                {
                    for (int j = 0; j < n; j++)
                    {
                        if (i == j)
                        {
```

```
                        Q[i, j] = 1;
                    }
                    else
                    {
                        Q[i, j] = 0;
                    }

                }
            }

            while (k < n - 1)
            {
                double[] a = new double[n - k];

                for (int i = k; i < n; i++)
                {
                    a[i - k] = R[i, k];
                }

                double[] u = new double[n - k];
                u = reflectionVector(a, n - k);

                double[,] U = new double[n - k, n - k];
                U = reflectionMatrix(u, n - k);

                QH = addMatrix(U, k, n);

                Q = matrixMult(Q, QH, n);

                R = matrixMult(QH, R, n);

                k += 1;
            }

            QR_A = matrixMult(R, Q, n);

            //Console.WriteLine("Итерация номер {0}", QRk + 1);
            label1.Text = "Номер итерации: " + (QRk + 1);

            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    //Console.Write("{0:0.0000}\t\t", QR_A[i, j]); //matrixMult( Q,
R, n)
                    dataGridView2.Rows[i].Cells[j].Value = QR_A[i, j];
                }
                //Console.WriteLine();
            }

            //Thread.Sleep(1000);

            QRk += 1;

        }

        Console.ReadLine();

    }

    // нахождение нормы вектора
    public static double vectorNorm(double[] a, int n)
    {
        double sum = 0;
```

```
            for (int i = 0; i < n; i++)
            {
                sum += Math.Pow(a[i], 2);
            }
            return Math.Sqrt(sum);
        }

        // нахождение вектора отражения
        public static double[] reflectionVector(double[] a, int n)
        {
            double[] u = new double[n];

            u[0] = a[0] - vectorNorm(a, n);

            for (int i = 1; i < n; i++)
            {
                u[i] = a[i];
            }

            return u;
        }

        // нахождение матрицы отражения
        public static double[,] reflectionMatrix(double[] u, int n)
        {
            double[,] reflMatrix = new double[n, n];

            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    if (i == j)
                    {
                        reflMatrix[i, j] = 1 - 2 * u[i] * u[j] / Math.Pow(vectorNorm(u,
n), 2);
                    }
                    else
                    {
                        reflMatrix[i, j] = -2 * u[i] * u[j] / Math.Pow(vectorNorm(u, n),
2);
                    }

                }
            }

            return reflMatrix;
        }

        // дополнение матрицы
        public static double[,] addMatrix(double[,] U, int k, int n)
        {

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

            for (int i = 0; i < k; i++)
            {
                for (int j = 0; j < k; j++)
                {
                    if (i == j)
                    {
                        fullMatrix[i, j] = 1;
                    }
                    else
                    {
```

```
                fullMatrix[i, j] = 0;
            }
        }
    }

    for (int i = k; i < n; i++)
    {
        for (int j = k; j < n; j++)
        {
            fullMatrix[i, j] = U[i - k, j - k];
        }
    }

    return fullMatrix;
}

// умножение матриц
public static double[,] matrixMult(double[,] x, double[,] y, int n)
{
    double[,] resultMatrix = new double[n, n];

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            for (int k = 0; k < n; k++)
            {
                resultMatrix[i, j] += x[i, k] * y[k, j];
            }
        }
    }
    return resultMatrix;
}
```

Приложение Ж

Исходный код лабораторной работы №7

```
double Lagr(int n, double[] x, double[] y, double q)
        {
            double result = 0;
            double s;
            for (int i = 0; i < n; i++)
            {
                s = 1;
                for (int j = 0; j < n; j++)
                {
                    if (j != i) { s = s * (q - x[j]) / (x[i] - x[j]); }
                }
                result += y[i] * s;
            }
            return result;
        }
        double[] progon(double[] a2, double[] b2, double[] c2, double[] d2, double[] u2,
double[] v2, int n2)
        {
            double[] result = new double[n2];
            a2[0] = 0;
            c2[n2 - 1] = 0;
            u2[0] = 0;
            v2[0] = 0;
            int i1, nm, nn, im;
            double zz;
            int i = 0;
            for (i = 0; i < n2; i++)
            {
                i1 = i + 1;
                zz = 1 / (b2[i] - a2[i] * v2[i]);
                v2[i1] = c2[i] * zz;
                u2[i1] = (a2[i] * u2[i] - d2[i]) * zz;
            }
            nm = n2 - 1;
            nn = n + 1;
            result[nm] = u2[nn];
            i = 0;
            for (int j = 0; j < nm; j++)
            {
                i = nn - j;
                im = i - 1;
                result[im] = v2[i] * result[i] + u2[i];
            }
            return result;

        }
        double spline(int n1, double[] x1, double[] y1, double[] a1, double[] b1,
double[] c1, double[] d1, double[] u1, double[] v1, double[] z1, double xx1, int ind1)
        {
            double result = 0.0, hj, hj1, am, al, t, t1, t12, s1, t2;
            int nm = n - 1;
            if (ind1 == 0)
            {
                a1[0] = 0;
                b1[0] = -2;
                c1[0] = 1;
                d1[0] = 3 * (y1[1] - y1[0]) / (x1[1] - x1[0]);
                for (int j = 1; j < nm; j++)
                {
                    hj = x1[j + 1] - x1[j];
```

```
                    hj1 = x1[j] - x1[j - 1];
                    am = hj1 / (hj1 + hj);
                    al = 1 - am;
                    a1[j] = al;
                    b1[j] = -2;
                    c1[j] = am;
                    d1[j] = 3 * (am * (y1[j + 1] - y1[j]) / hj + al * (y1[j] - y1[j - 1])
/ hj1);
                }
                a1[0] = 1;
                b1[0] = -2;
                c1[0] = 0;
                d1[0] = 3 * (y1[n1] - y1[n1 - 1]) / (x1[n1] - x1[n1 - 1]);

                z1 = progon(a1, b1, c1, d1, u1, v1, n1);
            }
            else
            {
                int j;
                for (j = 1; j < n - 1; j++)
                {
                    if (x1[j] > xx) { break; }
                }
                t = (xx1 - x1[j - 1]) / (x1[j] - x1[j - 1]);
                t1 = (1 - t);
                hj = x1[j] - x1[j - 1];
                t2 = t * t;
                t12 = t1 * t1;
                s1 = y1[j - 1] * t12 * (1 + 2 * t);
                s1 = y1[j] * t2 * (3 - 2 * t) + s1;
                s1 = z1[j - 1] * hj * t * t12 + s1;
                result = s1 - z1[j] * hj * t2 * t1;
            }

            return result;

        }

        double Newton(int n, double[] a, double[] b, double x)
        {
            double result, s, p;
            double[,] m = new double[n * 2, n * 2];
            for (int i = 1; i < n; i++)
            {
                m[0, i - 1] = b[i] - b[i - 1];
            }
            for (int i = 1; i < n - 1; i++)
            {
                int f = 1;
                for (int k = i + 1; k >= 1; k--)
                {
                    f *= k;
                }
                for (int j = 1; j < n - i; j++)
                {
                    m[i, j - 1] = (m[i - 1, j] - m[i - 1, j - 1]) / f;
                }
            }
            if ((a[n - 1] - x) - (x - a[1]) < 0)
            {
                s = b[n - 1];
                for (int i = 0; i < n - 1; i++)
                {
                    p = 1;
```

```
                for (int j = 0; j <= i; j++)
                {
                    p = p * (x - a[n - j - 1]);
                }
                s = s + p * m[i, n - i - 2];
            }
            result = s;
        }
        else
        {
            s = b[0];
            for (int i = 0; i < n - 1; i++)
            {
                p = 1;
                for (int j = 0; j <= i; j++)
                {
                    p = p * (x - a[j]);
                }
                s = s + p * m[i, 0];
            }
            result = s;
        }
        return result;
    }
    double Kvadr(int n, double[] x, double[] y, double q)
    {
        double result = 0.0;
        double[,] a = new double[3, 3];
        double[] b = new double[3], c = new double[3];
        double s;
        a[0, 0] = n;
        s = 0;
        for (int i = 0; i < n; i++)
        {
            s = s + x[i];
        }
        a[0, 1] = s;
        a[1, 0] = s;
        s = 0;
        for (int i = 0; i < n; i++)
        {
            s = s + x[i] * x[i];
        }
        a[0, 2] = s;
        a[1, 1] = s;
        a[2, 0] = s;
        s = 0;
        for (int i = 0; i < n; i++)
        {
            s = s + x[i] * x[i] * x[i];
        }
        a[1, 2] = s;
        a[2, 1] = s;
        s = 0;
        for (int i = 0; i < n; i++)
        {
            s = s + x[i] * x[i] * x[i] * x[i];
        }
        a[2, 2] = s;
        s = 0;
        for (int i = 0; i < n; i++)
        {
            s = s + y[i];
        }
```

```
    b[0] = s;
    s = 0;
    for (int i = 0; i < n; i++)
    {
        s = s + y[i] * x[i];
    }
    b[1] = s;
    s = 0;
    for (int i = 0; i < n; i++)
    {
        s = s + y[i] * x[i] * x[i];
    }
    b[2] = s;
    int j = n;
    n = 3;
    Lab1 lb1 = new Lab1();
    b = lb1.StartGauss(a, b, n);
    return b[0] + b[1] * q + b[2] * q * q;
}
```

Приложение 3

Презентационный материал

Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Поволжский государственный университет телекоммуникаций и информатики»

Факультет Заочного Отделения
Кафедра «ПОУТС»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Программирование численных методов.

Часть 1

Разработал:
студент группы 26П
Елистратов Андрей Анатольевич

Руководитель:
ст. преп. каф. ПОУТС
Малахов С.В.

Самара, 2016

Актуальность

В рамках работы требуется разработать комплекс программ решения численных методов на языке программирования высокого уровня, который изучается студентами ПГУТИ.

Предметом данной бакалаврской работы является программирование численных методов.

Объектом исследования – задачи линейного программирования.

2

Введение

Широкая применяемость численных методов в совокупности друг с другом при разработке приложений, связанных с моделированием и исследованием процессов, протекающих в системах. Так же язык Pascal и Delphi используемые в лабораторном комплексе, морально устаревают, поэтому требуется наличие программного кода на более современном языке.

3

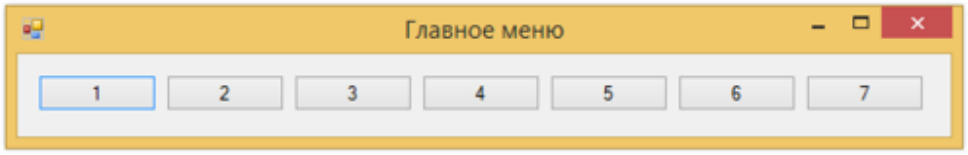
Язык программирования и среда разработки

C#, так как основной упор в университете акцентирован именно на этом языке программирования. Среда разработки **Microsoft Visual Studio**.

4

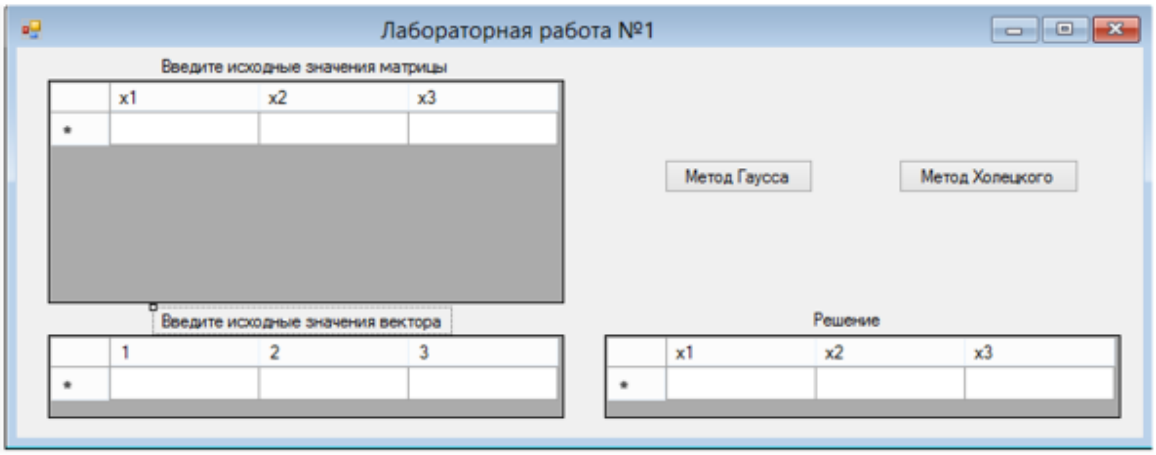
Требования технической части

1. Наличие интуитивного понятного графического интерфейса для ввода/вывода значений, полученных в результате вычисления конкретного метода
2. Применение такой парадигмы программирования, как ООП
3. Каждая лабораторная работа, содержащая в себе один или несколько численных методов, должна представлять собой отдельный класс.



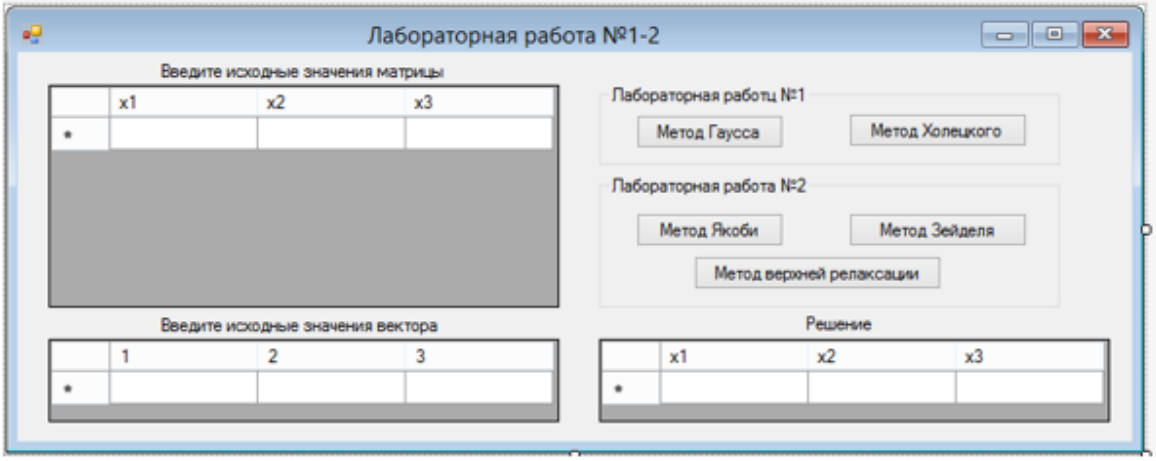
5

1. Решение систем линейных алгебраических уравнений точными методами: метод Гаусса и метод квадратных корней (Холецкого)

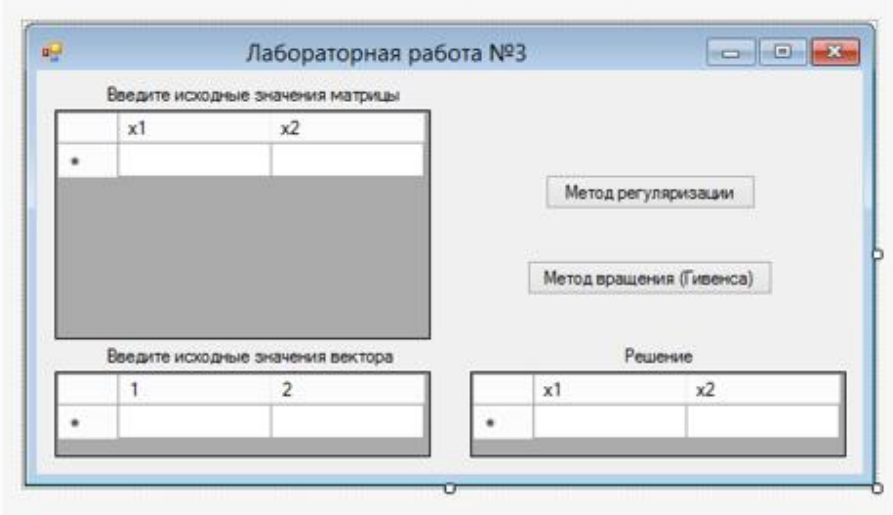


6

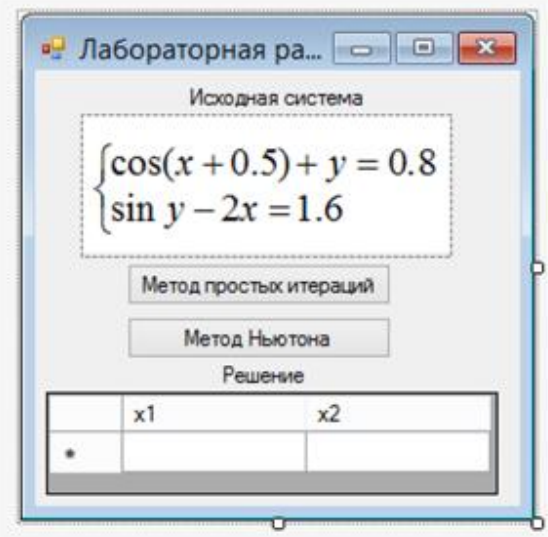
2. Решение систем линейных алгебраических уравнений итерационными методами: метод Якоби, метод Зейделя и метод верхней релаксации (обобщенный метод Зейделя)



3. Решение плохо обусловленных систем линейных алгебраических уравнений: метод регуляризации и метод вращения (Гивенса)

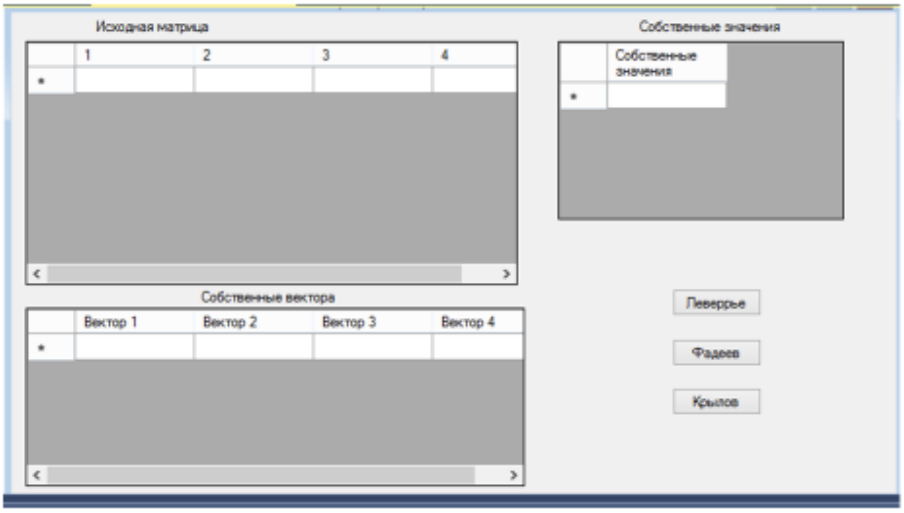


4. Решение нелинейных уравнений и систем нелинейных уравнений: метод простых итераций и метод Ньютона



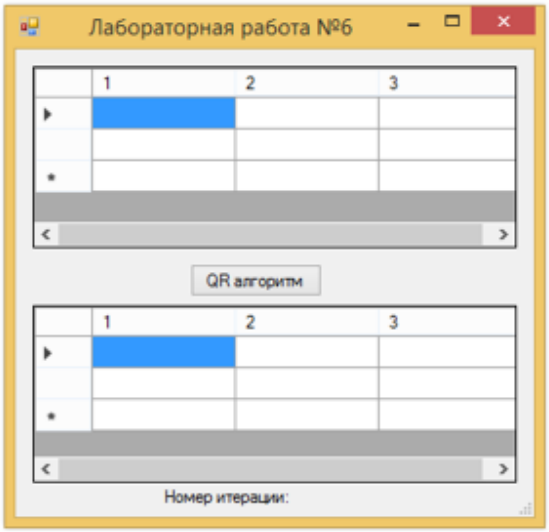
9

5. Решение проблемы собственных значений и собственных векторов: метод Леверрье, метод Фадеева и метод Крылова



10

6. Решение проблемы собственных значений и собственных векторов: метод QR-разложения и метод итераций



11

7. Приближение функций: интерполяционный полином Лагранжа, приближение полиномами Ньютона, интерполирование функций с помощью кубического сплайна и аппроксимация функций методом наименьших квадратов многочленом второй степени.

12

Основные результаты и краткие выводы

В ходе выполнения бакалаврской работы, было проделано следующее:
1) проведен теоретический обзор предметной области.
2) разработано ПО, которое подходит для применения в лабораторном комплексе по предмету «Численные методы» на кафедре ПОУТС;
3) в качестве среды разработки была выбрана MS Visual Studio, а язык программирования C#;
4) была проведена верификация данных полученных старым исходным кодом и новым, результаты совпали;
5) разработан графический интерфейс, который позволил отказаться от «консоли» и сконцентрировать внимание пользователя на результатах;
6) запрограммированы численные методы с первой по седьмую лабораторную работу;
7) описаны основные возможности программного обеспечения, такие как примеры решения численных методов и описание графического интерфейса.