СОДЕРЖАНИЕ

Введение

Многие прикладные задачи, при их численном анализе, сводятся к решению системы линейных арифметических уравнений (СЛАУ). В частности, к СЛАУ сводится решение некоторых интегральных уравнений, уравнений в частных производных, задачи линейной и нелинейной оптимизации и т.д.

При этом, как правило, для достижения удовлетворительной точности решения задачи требуется большое количество неизвестных и уравнений, входящих в систему. Аналитическое решение СЛАУ уже с четырьмя неизвестными сопряжено со значительными трудностями, в то время, как решение без использования ЭВМ систем в которые входят сотни уравнений не предоставляется возможным. Это обуславливает актуальность разработки алгоритмов и реализующих их программ для решения данной задачи.

Большинство современных языков программирования, таких как C#, C++, Visual Basic и пр., являются объектно-ориентированными (ООП). Суть ООП состоит в том, чтобы обращаться с данными и процедурами, которые выполняют действия над этими данными, как с единым объектом, т.е. самодостаточным элементом, который в чём-то идентичен другим таким же объектам, но в то же время отличается от них определёнными уникальными свойствами. В данной работе особенности ООП применены для построения алгоритма и решения СЛАУ четвёртого порядка.

1. Решение СЛАУ

1.1 Метод Крамера

В общем виде система линейных алгебраических уравнений имеет вид:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases} \quad (1)$$

Часто, предоставляется удобным матричное представление СЛАУ:

$$AX = B \quad (2)$$

где A – матрица, составленная из коэффициентов a_{nm}, B и X – соответствующие вектора, составленные из свободных членов системы b_n и неизвестных x_n.

Существует большое количество для решения СЛАУ. Но все они делятся на две группы: прямые методы решения и итерационные. В прямых методах точное решение системы получается путём применения к системе конечного количества определяемых алгоритмом шагов. К таким методам относятся метод Крамера, метод Жордана-Гаусса и т.д.

В итерационных методах задаются начальным приближением решения, а на каждом шаге (итерации) получается уточнённое решение. Процесс продолжается до тех пор, пока отличие между векторами X на соседних итерациях отличалось на достаточно малую величину. К итерационным методам относятся метод Якоби, метод Гаусса-Зейделя и пр.

Преимущество итерационных методов перед прямыми заключается в меньшем использовании ресурсов ЭВМ при решении СЛАУ больших размерностей. Но их недостатком есть сходимость решения лишь для ограниченного класса систем.

Для СЛАУ небольшой размерности широкое распространение имеет метод Крамера. Достоинством данного метода является его простота и наглядность. Кроме того, метод позволяет получить желаемые неизвестные, а не вектор X целиком. Данное обстоятельство делает его применение удобным для широкого спектра прикладных задач. Например, при анализе линейных электрических цепей часто интересуют токи лишь в отдельных ветвях схемы. Таким образом, является целесообразным применение метода Крамера к системе уравнений, составленной на основе законов Кирхгофа.

Согласно данному методу, корни СЛАУ находятся из следующих соотношений:

$$x_1 = \frac{\Delta_1}{\Delta}$$
$$x_2 = \frac{\Delta_2}{\Delta} \quad (3)$$
$$\dots$$
$$x_n = \frac{\Delta_n}{\Delta}$$

где Δ – определитель матрицы A, Δ_k – определитель матрицы, которая получена заменой k-го столбца матрицы A на вектор свободных членов B. Так, например:

$$\Delta_1 = \begin{vmatrix} b_1 & a_{12} & \dots & a_{n1} \\ b_2 & a_{22} & \dots & a_{n2} \\ \dots & \dots & \dots & \dots \\ b_n & a_{n2} & \dots & a_{nn} \end{vmatrix} \quad (4)$$

$$\Delta_2 = \begin{vmatrix} a_{11} & b_1 & \dots & a_{n1} \\ a_{21} & b_2 & \dots & a_{n2} \\ \dots & \dots & \dots & \dots \\ a_{n1} & b_n & \dots & a_{nn} \end{vmatrix} \quad (5)$$

и так далее.

Таким образом, применение метода Кирхгофа сводится к расчёту соответствующих определителей.

1.2 Методы расчёта определителя матрицы

Существует несколько способов расчёта определителя квадратной матрицы. Одним из них является сведение матрицы к треугольному виду, когда все элементы ниже главной диагонали равны нулю и нахождение произведения диагональных элементов. Данный способ сведения преобразования матрицы лежит в основе метода Гаусса для решения СЛАУ.

Кроме того, одним из наиболее распространённых методов рссчёта определителя есть его разложение по алгебраическим дополнениям выделенной строки или столбца. Данный метод и был применён в настоящей работе.

Алгебраическим дополнением элемента a_{ij} матрицы A является число:

$$A_{ij} = (-1)^{i+j} \Delta_{ij} \quad (6)$$

где Δ_{ij} – определитель матрицы, которая составлена из элементов исходной матрицы A за исключением элементов, входящих в j столбец и i строку. Заметим, что размерность данной матрицы на единицу меньше размерности исходной.

Определитель матрицы раскладывается следующим образом (по k-й строке):

$$|A| = a_{k1}A_{k1} + a_{k2}A_{k2} + \ldots + a_{kn}A_{kn} \quad (7)$$

В качестве иллюстрации приведём разложение матрицы размерностью 3×3 по элементам первого столбца:

$$\begin{vmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}\begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{21}\begin{vmatrix} a_{21} & a_{31} \\ a_{32} & a_{33} \end{vmatrix} + a_{13}\begin{vmatrix} a_{21} & a_{31} \\ a_{22} & a_{23} \end{vmatrix} \quad (8)$$

Определитель матрицы 2×2 рассчитывается как:

$$\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{21}a_{12} \quad (9)$$

Для данного метода расчета определителя хорошо подходят рекурсивные методы вычисления.

2. Программная реализация метода Крамера для решения СЛАУ четвёртого порядка на языке C#

2.1 Класс Determ

Данный класс предназначен для хранения и обработки квадратной матрицы A размерностью 4×4, которая составлена из коэффициентов СЛАУ. Класс содержит единственное приватное поле – массив 4×4:

```
private double[,] A=new double[4,4];
```

Конструктор класса:

```
public Determ()
    {
       int n, m;
       for (n = 0; n < 4; n++)
       {
          for (m = 0; m < 4; m++)
          {
             A[n, m] = 0;
          }
       }
            }
```

Конструктор инициализирует поля матрицы нулями.
Метод, осуществляющий загрузку матрицы из текстового файла:

```
public void A_file_in(string FName)
    {
       char[] spl = new char[1];
       spl[0] = ' ';
       StreamReader sr = new StreamReader(Fname);
       int n = 0;
       while (true)
       {
          string s = sr.ReadLine();
          if (s == null)
             break;
          string[] s_spl = s.Split(spl);
          for (int m = 0; m < 4; m++)
          {
             A[m, n] = Convert.ToDouble(s_spl[m]);
```

```
        }
        n++;
    }
    sr.Close();
        }
```

При обращении к данному методу в качестве строковой переменной ему передаётся имя исходного текстового файла. Метод является публичным, т.к. должен быть доступен из главной процедуры в консольном приложении и процедуре – обработки события нажатия на соответствующую кнопку в оконном приложении. В методе реализуется последовательное чтение четырёх строк, их разбиение по символу *пробел* и сохранение в соответствующие поля массива.

Для консольного приложения реализован метод сохранения матрицы при вводе вектора с клавиатуры:

```
public void A_cons_in()
    {
        for (int m = 0; m < 4; m++)
        {
            Console.Write("Введите " + Convert.ToString(m+1) + " строку");
            Console.Write("\n");
            for (int n = 0; n < 4; n++)
            {
                A[n, m] = double.Parse(Console.ReadLine());
            }
        }
            }
```

Для работы оконного приложения создан метод, который сохраняет значения элементов с формы:

```
public void A_form_in(string[] s)
    {
        char[] spl = new char[1];
        spl[0] = ' ';

        for (int n = 0; n < 4; n++)
        {
            string[] s_spl = s[n].Split(spl);
            for (int m = 0; m < 4; m++)
                A[m, n] = Convert.ToDouble(s_spl[m]);
        }
            }
```

При обращении к данному методу в качестве параметра ему передаётся массив строк – значения соответствующих коэффициентов матрицы. Строки разбиваются, конвертируются в числовой формат и сохраняются в соответствующих полях массива.

Расчёт определителя матрицы осуществляется с помощью следующих методов:

```
public double det()
    {
       double D = A[0, 0] * det_3(0, 0,A) – A[1, 0] * det_3(1, 0,A) + A[2, 0] * det_3(2, 0,A) – A[3, 0] * det_3(3, 0,A);
       return D;
    }

    private double det_3(int N, int M,double[,] T)
    {
       double[,] X = new double[3, 3];
       int n_s = 0;
       int m_s = 0;
       for (int n = 0; n < 4; n++)
       {
          m_s = 0;
          if (n == N)
             continue;
          for (int m = 0; m < 4; m++)
          {
             if (m == M)
                continue;
             X[n_s, m_s] = T[n, m];
             m_s++;
          }
          n_s++;
       }
       double D = X[0, 0] * det_2(0, 0, X) – X[1, 0] * det_2(1, 0, X) + X[2, 0] * det_2(2, 0, X);
       return D;
    }

    private double det_2(int N, int M,double[,] Y)
    {
       double[,] X = new double[2, 2];
       int n_s = 0;
       int m_s = 0;
       for (int n = 0; n < 3; n++)
       {
```

```
            m_s = 0;
            if (n == N)
               continue;
            for (int m = 0; m < 3; m++)
            {
               if (m == M)
                  continue;
               X[n_s, m_s] = Y[n, m];
               m_s++;
            }
            n_s++;
         }
         double D = X[0, 0] * X[1, 1] – X[0, 1] * X[1, 0];
         return D;
      }
```

Метод *det* имеет идентификатор *public* т.к он должен быть доступным из любого класса. Методы *det_3* и *det_2* доступны только из описываемого класса и служат для расчета определителей второго и третьего порядка при разложении по первому столбцу исходной матрицы.

Метод *det* имеет перегруженный вариант для расчёта определителя произвольной матрицы 4×4 и используется для определения чисел Δ_1, $\Delta_2\ldots$ и т.д. в формулах (3).

```
Protected double det(double[,] T)
      {
         double D = T[0, 0] * det_3(0, 0,T) – T[1, 0] * det_3(1, 0,T) + T[2, 0] *
det_3(2, 0,T) – T[3, 0] * det_3(3, 0,T);
         return D;
      }
```

Поскольку метод будет вызываться лишь при решении СЛАУ, он имеет идентификатор *protected*, который позволяет его использование только из наследственных классов.

Для передачи значений матрицы *A* без риска их изменения в наследственный класс для решения СЛАУ создан метод:

```
protected double[,] A_out()
      {
         return A;
      }
```

Поскольку в описываемом классе присутствуют операции чтения из файла, в его заголовке дополнительно подключено пространства имён *System.IO*.

2.2 Класс Eq_Calc

Данный класс служит для хранения коэффициентов СЛАУ, нахождения её решения и проверки результата расчёта. Класс является наследственным классу Determ, в котором содержатся методы для нахождения соответствующих определителей.

Класс имеет следующие приватные поля:

private double[] B = new double[4];
– вектор свободных членов СЛАУ.

Private double[] Dx = new double[4];
– массив определителей Δ_1, $\Delta_2 \ldots$

private double[] X=new double[4];
– вектор решения СЛАУ.

Private double[,] A = new double[4, 4];
– квадратная матрица коэффициентов СЛАУ.

Private bool sing = true;
– булева переменная, показывающая равенство/неравенство нулю определителя матрицы *A*.

Все поля класса являются личными для устранения риска их случайного изменения.

Конструктор класса инициализирует поля массива нулями:

```
public Eq_Calc()
    {
       for (int n = 0; n < 4; n++)
          B[n] = 0;
    }
```

Метод для ввода из файла вектора свободных членов:

```
public void B_file_in(string Fname)
    {
       char[] spl = new char[1];
       spl[0] = ' ';
       StreamReader sr = new StreamReader(Fname);
       int n = 0;
       while (true)
       {
```

```
            string s = sr.ReadLine();
            if (s == null)
               break;
            string[] s_spl = s.Split(spl);
            B[n] = Convert.ToDouble(s_spl[4]);
            n++;
         }
         sr.Close();
      }
```

Метод для консольного ввода вектора свободных членов:

```
public void B_cons_in()
      {
         Console.Write(«Введите свободные коэффициенты:»);
         Console.Write("\n");
         for (int n = 0; n < 4; n++)
         {
            B[n] = double.Parse(Console.ReadLine());
         }
      }
```

Метод для ввода данного вектора с формы (для оконного приложения):

```
public void B_form_in(string[] s)
      {
         for (int n = 0; n < 4; n++)
            B[n] = Convert.ToDouble(s[n]);
      }
```

Метод для расчёта определителей Δ_1, Δ_2 …:

```
private void Dx_Calc()
      {
         A = A_out();
         double[,] T = new double[4, 4];
         for (int k = 0; k < 4; k++)
         {
            for (int n = 0; n < 4; n++)
               for (int m = 0; m < 4; m++)
                  if (n == k)
                     T[n,m] = B[m];
                  else
                     T[n,m] = A[n,m];
```

```
            Dx[k] = det(T);
        }
    }
```

Метод для расчёта корней СЛАУ:

```
    public void X_Calc()
    {
        A = A_out();
        Dx_Calc();
        double D = det(A);
        if (D == 0)
        {
            sing = false;
            for (int n = 0; n < 4; n++)
                X[n] = 0;
                return;
        }
        for (int n = 0; n < 4; n++)
        {
            X[n] = Dx[n]/D;
        }
    }
```

Данный метод проверяет равенство нулю определителя Δ и в случае необходимости изменяет значение булевой переменной *sing*. Значение данной переменной выводится методом:

```
    public bool Sing()
    {
        return sing;
    }
```

Для проверки корректности полученного результата производится сравнение матрично-векторного произведения AX и вектора B. Поскольку точного совпадения значений элементов добиться невозможно из-за наличия ошибок округления при машинном расчёте, вектора принимаются равными при условии что разность каждой пары соответствующих элементов не превышает значения 0.001. Проверка осуществляется методами:

```
    private double[] Verify()
    {
        A = A_out();
        double[] V = new double[4];
        for (int n = 0; n < 4; n++)
```

```
            {
                V[n] = 0;
                for (int m = 0; m < 4; m++)
                {
                    V[n] += A[m, n] * X[m];
                }
            }

            return V;
        }
```

– который осуществляет соответствующее матрично-векторное произведение и:

```
public bool Verify_bool()
    {
        double[] V=Verify();
        double eps=0.001;
        for (int n = 0; n < 4; n++)
            if ((V[n]-B[n])>eps)
                return false;
        return true;
    }
```

– который осуществляет непосредственное поэлементное сравнение данных векторов.

Для вывода результатов расчёта в консольном приложении используется метод:

```
public void X_cons_out()
    {
        Console.Write("\n");
        Console.Write("X=");
        Console.Write("\n");
        for (int n = 0; n < 4; n++)
        {
            Console.Write(X[n]);
            Console.Write("\n");
        }
    }
```

Для вывода соответствующих данных в оконном приложении:

```
public double[] X_form_out()
        {
            return X;
        }
```

Как и предыдущий, данный класс использует операции ввода из файла-потока. Для этого в его заголовке дополнительно подключено пространство имён *System.IO*.

2.3 Алгоритм работы консольного приложения

При запуске консольного приложения пользователю предлагается выбрать способ ввода данных: ввод вектора с клавиатуры или из файла-потока. При выборе способа ввода с клавиатуры пользователь построчно вводит коэффициенты матрицы *A*. После чего предлагается ввести значения вектора *B*.

При выборе способа ввода СЛАУ из файла требуется указать путь к текстовому документу (или его имя, если файл находится в каталоге программы). В текстовом документе матрица 4×5, значения которой разделены пробелом, а дробная часть обозначается запятой. При этом, в последнем столбце хранится свободный вектор.

Далее программа производит расчет и выводит на экран искомый вектор *X*. Если определитель матрицы *A* равен нулю, на экран выводится соответствующее сообщение. После проведения проверки правильности решения выводится сообщение о его корректности.

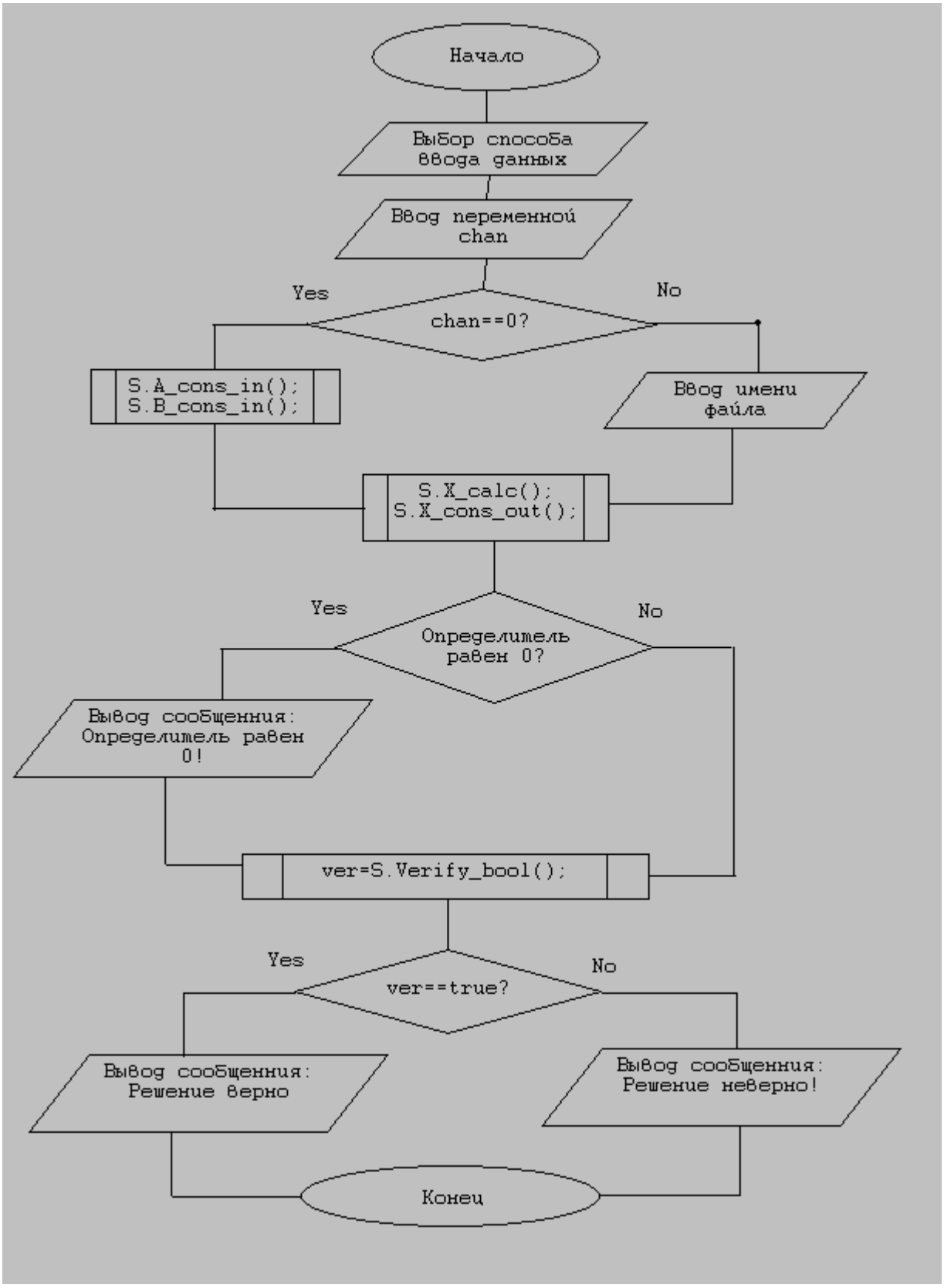

Рисунок 1. Блок схема работы консольного приложения

Листинг функции Main:

```
static void Main(string[] args)
    {
      Console.Write(«Выберите способ ввода СЛАУ:\n»);
```

```
    Console.Write(«[0] с клавиатуры\n»);
    Console.Write("[1] из файла\n");
    int chan;
    Eq_Calc S = new Eq_Calc();
    chan = Convert.ToInt16(Console.ReadLine());
    if (chan == 0)
    {
       S.A_cons_in();
       S.B_cons_in();

    }
    if (chan == 1)
    {
       Console.Write("Имя файла: ");
       string FName;
       Fname = Console.ReadLine();
       Console.Write("\n");
       S.A_file_in(Fname);
       S.B_file_in(Fname);
    }

    S.X_Calc();
    S.X_cons_out();

    if (!S.Sing())
    {
       Console.Write("\n");
       Console.Write(«Определитель равен нулю!!!»);
    }
    Console.Write("\n");
    bool ver = S.Verify_bool();
    if (ver)
    {
       Console.Write("Решение верно");
    }
    else
    {
       Console.Write("Решение НЕВЕРНО!!!");
    }
    Console.Write("\n");
    Console.ReadKey();
}
```

2.4 Описание работы оконного приложения

На форме имеются две кнопки: Ввод СЛАУ из файла и Ввод из формы. Алгоритм работы ввода значений из файла-потока повторяет соответствующий алгоритм в консольном приложении. Добавлен диалог выбора файла и выод результатов расчёта и проверки на форму. Листинг метода вызываемого нажатием данной кнопки:

```
private void btn_LoadFromFile_Click(object sender, EventArgs e)
    {
      OpenFileDialog ofd = new OpenFileDialog();
      ofd.Filter = "Text files|*.txt";
      if (ofd.ShowDialog() != DialogResult.OK)
        return;
      string FName = ofd.FileName;

      Eq_Calc S = new Eq_Calc();
      S.A_file_in(FName);
      S.B_file_in(FName);
      S.X_Calc();

      double[] X = new double[4];
      X = S.X_form_out();

      lbl_1.Text = "X_1 = " + X[0].ToString();
      lbl_2.Text = "X_2 = " + X[1].ToString();
      lbl_3.Text = "X_3 = " + X[2].ToString();
      lbl_4.Text = "X_4 = " + X[3].ToString();

      if (S.Sing())
      {
        if (S.Verify_bool())
          lbl_Verify.Visible = true;
        else
        {
          lbl_Verify.Text = "Система решена НЕВЕРНО!!!";
          lbl_Verify.Visible = true;
        }
      }
      else
      {
        lbl_Verify.Text = "Определитель матрицы \n равен нулю!!!";
        lbl_Verify.Visible = true;
      }
```

```
    }
```

При нажатии на кнопку загрузки данных с формы из соответствующих полей ввода загружаются значения коэффициентов. При вводе данные вводятся в соответствующие поля. Соседние элементы разделяются пробелом. Дробная часть отмечается запятой. Листинг метода обработки:

```
private void btn_from_Form_Click(object sender, EventArgs e)
    {
      Eq_Calc S = new Eq_Calc();
      string[] s = new string[4];

      s[0] = txt_a_1.Text;
      s[1] = txt_a_2.Text;
      s[2] = txt_a_3.Text;
      s[3] = txt_a_4.Text;

      S.A_form_in(s);

      s[0] = txt_b_1.Text;
      s[1] = txt_b_2.Text;
      s[2] = txt_b_3.Text;
      s[3] = txt_b_4.Text;

      S.B_form_in(s);

      S.X_Calc();

      double[] X = new double[4];
      X = S.X_form_out();

      lbl_1.Text = "X_1 = " + X[0].ToString();
      lbl_2.Text = "X_2 = " + X[1].ToString();
      lbl_3.Text = "X_3 = " + X[2].ToString();
      lbl_4.Text = "X_4 = " + X[3].ToString();

      if (S.Sing())
      {
        if (S.Verify_bool())
          lbl_Verify.Visible = true;
        else
        {
          lbl_Verify.Text = "Система решена НЕВЕРНО!!!";
          lbl_Verify.Visible = true;
        }
```

```
    }
    else
    {
        lbl_Verify.Text = "Определитель матрицы \n равен нулю!!!";
        lbl_Verify.Visible = true;
    }
}
```

3. Тестовый расчёт

Для проверки работоспособности программы был создан текстовый файл с коэффициентам СЛАУ:

1 2,2 6 9 3
5 6 4 9 7
5 6 9 2 4
1 6 8 3 9

И консольное, и оконное приложения выдают одинаковый результат расчета, который совпадает с истинным решением:

-1,11843913162957
2,05001374003847
-0,341852157186040
0,184391316295686

При вводе матрицы с данными:

0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0

Программа выдаёт сообщение о равенстве определителя нулю.

Данные тесты подтверждают корректность расчёта и правильность работы программы.

Выводы

На языке C# были созданы два класса, которые позволяют решать СЛАУ размерностью 4×4 и проверять корректность их решения. Данные классы использованы при написании консольного и оконного приложений осуществляющих данное решение и его проверку. В классах созданы методы для ввода значений коэффициентов СЛАУ из файла-потока.

Данные классы могут быть использованы при создании более сложных проектов.

Список использованной литературы

1. Фленов М.Е. Библия C#. – СПб: БХВ-Петербург, 2011. – 506 с.
2. Дейтел Х., Дейтел П. и др. C#. – СПб: БХВ-Петербург, 2006. – 1056 с.
3. Справочник по высшей математике / Под ред. А.А. Гусак. – Минск: ТетраСистемс, 1999. – 640 с.