Министерство Образования Республики Молдова

Технический Университет Молдовы

Департамент Программная Инженерия и Автоматика

Отчет

по лабораторной работе №2

Тема: ,, Решение систем линейных уравнений,,

Выполнил: ст.гр.SI-202 Абабий Эдуард

Проверил: Conf. Univ. Тутунару Е.Ф.

Кишинев – 2021

Лабораторная работа №2

Тема: Решение систем линейных уравнений

Цель: сформировать навыки решения систем линейных уравнений.

Задание:

Решить систему линейных алгебраических уравнений $\mathbf{Ax} = B$,где $\mathbf{A_{mxn}}$, $\mathbf{B_{mx1}}$, **m,n≤10,** A,B вводятся, методом Гаусса и Якоби.

Замечания:

1 Знать недостатки программы.

2. Для точных методах вывести расширенную матрицу на каждом шаге, для приближённых- расширенную матрицу системы подготовленную к применению метода и конечно ответ для каждого метода.

3. Протестировать программу для различных систем.

4. Оба метода в одной программе на выбор.

5. В конце предусмотреть возможность ввода другой системы или выход из программы.

1. Алгоритм метода Гаусса:

```
    while (true)
        {
            qInfo()<<"I work wich line "<< k-down_k;
            double rgt = matrix[k-down_k][k+up_k];
            for (int l = 0; l < columns+1;l++)
                {
                    if (rgt == 0)
                      {
                        double num_mod = rgt;
                        int line = 0;
                        for (int m = k-down_k; m < rows; m++)
                           {
                            if (fabs(matrix[m][k+up_k]) > fabs(num_mod))
                                {
                                 num_mod = matrix[m][k+up_k];
                                 line = m;
                                }
                           }
                        qInfo()<<line;
                        if (line == 0 and rgt == 0)
                             {
                            up_k++;
                            down_k++;
                            qInfo()<<"I am was here ----------------------";
                            break;
                        }
                            else
                            {
                            if (line != 0)
                                {
                                for (int m = 0;m < columns+1; m++)
                                    {
                                        double tmp = matrix[k-down_k][m];
                                        matrix[k-down_k][m] =
matrix[line][m];
                                        matrix[line][m] = tmp;
                                    }
```

```
                                        iteration++;
                                        if (iteration == iteratons)
                                                {
                                            for (int row = 0; row < model2-
>rowCount(); row++)
                                                    {
                                                for (int col = 0; col < model2-
>columnCount(); col++)
                                                        {
                                                    index = model-
>index(row,col);
                                                    model-
>setData(index,matrix[row][col]);
                                                    qInfo()<<matrix[row][col]<< "
--";
                                                }
                                                qInfo()<<"\n";
                                            }
                                           return;
                                        }
                                        rgt = matrix[k-down_k][k+up_k];
                                        matrix[k-down_k][l] = matrix[k-down_k][l]
/ rgt;
                                        qInfo()<< "RGT NUMBER "<<rgt;
                                    }
                                    //up_k++;
                                    //rgt = matrix[k][k+koef_k];
                                    //matrix[k][l] = matrix[k][l] / rgt;
                                    //qInfo()<<rgt<<" <<----- this is rgt
number";
                                }


                                        //matrix[k][l] = matrix[k][l] / rgt;
                                }
                                else
                                    {
                                matrix[k-down_k][l] = matrix[k-down_k][l] / rgt;
                                qInfo()<< "RGT NUMBER "<<rgt;
                            }

                        }
                    iteration++;
                    if (iteration == iteratons)
                            {
                        for (int row = 0; row < model2->rowCount(); row++)
                                {
                            for (int col = 0; col < model2->columnCount(); col++)
                                    {
                                index = model->index(row,col);
                                model->setData(index,matrix[row][col]);
                                qInfo()<<matrix[row][col]<< "  --";
                            }
                            qInfo()<<"\n";
                        }
                       return;
                    }
                    //if (k-down_k == rows - 1 )
                    //        {
                    //  break;
                    //}
                    qInfo()<<" JUST I NEED TO DO THIS CICLE?? " << rows-1-k+down_k;
                    for (int p = 0; p<rows-1-k+down_k;p++)
```

```
                {
                    double bmp = matrix[p+1+k-down_k][k+up_k];
                    for (int y = 0; y<columns+1;y++)
                    {
                                double tmp = matrix[k-down_k][y] * (bmp*(-
1));
                                matrix[k-down_k+1+p][y] = tmp + matrix[k-
down_k+1+p][y];
                                qInfo()<<"THIS LINE BE CHANGED ---> "<< k-
down_k+1+p<< "  "<<y;
                    }
                    iteration++;
                    if (iteration == iteratons)
                            {
                        for (int row = 0; row < model2->rowCount(); row++)
                                {
                            for (int col = 0; col < model2->columnCount();
col++)
                                    {
                                index = model->index(row,col);
                                model->setData(index,matrix[row][col]);
                                qInfo()<<matrix[row][col]<< "  --";
                            }
                            qInfo()<<"\n";
                        }
                       return;
                    }
                }
             k++;
            }
    for (int i = rows; i > 0; i--)
       {
         double tmp = 0;
         for (int j = 0; j < model2->columnCount()-1; j++)

             {
                 if (i-1 != j)
                   {
                     tmp+=matrix[i-1][j]*(-1) * matrix_wiht_roods[j];
                   }
             }
           matrix_wiht_roods[i-1] = (matrix[i-1][rows]+tmp)/matrix[i-1][i-1];
       }
    for (int i = 0; i < rows; i++)
            {
        ui->listWidget-
>addItem("X"+QString::number(i+1)+"="+QString::number(matrix_wiht_roods[i]));
        ui->listWidget->resize(131,columns*20);
  }
```

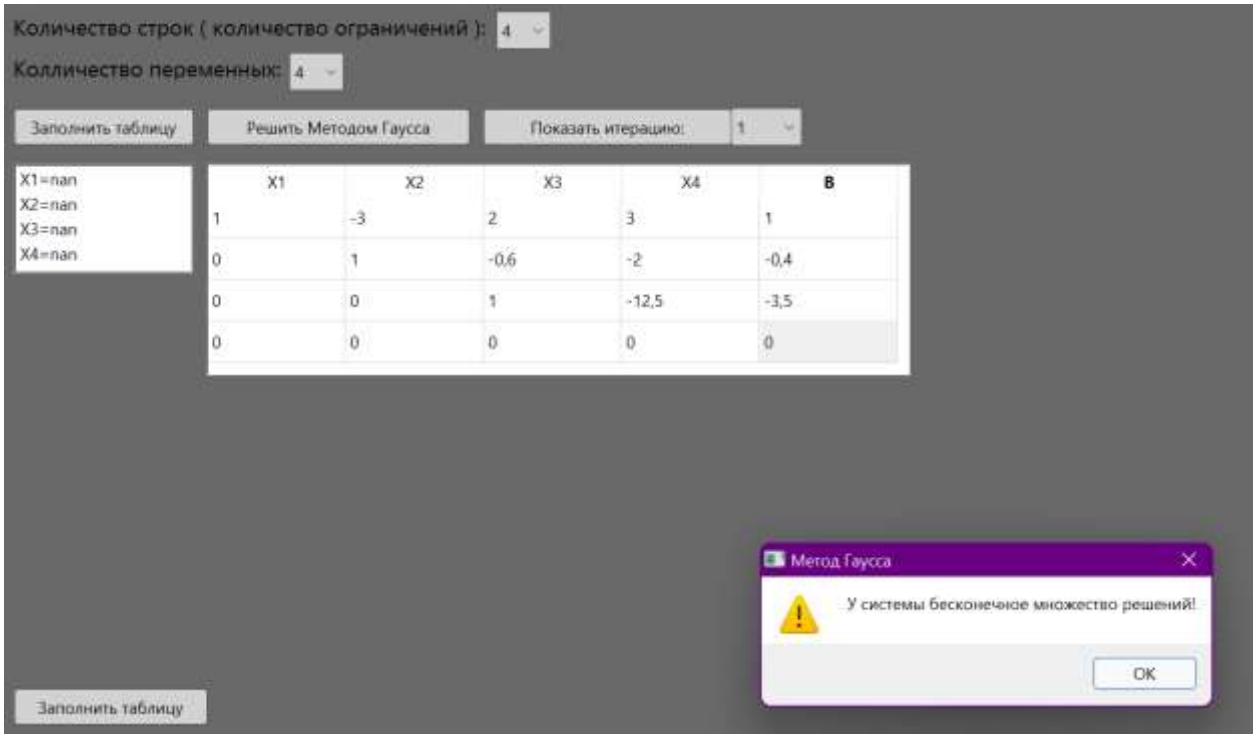
Решить систему нелинейных уравнений Методом Гаусса:

$X_1 - 3X_2 + 2X_3 + 3X_4 = 12$

$X_1 - X_2 + X_3 - 4X_4 = 0$

$2X_1 + X_2 - X_3 + 2X_4 = 2$

$3X_1 - 2X_2 + X_3 + 5X_4 = 3$



Алгоритм метода Якоби:

```
for (int i = 0; i < rows; i++)
            {
        double sum = 0;
        for (int j = 0; j < columns; j++)
                {
            if ( i != j)
                    {
                sum += fabs(matrix[i][j]);
            }
        }
        if (fabs(matrix[i][i])< sum)
                {
            critical = true;
            break;
        }
    }
    if (critical == true)
            {
        QMessageBox::critical(this,"Метод Якоби","Ошибка! Система не
сходится, решения нет");
        return;
    }
    else
    {
        normal = true;
```

```
    }
            int n = rows;
            double x1=0,x=0;
            bool exit_p = true;
            for (int i = 0; i < n; i++)
                {
                    matrix_wiht_roods_new[i] = 0;
                    matrix_wiht_roods_old[i] = 0;
                }
            for (int i = 0; i < rows; i++)
                    {
                for (int j = 0; j < columns+1; j++)
                        {
                    index = model->index(i,j);
                    index2 = model2->index(i,j);
                    matrix[i][j] = ui->tableView_3->model()-
>data(index).toDouble();
                    model2->setData(index2,matrix[i][j]);
                    qInfo()<<matrix[i][j]<<" ";
                }
                qInfo()<<"\n";
             }
            for (int i = 0; i < n;i++)
                {
                    for (int j = 0; j < n; j++)
                        {
                            if (i!=j)
                                {
                                    matrix[i][j]*=-1;
                                }
                        }
                }
            do {
                for (int i = 0; i < n; i++)
                {
                    matrix_wiht_roods_old[i] = matrix_wiht_roods_new[i];
                }
                for (int j = 0; j < n; j++)
                    {
                        double tmp = 0;
                        for (int i = 0; i<n; i++)
                            {
                                if (j != i)
                                    {
                                        tmp +=
matrix[j][i]*matrix_wiht_roods_old[i];
                                    }
                            }
                        matrix_wiht_roods_new[j] =
(matrix[j][n]+tmp)/matrix[j][j];
                    }
                x1 = 0;
                x = 0;
                //for (int i = 0; i < n;i++)
                //    {
                //        x1 += new_x[i];
                //        x += old_x[i];
                //        old_x[i] = new_x[i];
                //    }
                iteration++;
                exit_p = false;
                for (int i = 0; i < n; i++)
                {
```

```
                    if (fabs(matrix_wiht_roods_new[i] -
matrix_wiht_roods_old[i]) > eps)
                    {
                        exit_p = true;
                    }
                }
            } while (exit_p);
            for (int i = 0; i < rows; i++)
                    {
                ui->listWidget_2-
>addItem("X"+QString::number(i+1)+"="+QString::number(matrix_wiht_roods_new[i
]));
                ui->listWidget_2->resize(131,columns*20);
            }
            ui->listWidget_3->addItem(QString::number(iteration));
                    for (int i = 0; i < n; i++)
                {
                    qInfo()<< "X "<< i<< " = "<< matrix_wiht_roods_new[i];
                }
            if (normal == true)
                    {
                QMessageBox::information(this,"Метод Якоби","Решение
найдено");
            }
            qInfo()<< "Iteration = " << iteration;
}
```

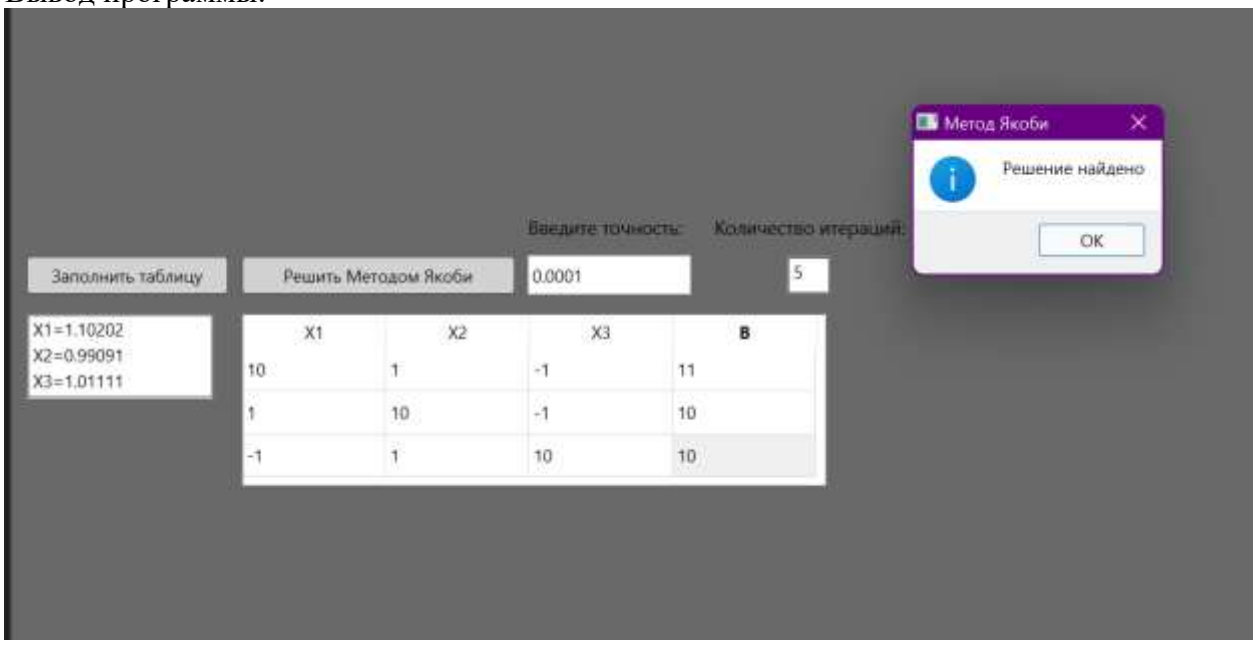
Решить систему нелинейных уравнений методом Якоби:

$10X_1 + X_2 - X_3 = 11$

$X_1 + 10X_2 - X_3 = 10$

$-X_1 + X_2 + 10X_3 = 10$

Вывод программы:



Вывод: В данной лабораторной работе я закрепил знания в решении систем линейных уравнений. Реализовал на ЭВМ метод решения систем линейных уравнений (Гаусс и Якоби).