ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
НИЖЕГОРОДСКИЙ ГОССУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р. Е. АЛЕКСЕЕВА

*Кафедра «Прикладная математика»*

Отчет

Лабораторная работа №3
по дисциплине «Численные Методы»

Тема: «Методы Якоби и Гаусса-Зейделя»

**Студент**

_____ Шохов М.Е.
(Подпись) (Фамилия, И., О.)

17-ПМ 18.12.19
(Группа) (Дата сдачи)

**Проверила**

_____ Талалушкина Л.В.
(Подпись) (Фамилия, И., О.)

Отчет защищен «___»_____2019г.

с оценкой_____

Нижний Новгород, 2019

# Оглавление

# 1. Цель работы

Изучить и программно реализовать методы решения системы линейных уравнений: метод Якоби, метод Гаусса-Зейделя на языках высокого уровня (C, Python, Matlab, Fortran), а также сравнить эти методы.

## 2. Пояснение к заданию

### 1. *Метод Якоби*

Возьмём систему уравнений:

$$A\vec{x} = \vec{b}, \text{ где } A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Для того, чтобы построить итеративную процедуру метода Якоби, необходимо провести предварительное преобразование системы уравнений $\boldsymbol{Ax = b}$ к итерационному виду $\boldsymbol{x = Bx + g}$. Оно может быть осуществлено по одному из следующих правил:

- $B = E - D^{-1}A = D^{-1}(D - A), \quad \vec{g} = D^{-1}\vec{b};$
- $B = -D^{-1}(L + U) = -D^{-1}(A - D), \quad \vec{g} = D^{-1}\vec{b}$
- $D_{ii}^{-1} = 1/D_{ii}, D_{ii} \neq 0, \; i = 1, 2, \dots, n \quad ;$

где в принятых обозначениях $\boldsymbol{D}$ означает матрицу, у которой на главной диагонали стоят соответствующие элементы матрицы $\boldsymbol{A}$, а все остальные нули; тогда как матрицы $\boldsymbol{U}$ и $\boldsymbol{L}$ содержат верхнюю и нижнюю треугольные части $\boldsymbol{A}$, на главной диагонали которых нули, $\boldsymbol{E}$ — единичная матрица.
Тогда процедура нахождения решения имеет вид:

$$\vec{x}^{(k+1)} = B\vec{x}^{(k)} + \vec{g},$$

Или в виде поэлементной формулы:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left( b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \dots, n.$$

где $\boldsymbol{k}$ счётчик итерации.

В отличие от метода Гаусса-Зейделя мы не можем заменять $\boldsymbol{x_i^{(k)}}$ на $\boldsymbol{x_i^{(k+1)}}$ в процессе итерационной процедуры, так как эти значения понадобятся для остальных вычислений. Это наиболее значимое различие между методом Якоби и методом Гаусса-Зейделя решения СЛАУ. Таким образом на каждой итерации придётся хранить оба вектора приближений: старый и новый.

## 2. *Метод Гаусса-Зейделя*

Возьмём систему:

$$A\vec{x} = \vec{b}, \text{ где } A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Чтобы пояснить суть метода, перепишем задачу в виде:

$$\begin{cases} a_{11}x_1 & = & -a_{12}x_2 - a_{13}x_3 - \ldots - a_{1n}x_n + b_1 \\ a_{21}x_1 + a_{22}x_2 & = & -a_{23}x_3 - \ldots - a_{2n}x_n + b_2 \\ \ldots & & \\ a_{(n-1)1}x_1 + a_{(n-1)2}x_2 + \ldots + a_{(n-1)(n-1)}x_{n-1} & = & -a_{(n-1)n}x_n + b_{n-1} \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{n(n-1)}x_{n-1} + a_{nn}x_n & = & b_n \end{cases}$$

Здесь в ***j***-м уравнении мы перенесли в правую часть все члены, содержащие $x_i$, для ***i*** > ***j***. Эта запись может быть представлена как:

$$(\mathrm{L} + \mathrm{D})\vec{x} = -\mathrm{U}\,\vec{x} + \vec{b},$$

где в принятых обозначениях ***D*** означает матрицу, у которой на главной диагонали стоят соответствующие элементы матрицы ***A***, а все остальные нули; тогда как матрицы ***U*** и ***L*** содержат верхнюю и нижнюю треугольные части ***A***, на главной диагонали которых нули.

Итерационный процесс в методе Гаусса — Зейделя строится по формуле:

$$(\mathrm{L} + \mathrm{D})\vec{x}^{(k+1)} = -\mathrm{U}\vec{x}^{(k)} + \vec{b}, \quad k = 0, 1, 2, \ldots$$

после выбора соответствующего начального приближения $\boldsymbol{x^{(0)}}$.

Метод Гаусса — Зейделя можно рассматривать как модификацию метода Якоби. Основная идея модификации состоит в том, что новые значения $x^{(i)}$ используются здесь сразу же по мере получения, в то время как в методе Якоби они не используются до следующей итерации:

$$\begin{cases} x_1{}^{(k+1)} & = & c_{12}x_2^{(k)} & + & c_{13}x_3^{(k)} & + & \ldots & + & c_{1n}x_n{}^{(k)} & + & d_1 \\ x_2{}^{(k+1)} & = & c_{21}x_1^{(k+1)} & + & c_{23}x_3^{(k)} & + & \ldots & + & c_{2n}x_n{}^{(k)} & + & d_2 \\ \quad \ldots & & & & & & & & & & \\ x_n{}^{(k+1)} & = & c_{n1}x_1^{(k+1)} & + & c_{n2}x_2^{(k+1)} & + & \ldots & + & c_{n(n-1)}x_{n-1}{}^{(k+1)} & + & d_n \end{cases},$$

где

$$c_{ij} = \begin{cases} -\frac{a_{ij}}{a_{ii}}, & j \neq i, \\ 0, & j = i. \end{cases} \quad d_i = \frac{b_i}{a_{ii}}, \quad i = 1, \ldots, n.$$

Таким образом, ***i***-я компонента ***(k+1)***-го приближения вычисляется по формуле:

$$x_i^{(k+1)} = \sum_{j=1}^{i-1} c_{ij}x_j^{(k+1)} + \sum_{j=i}^{n} c_{ij}x_j^{(k)} + d_i, \quad i = 1, \ldots, n.$$

## 3. Описание алгоритма

Основные функции:

- initializationFromFile() – считывает систему из файла и заполняет массивы coeff, freeTerm и задаёт переменную systemSize,
- showSystem() – выводит считанную из файла систему на экран,
- applyJakobyMethod() – производит поиск решения системы с заданной точностью методом Якоби и вывод решения на экран,
- applyZeidelMethod() – производит поиск решения системы с заданной точностью методом Зейделя и вывод решения на экран.

В функции *main()* объявляется объект класса *ZeidelAndJakoby*. После этого происходит вызов функции *initializationFromFile()*, считывающей из файла «mat.txt» систему и заполняющей массивы. Затем происходит вызов функции *showSystem()* с последующим отображением системы на экран (кроме Fortran). Далее поочерёдно применяются методы поиска решения *applyJakobyMethod()* и *applyZeidelMethod()* и вывод решения на экран.

Основные элементы:

- systemSize – размер системы уравнений,
- coeff – массив коэффициентов при неизвестных,
- freeTerm – массив свободных членов,
- filename – имя текстового файла, из которого осуществляется чтение системы,
- eps – необходимая для поиска решения точность (0.001)

Вывод программы: выводится система уравнений, считанная из файла, а затем решения, найденные при помощи методов Якоби и Зейделя.

| Исходная матрица | | | |
|---|---|---|---|
| Матрица коэффициентов A | | | Вектор b |
| 5 | -3 | 1 | 1 |
| 1 | -10 | 3 | 2 |
| -2 | 1 | 4 | -8 |

## 4. Реализация на языке C++

Реализация на языке C++

```
#include <iostream>
#include <fstream>
#include <math.h>

using namespace std;

class ZeidelAndJakoby{
private:
    int systemSize;
    double** coeff;
    double* freeTerm;
    double eps;
    string filename;
public:
    ZeidelAndJakoby();
    ~ZeidelAndJakoby();
    bool converge(double*,double*);
    void initializeFromFile();
    void change();
    void showSystem();
    void applyZeidelMethod();
    void applyJakobyMethod();
};

ZeidelAndJakoby::ZeidelAndJakoby()
{
    eps = 0.001;
    filename = "mat.txt";
}

ZeidelAndJakoby::~ZeidelAndJakoby()
{
    for (int i = 0; i < systemSize; i++)
        delete [] coeff[i];
    delete [] coeff;
    delete [] freeTerm;
}

bool ZeidelAndJakoby::converge(double *x, double *p)
{
    double* diff = new double[systemSize];
    double maxVal = 0;
    for (int i = 0; i < systemSize; i++)
    {
        diff[i] = fabs(x[i] - p[i]);
        if (diff[i] > maxVal)
            maxVal = diff[i];
    }
    return (maxVal<eps);
}
```

Реализация на языке C++

```
void ZeidelAndJakoby::initializeFromFile()
{
    ifstream fout(filename);

    fout >> systemSize;

    coeff = new double*[systemSize];
    freeTerm = new double[systemSize];

    for (int i = 0; i < systemSize; i++)
    {
        coeff[i] = new double[systemSize+1];
        for (int j = 0; j < systemSize; j++)
        {
            fout >> coeff[i][j];
        }

        fout >> freeTerm[i];
    }

    fout.close();
}

void ZeidelAndJakoby::showSystem()
{
    cout << "System: \n";
    for (int i = 0; i < systemSize; i++)
    {
        for (int j = 0; j < systemSize; j++)
        {
            if (coeff[i][j] < 0)
                cout << "(" << coeff[i][j] << ")*x" << j;
            else
                cout << coeff[i][j] << "*x" << j;
            if (j < systemSize - 1)
                cout << " + ";
        }
        cout << " = " << freeTerm[i] << endl;
    }
    return;
}

void ZeidelAndJakoby::applyZeidelMethod()
{
    cout << "--------ZEIDEL--------" << endl;

    double* p = new double[systemSize];
    double* x = new double[systemSize];

    for (int j = 0; j<systemSize; j++)
        x[j] = 1;
```

Реализация на языке C++

```
    int iterations = 0;
    do
    {
        for (int i = 0; i < systemSize; i++)
            p[i] = x[i];
        for (int i = 0; i < systemSize; i++)
        {
            double var = 0;
            for (int j = 0; j < i; j++)
                var += (coeff[i][j] * x[j]);
            for (int j = i + 1; j < systemSize; j++)
                var += (coeff[i][j] * p[j]);
            x[i] = (freeTerm[i] - var) / coeff[i][i];
        }
        iterations++;
    }
    while (!converge(x, p));

    cout << "Iterations:    " << iterations << endl;

    for (int i = 0; i<systemSize; i++)
        cout << "x[" << i << "] = " << x[i] << endl;

    delete [] p;
    delete [] x;
}

void ZeidelAndJakoby::applyJakobyMethod()
{
    cout << "--------JAKOBY--------" << endl;

    double* TempX = new double[systemSize];
    double norm;
    double* x = new double[systemSize];

    for (int j = 0; j<systemSize; j++)
        x[j] = 1;

    int iterations = 0;
    do {
        for (int i = 0; i < systemSize; i++) {
            TempX[i] = freeTerm[i];
            for (int g = 0; g < systemSize; g++) {
                if (i != g)
                    TempX[i] -= coeff[i][g] * x[g];
            }
            TempX[i] /= coeff[i][i];
        }
```

Реализация на C++

```
        norm = fabs(x[0] - TempX[0]);
        for (int h = 0; h < systemSize; h++) {
            if (fabs(x[h] - TempX[h]) > norm)
                norm = fabs(x[h] - TempX[h]);
            x[h] = TempX[h];
        }
        iterations++;
    } while (norm > eps);

    cout << "Iterations:    " << iterations << endl;

    for (int i = 0; i<systemSize; i++)
        cout << "x[" << i << "] = " << x[i] << endl;

    delete[] TempX;
}

int main()
{
    ZeidelAndJakoby solve;

    solve.initializeFromFile();
    solve.showSystem();
//    solve.change();
    solve.applyJakobyMethod();
    solve.applyZeidelMethod();

    return 0;
}
```

## 5. Реализация на языке Python

Реализация на языке Python

```
import numpy as np
import math

class ZeidelAndJakoby :
    def __init__ (self) :
        self.systemSize = 0
        self.coeff = []
        self.freeTerm = []
        self.filename = "mat.txt"
        self.eps = 0.001

    def initializationFromFile(self) :
        with open(self.filename) as file :
            self.coeff = [list(map(float, row.split())) for row in file.readlines()]
            self.systemSize = len(self.coeff[0])-1

        for i in range(0,self.systemSize) :
            self.freeTerm.append([self.coeff[i][-1]])
            self.coeff[i].pop()

    def showSystem(self) :
        print("Введенная система: ")
        for row in range(len(self.freeTerm)):
            for col in range(len(self.coeff[row])):
                print(f"{self.coeff[row][col]}*x{col}", end=")
                if col < self.systemSize - 1 :
                    print(" + ", end=")
            print(f" = {self.freeTerm[row][0]}")


    def applyJakobyMethod(self) :
        print("--------JAKOBY--------")
        npCoeff = np.asarray(self.coeff)
        npFreeTerm = np.asarray(self.freeTerm)

        tmp = []
        for i in range(self.systemSize) :
            tmp.append([])
            for j in range(self.systemSize) :
                tmp[i].append(0)
            tmp[i][i] = self.coeff[i][i]

        diagonalMatrix = np.asarray(tmp)
        inversedDiagMatrix = np.linalg.inv(diagonalMatrix)
```

Реализация на языке Python

```
        b = np.eye(self.systemSize) - np.dot(inversedDiagMatrix,npCoeff)
        d = np.dot(inversedDiagMatrix,npFreeTerm)

        currentX = np.array([[0]]*self.systemSize,dtype = float)
        previousX = np.array([[1]]*self.systemSize,dtype = float)

        iterations = 0
        while True :
            currentX = np.dot(b,previousX) + d

            diff = currentX - previousX

            maxValue = math.fabs(diff.max())
            minValue = math.fabs(diff.min())

            iterations += 1
            if maxValue > minValue and maxValue < self.eps :
                print(f"Number of iterations: {iterations}")
                for i in range(self.systemSize) :
                    print(f"x[{i}] = {currentX[i]}")
                break;
            elif maxValue < minValue and minValue < self.eps :
                print(f"Number of iterations: {iterations}")
                for i in range(self.systemSize) :
                    print(f"x[{i}] = {currentX[i]}")
                break;

            previousX = currentX

    def applyZeidelMethod(self) :
        print("--------ZEIDEL--------")
        npCoeff = np.asarray(self.coeff)
        npFreeTerm = np.asarray(self.freeTerm)

        triangMatrix = np.tril(npCoeff)
        inversedMatrix = np.linalg.inv(triangMatrix)

        b = np.eye(self.systemSize) - np.dot(inversedMatrix,npCoeff)
        d = np.dot(inversedMatrix,npFreeTerm)

        currentX = np.array([[0]]*self.systemSize,dtype = float)
        previousX = np.array([[1]]*self.systemSize,dtype = float)

        iterations = 0
        while True :
```

Реализация на языке Python

```
        currentX = np.dot(b,previousX) + d

        diff = currentX - previousX

        maxValue = math.fabs(diff.max())
        minValue = math.fabs(diff.min())

        iterations += 1
        if maxValue > minValue and maxValue < self.eps :
           print(f"Number of iterations: {iterations}")
           for i in range(self.systemSize) :
              print(f"x[{i}] = {currentX[i]}")
           break;
        elif maxValue < minValue and minValue < self.eps :
           print(f"Number of iterations: {iterations}")
           for i in range(self.systemSize) :
              print(f"x[{i}] = {currentX[i]}")
           break;

        previousX = currentX

if __name__ == '__main__':
   solve = ZeidelAndJakoby()

   solve.initializationFromFile()
   solve.showSystem()
   solve.applyJakobyMethod()
   solve.applyZeidelMethod()
```

# 6. Реализация на языке Fortran

Реализация на языке Fortran

```
module GaussMethod
    integer systemSize
    real, dimension (:,:), allocatable :: coeff
    real, dimension (:), allocatable :: freeTerm
    !real, dimension (:), allocatable :: solutions
    character(12) :: filename
    real eps
    !integer isSolutionFounded

    contains
        subroutine init()
            systemSize = 0
            filename = "mat.txt"
            eps = 0.001
            !isSolutionFounded = 0
        end subroutine init

        subroutine destruct()
            deallocate (coeff)
            deallocate (freeTerm)
            !deallocate (solutions)
        end subroutine destruct

        subroutine initializationFromFile()
            real tmp

            open(1,file=filename)

            do
                read(1,*, end=1) tmp
                systemSize = systemSize + 1
            end do

            1 close (1)

            allocate ( coeff(systemSize,systemSize+1) )
            allocate ( freeTerm(systemSize) )
            !allocate ( solutions(systemSize) )

            open(2,file=filename)

            do i = 1,systemSize
                read(2,*)(coeff(i,j),j=1,systemSize+1)
            end do

            do i = 1,systemSize
                freeTerm(i) = coeff(i,systemSize+1)
                coeff(i,systemSize+1) = 0
            end do

            close (2)
        end subroutine
```

Реализация на языке Fortran

```
        subroutine showSystem()
            write (*,*) 'P’PIPµPrC‘PSPSP°CЦ CЃPёCЃC,PµPjP°: '
            do i = 1, systemSize
                do j = 1, systemSize
                    if (coeff(i,j) < 0) then
                        write (*,*) '(', coeff(i,j), ')*x', j
                    else
                        write (*,*) coeff(i,j), '*x', j
                    end if
                    if (j < systemSize) then
                        write (*,*) ' + '
                    end if
                end do
                write (*,*) ' = ', freeTerm(i)
            end do

        end subroutine showSystem

        subroutine applyJakobyMethod()
            real, dimension (:), allocatable :: TempX
            real, dimension (:), allocatable :: x
            real norm
            integer iterations

            allocate( TempX(systemSize) )
            allocate( x(systemSize) )
            norm = 1
            iterations = 0

            print *, "---------JAKOBY--------"
            do i = 1, systemsize
                x(i) = 1
            end do

            do while (norm > eps)
                do i = 1,systemSize
                    TempX(i) = freeTerm(i)
                    do g = 1,systemSize
                        if (i .NE. g) then
                            TempX(i) = TempX(i) - coeff(i,g) * x(g)
                        end if
                    end do
                    TempX(i) = TempX(i) / coeff(i,i)
                end do

                norm = abs(x(1) - TempX(1))
                do h = 1,systemSize
                    if (abs(x(h) - TempX(h)) > norm) then
                        norm = abs(x(h) - TempX(h))
                    end if
                    x(h) = TempX(h)
                end do

                iterations = iterations + 1
```

Реализация на языке Fortran

```
            end do

            print *, "Iterations:    ",iterations
            do i = 1,systemSize
                print *, "x[",i,"] = ",x(i)
            end do

            deallocate (TempX)
            deallocate (x)
        end subroutine applyJakobyMethod

        integer function converge(x,p)
            real x(systemSize)
            real p(systemSize)
            real, dimension (:), allocatable :: diff
            real maxValue

            allocate( diff(systemSize) )
            maxValue = 0

            do i = 1,systemSize
                diff(i) = abs(x(i) - p(i))
                if (diff(i) > maxValue) then
                    maxValue = diff(i)
                end if
            end do

            deallocate(diff)
            if (maxValue < eps) then
                converge = 1
            else if (maxValue .GE. eps) then
                converge = 0
            end if
        end function converge

        subroutine applyZeidelMethod()
            real, dimension (:), allocatable :: p
            real, dimension (:), allocatable :: x
            real, dimension (:), allocatable :: diff
            real var
            integer iterations, tmp, maxValue


            allocate( p(systemSize) )
            allocate( x(systemSize) )
            allocate( diff(systemSize) )
            iterations = 0

            print *, "---------ZEIDEL--------"
            do i = 1, systemsize
                x(i) = 1
            end do
```

Реализация на языке Fortran

```
            maxValue = 1
            tmp = converge(x,p)
            do while (tmp .EQ. 0)
                do i = 1,systemSize
                    p(i) = x(i)
                end do

                do i = 1, systemSize
                    var = 0
                    do j = 1,i-1
                        var = var + (coeff(i,j) * x(j))
                    end do
                    do j = i+1,systemSize
                        var = var + (coeff(i,j) * p(j))
                    end do
                    x(i) = (freeTerm(i) - var) / coeff(i,i)
                end do

                !print *, x(1)

                iterations = iterations + 1
                tmp = converge(x,p)
            end do

            print *, "Iterations:    ",iterations
            do i = 1,systemSize
                print *, "x[",i,"] = ",x(i)
            end do

            deallocate (p)
            deallocate (x)
        end subroutine applyZeidelMethod
end module


program main
    use gaussMethod
    call init()

    call initializationFromFile()
    !call showSystem()
    call applyJakobyMethod()
    call applyZeidelMethod()
    !call showSolutions()

    call destruct()
end
```

# 7. Реализация на языке Matlab

Реализация на языке Matlab (файл ZeidelAndJakoby.m)

```
classdef ZeidelAndJakoby < handle
    properties
        systemSize = 0;
        coeff = [];
        freeTerm = [];
        filename = "mat.txt";
        eps = 0.001
    end

     methods
        function initializationFromFile(obj)
            m = 0;
            n = 0;

            obj.coeff = load(obj.filename);
            [m,n] = size(obj.coeff);

            obj.systemSize = m;

            for i = 1:obj.systemSize
               obj.freeTerm(i,1) = obj.coeff(i,obj.systemSize+1);
               obj.coeff(i,obj.systemSize+1) = 0;
            end
            obj.coeff(:,obj.systemSize+1)=[];
        end

        function showSystem(obj)
            disp('P’PIPµPґC‘PSPSP°CЦ CЃPёCЃC,PµPjP°:');
            for i = 1:obj.systemSize
                for j = 1:obj.systemSize
                    if (obj.coeff(i,j) < 0)
                        fprintf('(%f)*x%d',obj.coeff(i,j), j);
                    else
                        fprintf('%f*x%d',obj.coeff(i,j), j);
                    end

                    if (j < obj.systemSize)
                        fprintf( ' + ' );
                    end
                end
                fprintf(' = %d\n', obj.freeTerm(i));
            end
        end
```

Реализация на языке Matlab (файл ZeidelAndJakoby.m)

```
        function applyJakobyMethod(obj)
            disp("--------JAKOBY--------");
            diagonalElements = diag(obj.coeff);
            diagonalMatrix = diag(diagonalElements);
            inversedDiagMatrix = inv(diagonalMatrix);

            b = eye(obj.systemSize) -
mtimes(inversedDiagMatrix,obj.coeff);
            d = mtimes(inversedDiagMatrix,obj.freeTerm);

            currentX = zeros(obj.systemSize,1);
            previousX = ones(obj.systemSize,1);

            iterations = 0;
            while 1
               currentX = mtimes(b,previousX) + d;
               diff = currentX - previousX;

               maxValue = abs(max(diff));
               minValue = abs(min(diff));

               iterations = iterations + 1;
               if ( (maxValue > minValue) && (maxValue < obj.eps) )
                   fprintf ('Iterations:   %d\n',iterations);
                   for i = 1:obj.systemSize
                       fprintf ('x[%d] = %f\n',i, currentX(i));
                   end
                   break;
               elseif ( (maxValue < minValue) && (minValue < obj.eps)
)
                   fprintf ('Iterations:   %d\n',iterations);
                   for i = 1:obj.systemSize
                       fprintf ('x[%d] = %f\n',i, currentX(i));
                   end
                   break;
               end

               previousX = currentX;
            end

        end

        function applyZeidelMethod(obj)
            disp("--------ZEIDEL--------");
            triangMatrix = tril(obj.coeff);
            inversedMatrix = inv(triangMatrix);

            b = eye(obj.systemSize) -
mtimes(inversedMatrix,obj.coeff);
            d = mtimes(inversedMatrix,obj.freeTerm);
```

Реализация на языке Matlab (файл ZeidelAndJakoby.m)

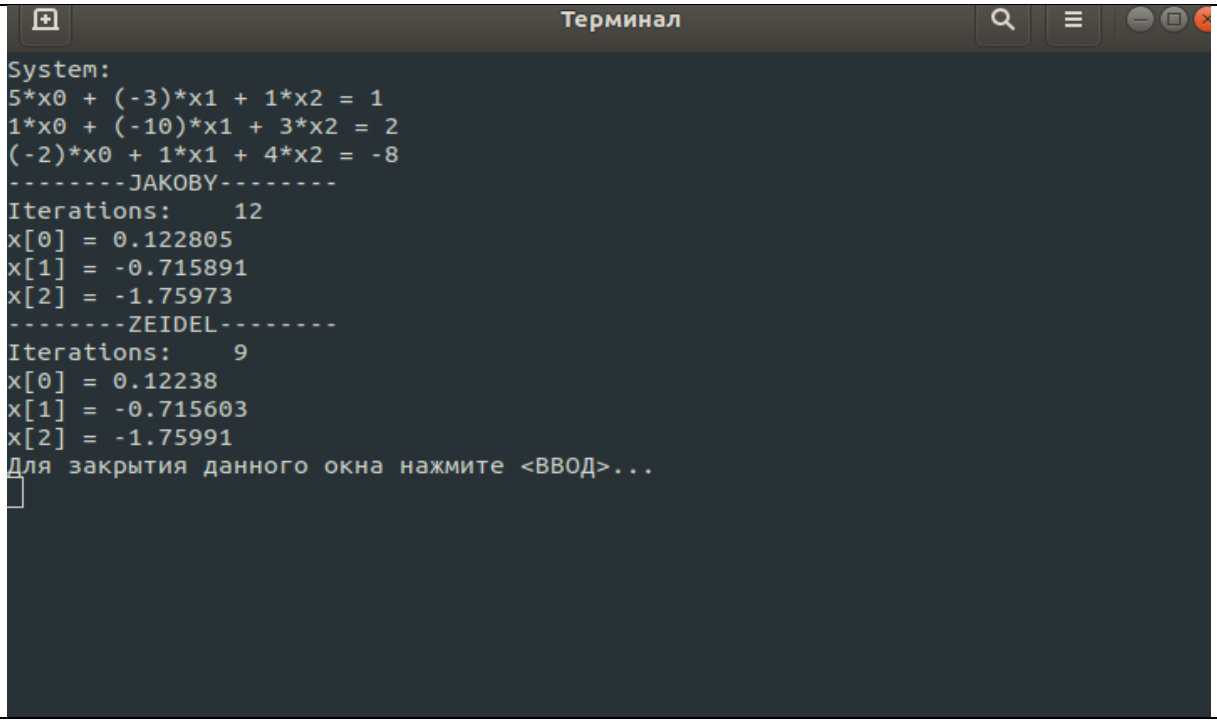
```
            currentX = zeros(obj.systemSize,1);
            previousX = ones(obj.systemSize,1);

            iterations = 0;
            while 1
               currentX = mtimes(b,previousX) + d;
               diff = currentX - previousX;

               maxValue = abs(max(diff));
               minValue = abs(min(diff));

               iterations = iterations + 1;
               if ( (maxValue > minValue) && (maxValue < obj.eps) )
                   fprintf ('Iterations:   %d\n',iterations);
                   for i = 1:obj.systemSize
                       fprintf ('x[%d] = %f\n',i, currentX(i));
                   end
                   break;
               elseif ( (maxValue < minValue) && (minValue < obj.eps)
)
                   fprintf ('Iterations:   %d\n',iterations);
                   for i = 1:obj.systemSize
                       fprintf ('x[%d] = %f\n',i, currentX(i));
                   end
                   break;
               end

               previousX = currentX;
            end
        end

    end
end
```

Реализация на языке Matlab (файл main.m)

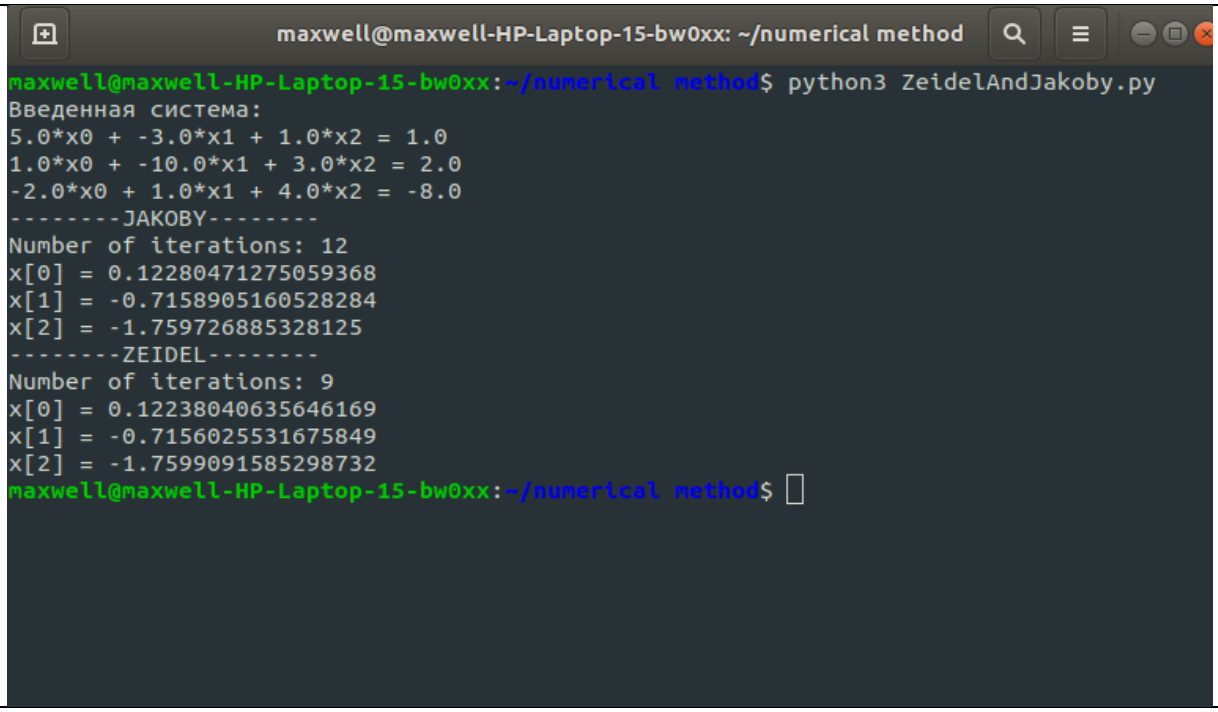
```
function main
    solve = ZeidelAndJakoby;

    initializationFromFile(solve);
    showSystem(solve);
    applyJakobyMethod(solve);
    applyZeidelMethod(solve);
end
```

## 8. Результат выполнения программ

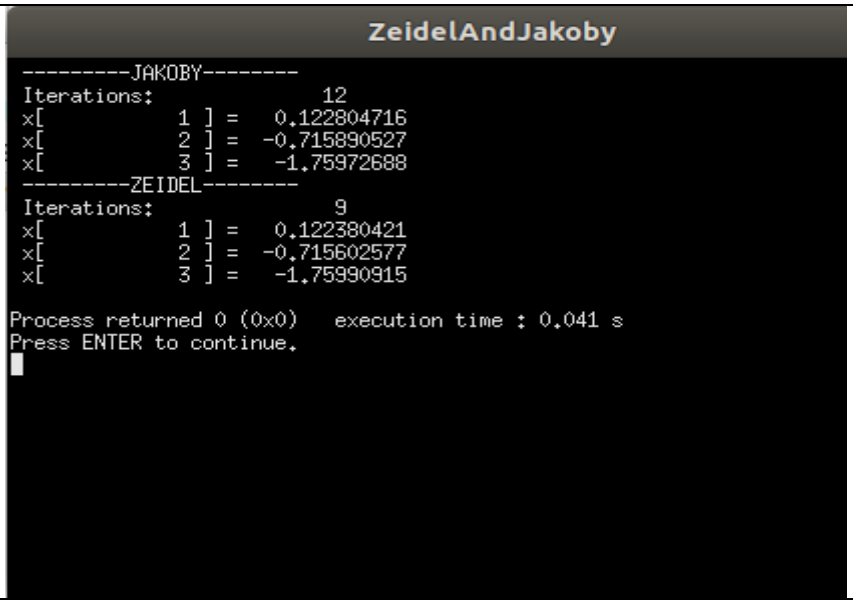
Результат программы на языке C++



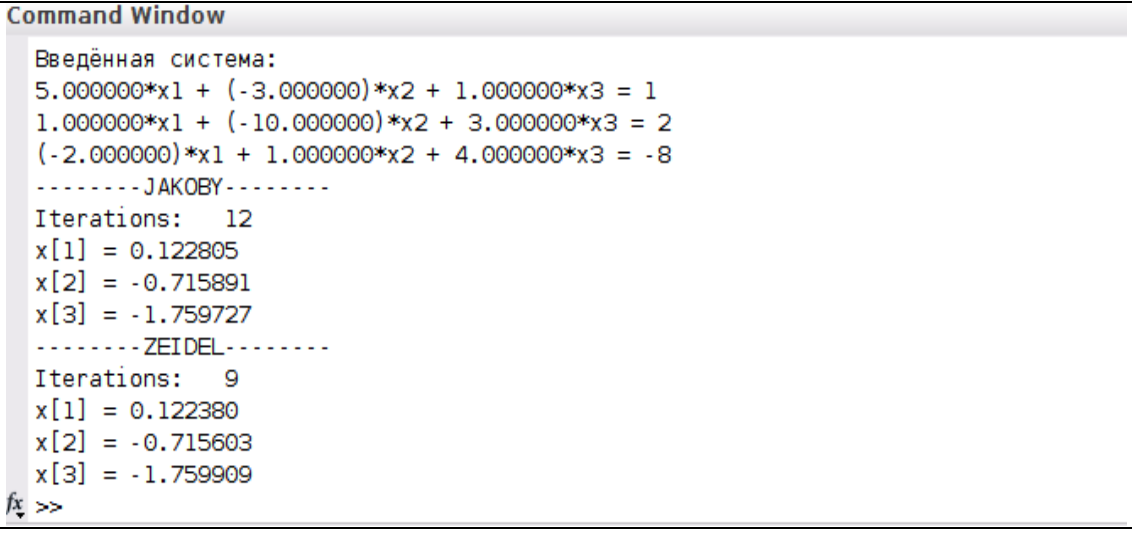

Результат программы на языке Python



Результат программы на языке Fortran




Результат программы на языке Matlab



| | C++ | Python | Fortran | Matlab |
|---|---|---|---|---|
| Метод Якоби | 0.122805 | 0.12280471275059368 | 0.122804716 | 0.122805 |
| | -0.715891 | -0.7158905160528284 | -0.715890527 | -0.715891 |
| | -1.75973 | -1.759726885328125 | -1.75972688 | -1.759727 |
| Метод Гаусса-Зейделя | 0.12238 | 0.12238040635646169 | 0.122380421 | 0.12238 |
| | -0.715603 | -0.7156025531675849 | -0.715602577 | -0.715603 |
| | -1.75991 | -1.7599091585298732 | -1.75990915 | -1.759909 |

## 9. Вывод

В ходе выполнения лабораторной работы №3 были изучены и программно реализованы методы Якоби и Гаусса-Зейделя решения системы линейных уравнений на языках высокого уровня (C++, Python, Fortran, Matlab). Так же было установленно, что меньше всего итераций понадобилось методу Гаусса - Зейделя.