МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА



Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Лабораторная работа №7

«Определение собственных векторов матрицы методом

Крылова»

по дисциплине

Вычислительная математика

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Суркова А.С.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Сухоруков В.А.

19-ИВТ-3

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2021

Оглавление

[Цель работы 3](#_Toc72147215)

[Постановка задачи 4](#_Toc72147216)

[Теоретические сведения 5](#_Toc72147217)

[Расчетные данные 8](#_Toc72147218)

[Код программы 8](#_Toc72147219)

[Equation.h 8](#_Toc72147220)

[system\_of\_equations.h 11](#_Toc72147221)

[Matrix.h 15](#_Toc72147222)

[Main.cpp 16](#_Toc72147223)

[Результаты работы программы 20](#_Toc72147224)

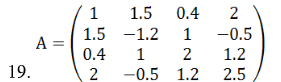
[Вывод 21](#_Toc72147225)

# Цель работы

Закрепление знаний и умений определения собственных числе и векторов матрицы методом Крылова.

# Постановка задачи

Используя метод Крылова, найти собственные числа и собственные векторы матрицы. Собственные числа определить с четырьмя верными цифрами, а собственные векторы – с тремя десятичными знаками.



# Теоретические сведения

Пусть

- характеристический полином (с точность до знака) матрицы A. Согласно тождеству Гамильтона-Кели, матрица А обращает в нуль свой характеристический полином; поэтому

Возьмем теперь произвольный ненулевой вектор

Умножая обе части равенства справа на , получим:

Положим:

Тогда равенство приобретает вид:

или

где

Следовательно, векторное равенство эквивалентно системе уравнений:

из которой, вообще говоря, можно определить неизвестный коэффициенты

Так как на основании формулы

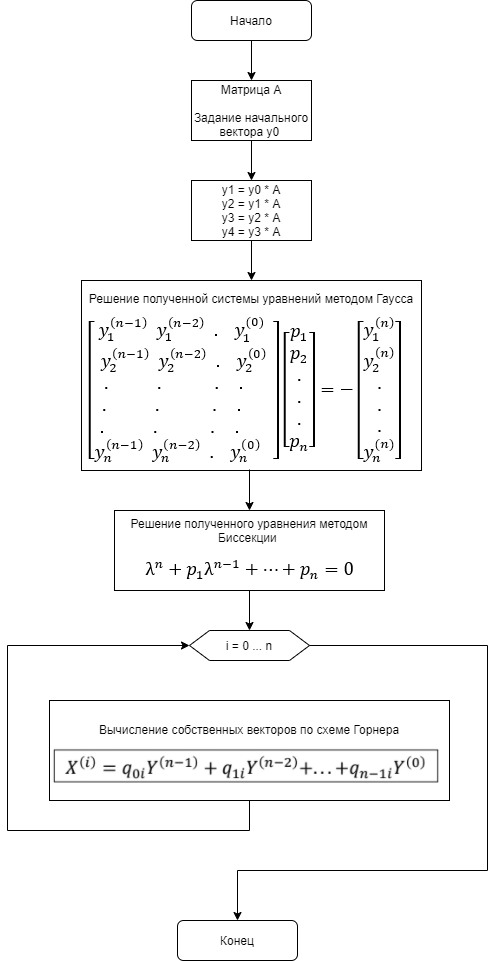
то координаты вектора последовательно вычисляются по формулам:

таким образом, определение коэффициентов характеристического полинома методом А.Н Крылова сводится к решению линейной системы уравнений, коэффициенты которой вычисляются по формулам, причем координаты начального вектора

произвольны. Если система имеет единственное решение, то ее корни являются коэффициентами характеристического полинома. Это решение может быть найдено, например методом Гаусса. Если система не имеет единственного решения, то задача усложняется. В этом случае рекомендуется изменить начальный вектор.

**Определение собственных векторов:**

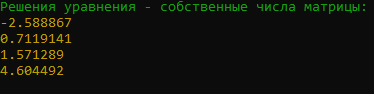
Таким образом, если , то полученная линейная комбинация векторов дает собственный вектор с точностью до числового множителя. Коэффициенты могут быть легко определены по схеме *Горнера*



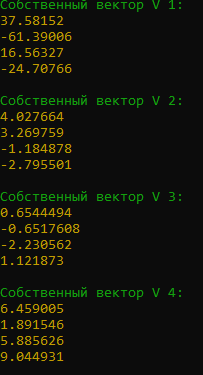
# Расчетные данные

**Значения, полученные в ходе лабораторной работы:**

Собственные числа



Собственные вектора



# Код программы

## Equation.h

#pragma once

#include<vector>

#include <algorithm>

#include<iostream>

using namespace std;

double const E = 0.001;

//Класс уравнений четвертой степени.

struct equation {

vector<double>coefficients;

//Метод для задания коэфициентов

void setCoefficients(vector<double> coeff){

for (int i = 0; i < coeff.size(); i++){

this->coefficients.push\_back(coeff[i]);

}

}

//Метод для получение значения функции в заданной точке

double getValueAtX(double x){

double res;

res=pow(x, 4) + coefficients[0] \* pow(x, 3) +

coefficients[1] \* pow(x, 2) +coefficients[2] \* x +

coefficients[3];

return res;

}

//Метод для получения интервалов смены знаков функции.

vector<vector<double>> getIntervals(){

vector<double> x\_vector;

vector<double> y\_vector;

//Задаем интервал иксов от -100 до 100

for (double i = -100.0; i <= 100; i++){

x\_vector.push\_back(i);

}

//Получаем значения функии в каждой точке интервала

//от -100 до 100

for (int i=0; i < x\_vector.size(); i++){

y\_vector.push\_back(getValueAtX(x\_vector[i]));

}

//В векторзначения при которых функция меняет знак

vector<double> new\_vector;

for (int i = 0; i < y\_vector.size() - 1; i++){

if (y\_vector[i] \* y\_vector[i + 1] < 0){

new\_vector.push\_back(x\_vector[i]);

new\_vector.push\_back(x\_vector[i+1]);

}

}

//Сортировка по возрастанию

sort(new\_vector.begin(), new\_vector.end());

//Записываем по парам полученные значения

vector<vector<double>> cord\_vector;

for (int i = 0; i < new\_vector.size(); i = i + 2) {

vector<double> temp;

temp.push\_back(new\_vector[i]);

temp.push\_back(new\_vector[i + 1]);

cord\_vector.push\_back(temp);

}

return cord\_vector;

}

//Метод для получения решений методом бисекций.

vector<double> Bisection() {

vector<double> res;

//Значение, для хранения X(i-1) - ответ, полученный на

//пред. итерации

double prev\_value ;

//Переменная для хранения значения Xi

double xI;

//Получение списка интервалов монотонности,

//которые содержат решения

vector<vector<double>> intervals =

this->getIntervals();

//Проход по каждому интервалу [a;b]

double a, b;

for (int i = 0; i < intervals.size(); i++){

prev\_value = -DBL\_MAX;

xI = DBL\_MAX;

a = intervals[i][0];

b = intervals[i][1];

//Пока разница текущего и предыдущего значений

//больше установленной погрешность E=0.001,

//вычисляем новое значение

while (abs(prev\_value - xI) > E) {

prev\_value = xI;

xI = (a + b) / 2.0;

//Если значение функии при Xi и при X = a

//имеет разные знаки, то меняем b из

//промежутка[a;b] на Xi иначе меняем a на Xi

if (this->getValueAtX(xI) \*

this->getValueAtX(a) < 0){

b = xI;

}

else{

a = xI;

}

}

//Записываем полученный ответ в результирующий

// вектор

res.push\_back(xI);

}

return res;

}

};

## system\_of\_equations.h

#pragma once

#include<vector>

#include<iomanip>

#include "Colors.h"

using namespace std;

//Класс системы линейных уравнений

class system\_of\_equations{

public:

//Коэффициенты перед х

vector<vector<double>> coefficients;

//Столбец свободных членов

vector<double> free;

//Количество уравнений/переменных

int n;

//Конструктор с параметрами

system\_of\_equations(vector<vector<double>> a, vector<double> b) {

this->n = b.size();

vector<double> tmp(this->n,0);

for (int i = 0; i < this->n; i++) {

this->coefficients.push\_back(tmp);

this->free.push\_back(0);

}

for (int i = 0; i < this->n; i++){

for (int j = 0; j < this->n; j++){

this->coefficients[i][j] = a[i][j];

}

this->free[i] = b[i];

}

}

/\*Метод Гаусса

Принцип работы:

1)Поиск масимального коэффициента

2)Перестановка первой строки и строки с максимальным

коэффициентом в матрице A

3)Перестановка первой строки и строки с максимальным

коэффициентом в векторе b

4)Перестановка первого столбца и столбца с максимальным

коэффициентом в матрице A

6)Приведение матрицы к треугольному виду

6)Обратный ход метода Гаусса для поиска корней системы

\*/

vector<double> Gauss() {

cout<<Green<<"\nСоставленная система уравнений:\n"

<<setprecision(4)<<Yellow<< setw(6)

<< coefficients[0][0] << " \* p1 + "

<< setw(6) << coefficients[0][1] << " \* p2 + "

<< setw(6) << coefficients[0][2] << " \* p3 + "

<< setw(6) << coefficients[0][3] << " \* p4 ="

<< free[0] << "\n"<< setw(6) << coefficients[1][0]

<< " \* p1 + " << setw(6) << coefficients[1][1]

<< " \* p2 + " << setw(6)<< coefficients[1][2]

<< " \* p3 + " << setw(6)<< coefficients[1][3]

<< " \* p4 =" << free[1]<< "\n"<< setw(6)

<< coefficients[2][0]

<< " \* p1 + " << setw(6) << coefficients[2][1]

<< " \* p2 + " << setw(6) << coefficients[2][2]

<< " \* p3 + " << setw(6) << coefficients[2][3]

<< " \* p4 =" << free[2] << "\n"

<< setw(6) << coefficients[3][0] << " \* p1 + "

<< setw(6) << coefficients[3][1] << " \* p2 + "

<< setw(6) << coefficients[3][2] << " \* p3 + " << setw(6) << coefficients[3][3] << " \* p4 =" << free[3] << "\n";

vector<double>x(4);

double a\_max = 0;

int i\_max = 0, j\_max = 0;

//Поиск максимального коэффициента в матрице A

for (int i = 0; i < 4; i++) {

for (int j = 0; j < 4; j++) {

if (coefficients[i][j] > a\_max) {

a\_max = coefficients[i][j];

i\_max = i;

j\_max = j;

}

}

}

//Перестановка первой строки и строки с максимальным коэффициентом в матрице a

double tmp = 0;

for (int j = 0; j < 4; j++) {

tmp = coefficients[0][j];

coefficients[0][j] = coefficients[i\_max][j];

coefficients[i\_max][j] = tmp;

}

//Перестановка первой строки и строки с максимальным коэффициентом в векторе b

tmp = free[0];

free[0] = free[i\_max];

free[i\_max] = tmp;

//Перестановка первого столбца и столбца с максимальным коэффициентом в матрице a

for (int i = 0; i < 4; i++) {

double tmp = 0;

tmp = coefficients[i][0];

coefficients[i][0] = coefficients[i][j\_max];

coefficients[i][j\_max] = tmp;

}

//Приведение матрицы к треугольному виду

tmp = coefficients[0][0];

coefficients[0][0] = 1;

coefficients[0][1] = coefficients[0][1] / tmp;

coefficients[0][2] = coefficients[0][2] / tmp;

coefficients[0][3] = coefficients[0][3] / tmp;

free[0] = free[0] / tmp;

tmp = coefficients[1][0];

coefficients[1][0] =

coefficients[1][0] - coefficients[0][0] \* tmp;

coefficients[1][1] =

coefficients[1][1] - coefficients[0][1] \* tmp;

coefficients[1][2] =

coefficients[1][2] - coefficients[0][2] \* tmp;

coefficients[1][3] =

coefficients[1][3] - coefficients[0][3] \* tmp;

free[1] = free[1] - free[0] \* tmp;

tmp = coefficients[2][0];

coefficients[2][0] =

coefficients[2][0] - coefficients[0][0] \* tmp;

coefficients[2][1] =

coefficients[2][1] - coefficients[0][1] \* tmp;

coefficients[2][2] =

coefficients[2][2] - coefficients[0][2] \* tmp;

coefficients[2][3] =

coefficients[2][3] - coefficients[0][3] \* tmp;

free[2] = free[2] - free[0] \* tmp;

tmp = coefficients[3][0];

coefficients[3][0] =

coefficients[3][0] - coefficients[0][0] \* tmp;

coefficients[3][1] =

coefficients[3][1] - coefficients[0][1] \* tmp;

coefficients[3][2] =

coefficients[3][2] - coefficients[0][2] \* tmp;

coefficients[3][3] =

coefficients[3][3] - coefficients[0][3] \* tmp;

free[3] = free[3] - free[0] \* tmp;

tmp = coefficients[1][1];

coefficients[1][1] = 1;

coefficients[1][2] = coefficients[1][2] / tmp;

coefficients[1][3] = coefficients[1][3] / tmp;

free[1] = free[1] / tmp;

tmp = coefficients[2][1];

coefficients[2][0] =

coefficients[2][0] - coefficients[1][0] \* tmp;

coefficients[2][1] =

coefficients[2][1] - coefficients[1][1] \* tmp;

coefficients[2][2] =

coefficients[2][2] - coefficients[1][2] \* tmp;

coefficients[2][3] =

coefficients[2][3] - coefficients[1][3] \* tmp;

free[2] = free[2] - free[1] \* tmp;

tmp = coefficients[3][1];

coefficients[3][0] =

coefficients[3][0] - coefficients[1][0] \* tmp;

coefficients[3][1] =

coefficients[3][1] - coefficients[1][1] \* tmp;

coefficients[3][2] =

coefficients[3][2] - coefficients[1][2] \* tmp;

coefficients[3][3] =

coefficients[3][3] - coefficients[1][3] \* tmp;

free[3] = free[3] - free[1] \* tmp;

tmp = coefficients[2][2];

coefficients[2][2] = 1;

coefficients[2][3] = coefficients[2][3] / tmp;

free[2] = free[2] / tmp;

tmp = coefficients[3][2];

coefficients[3][0] =

coefficients[3][0] - coefficients[2][0] \* tmp;

coefficients[3][1] =

coefficients[3][1] - coefficients[2][1] \* tmp;

coefficients[3][2] =

coefficients[3][2] - coefficients[2][2] \* tmp;

coefficients[3][3] =

coefficients[3][3] - coefficients[2][3] \* tmp;

free[3] = free[3] - free[2] \* tmp;

//Обратный ход метода Гаусса для поиска корней системы

x[3] = free[3] / coefficients[3][3];

x[2] = (free[2] - coefficients[2][3] \* x[3]) / coefficients[2][2];

x[1] = (free[1] - coefficients[1][2] \* x[2] - coefficients[1][3] \* x[3]) / coefficients[1][1];

x[0] = (free[0] - coefficients[0][1] \* x[1] - coefficients[0][2] \* x[2] - coefficients[0][3] \* x[3]) / coefficients[0][0];

return x;

}

};

## Matrix.h

#pragma once

#include<vector>

using namespace std;

//Класс квадратной матрицы

struct matrix{

int n; //Размерность матрицы

vector<vector<double>> a; //Коэффициенты матрицы

//Конструктор с параметрами

matrix(vector<vector<double>> A, int N) {

vector<double> tmp(N,0);

for (int i = 0; i < N; i++){

this->a.push\_back(tmp);

}

for (int i = 0; i < N; i++){

for (int j = 0; j < N; j++){

this->a[i][j] = A[i][j];

}

}

this->n = N;

}

//Метод умножения матрицы на вектор

vector<double>multiplyByVector(vector<double> vec) {

vector<double>res(vec.size());

for (int i = 0; i < this->n; i++){

for (int j = 0; j < this->n; j++){

res[i] += this->a[i][j] \* vec[j];

}

}

return res;

}

};

## Main.cpp

#include <iostream>

#include <vector>

#include "Matrix.h"

#include "system\_of\_equations.h"

#include "Colors.h"

#include "Print.h"

#include "equation.h"

using namespace std;

int main() {

vector<vector<double>>c;

vector<double>y0(4),y1(4), y2(4), y3(4), y4(4);

setlocale(LC\_ALL, "Russian"); //Включение русского языка в консоли

cout<<Green << "Определение собственных чисел и собственных векторов матрицы методом Крылова\n\n" << Reset;

//создаем объект матрица, передаем туда исходные данные

vector<double> tmp(4, 0);

for (int i = 0; i < 4; i++){c.push\_back(tmp);}

c[0][0] = 1; c[0][1] = 1.5; c[0][2] = 0.4; c[0][3] = 2;

c[1][0] = 1.5; c[1][1] = -1.2; c[1][2] = 1; c[1][3] = -0.5;

c[2][0] = 0.4; c[2][1] = 1; c[2][2] = 2; c[2][3] = 1.2;

c[3][0] = 2; c[3][1] = -0.5; c[3][2] = 1.2; c[3][3] = 2.5;

matrix mat(c, 4);

cout<<Green << "Исходная матрица:\n";

print\_matrix(mat);

//задаем вектор y0

y0[0] = 0; y0[1] = 1;

y0[2] = 0; y0[3] = 0;

cout << Green << "\nВектор y0:\n";

print\_vector(y0);

//вычсиление вектора y1 и его вывод

y1 = mat.multiplyByVector(y0);

cout << Green << "Вектор y1:\n";

print\_vector(y1);

//вычсиление вектора y2 и его вывод

y2 = mat.multiplyByVector(y1);

cout << Green << "Вектор y2:\n";

print\_vector(y2);

//вычсиление вектора y3 и его вывод

y3 = mat.multiplyByVector(y2);

cout << Green << "Вектор y3:\n";

print\_vector(y3);

//вычсиление вектора y4 и его вывод

y4 = mat.multiplyByVector(y3);

cout << Green << "Вектор y4:\n";

print\_vector(y4);

//Создание объекта - система уравнений, на основе полученных векторов

vector<vector<double>> temporary(4);

temporary[0].push\_back(y3[0]); temporary[0].push\_back(y2[0]);

temporary[0].push\_back(y1[0]); temporary[0].push\_back(y0[0]);

temporary[1].push\_back(y3[1]); temporary[1].push\_back(y2[1]);

temporary[1].push\_back(y1[1]); temporary[1].push\_back(y0[1]);

temporary[2].push\_back(y3[2]); temporary[2].push\_back(y2[2]);

temporary[2].push\_back(y1[2]); temporary[2].push\_back(y0[2]);

temporary[3].push\_back(y3[3]); temporary[3].push\_back(y2[3]);

temporary[3].push\_back(y1[3]); temporary[3].push\_back(y0[3]);

vector<double> y4\_invert(4);

y4\_invert[0] = y4[0] \* (-1); y4\_invert[1] = y4[1] \* (-1);

y4\_invert[2] = y4[2] \* (-1); y4\_invert[3] = y4[3] \* (-1);

system\_of\_equations system(temporary, y4\_invert);

//Решение системы методом Гаусса

vector<double>p = system.Gauss();

cout << Green << "\nРешение системы уравнений методом Гаусса\n";

print\_vector(p);

//Создаем объект уравнение на основе полученных решений системы

equation polinom;

polinom.setCoefficients(p);

cout << Green << "Полученное уравнение P(z):\n";

print\_equation(polinom);

//Получение собственных чисел - решений полученного уравнения

vector<double> res;

res=polinom.Bisection();

cout << Green << "\nРешения уравнения - собственные числа матрицы:\n";

print\_vector(res);

//На основе полученных собственных чисел

//векторов p и векторов y0-y4 через схему Горнера

//находим собственные вектора

cout<<Green<<"\nНахождение собсвтенных векторов:\n";

vector < vector <double> > ownVectors(4, vector <double>(4));

for (int k = 0; k < 4; k++){

vector<double>q(5);

q[0] = 1.0;

vector<double>y0\_tmp, y1\_tmp, y2\_tmp, y3\_tmp;

for (int i = 0; i < y0.size(); i++){

y0\_tmp.push\_back(y0[i]);

y1\_tmp.push\_back(y1[i]);

y2\_tmp.push\_back(y2[i]);

y3\_tmp.push\_back(y3[i]);

}

//вычисление значения qi

for (int j = 1, i = 0; i < p.size(); i++, j++){

q[j] = res[k] \* q[j - 1] + p[i];

}

//умножаем вектор Y3 на q0

for (int j = 0; j < y3.size(); j++){

y3\_tmp[j] \*= q[0];

}

//умножаем вектор Y2 на q1

for (int j = 0; j < y2.size(); j++) {

y2\_tmp[j] \*= q[1];

}

//умножаем вектор Y1 на q2

for (int j = 0; j < y1.size(); j++) {

y1\_tmp[j] \*= q[2];

}

//умножаем вектор Y0 на q3

for (int j = 0; j < y0.size(); j++) {

y0\_tmp[j] \*= q[3];

}

//Получаем собтсвенный вектор путем сложения

//произвдеений qi на вектор Yj

ownVectors[k][0] = y0\_tmp[0] + y1\_tmp[0] + y2\_tmp[0] + y3\_tmp[0];

ownVectors[k][1] = y0\_tmp[1] + y1\_tmp[1] + y2\_tmp[1] + y3\_tmp[1];

ownVectors[k][2] = y0\_tmp[2] + y1\_tmp[2] + y2\_tmp[2] + y3\_tmp[2];

ownVectors[k][3] = y0\_tmp[3] + y1\_tmp[3] + y2\_tmp[3] + y3\_tmp[3];

cout<<Green<<"Собственный вектор V " << (k + 1) << ":\n";

print\_vector(ownVectors[k]);

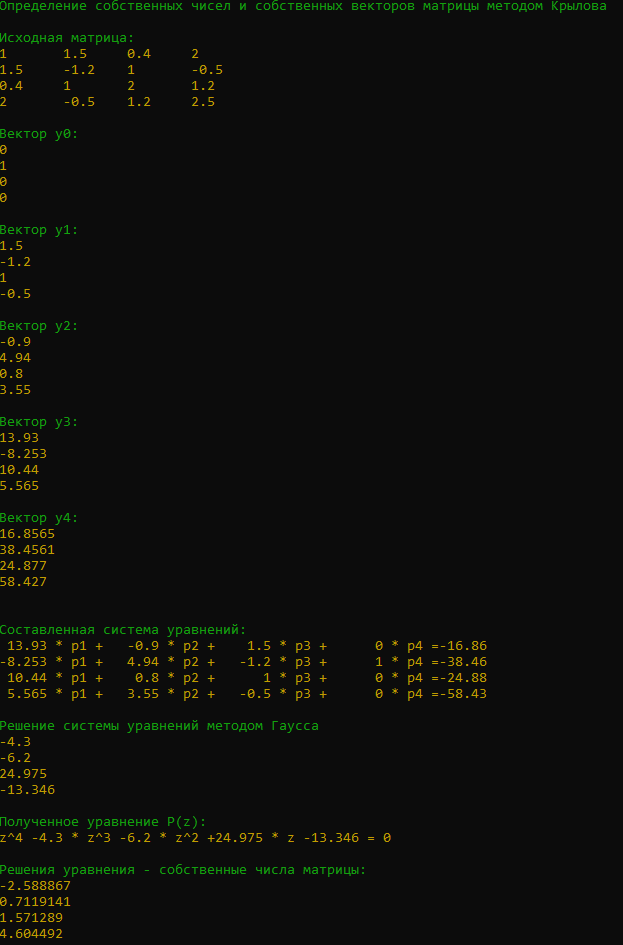
}

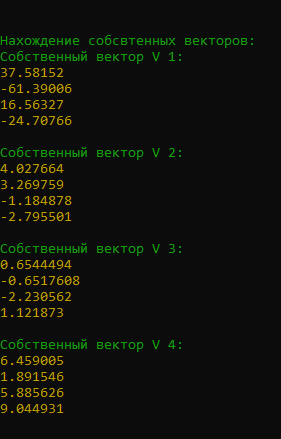
cout << Reset;

return 0;

}

# Результаты работы программы





# Вывод

В ходе данной работы были закреплены знания и умения по нахождение собственных чисел и собственных векторов методом Крылова.