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

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.

19. $$A = \begin{pmatrix} 1 & 1.5 & 0.4 & 2 \\ 1.5 & -1.2 & 1 & -0.5 \\ 0.4 & 1 & 2 & 1.2 \\ 2 & -0.5 & 1.2 & 2.5 \end{pmatrix}$$

Теоретические сведения

Пусть

$$D(\lambda) \equiv \det(\lambda \mathrm{E} - \mathrm{A}) = \lambda^n + p_1\lambda^{n-1} + \cdots + p_n$$

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

$$A^n + p_1A^{n-1} + \cdots + p_nE = 0.$$

Возьмем теперь произвольный ненулевой вектор

$$y^0 = \begin{bmatrix} y_1^{(0)} \\ . \\ . \\ . \\ y_n^{(0)} \end{bmatrix}$$

Умножая обе части равенства справа на $y_n^{(0)}$, получим:

$$A^n + p_1A^{n-1}y^{(0)} + \cdots + p_nE = 0$$

Положим:

$$A^k y^{(0)} = y^{(k)} \qquad (k = 1, 2, \dots, n)$$

Тогда равенство приобретает вид:

$$y^{(n)} + p_1y^{(n-1)} + \cdots + p_ny^{(0)} = 0$$

или

$$\begin{bmatrix} y_1^{(n-1)} & y_1^{(n-2)} & . & y_1^{(0)} \\ y_2^{(n-1)} & y_2^{(n-2)} & . & y_2^{(0)} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ y_n^{(n-1)} & y_n^{(n-2)} & . & y_n^{(0)} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ . \\ . \\ . \\ p_n \end{bmatrix} = -\begin{bmatrix} y_1^{(n)} \\ y_2^{(n)} \\ . \\ . \\ . \\ y_n^{(n)} \end{bmatrix}$$

где

$$y^{(k)} = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ . \\ . \\ . \\ y_n^{(k)} \end{bmatrix} \qquad (k = 0, 1, 2, \dots, n)$$

Следовательно, векторное равенство эквивалентно системе уравнений:

$$p_1y_i^{(n-1)} + p_2y_i^{(n-2)} + \cdots + p_ny_i^{(0)} = -p_1y_i^{(n-1)} \qquad (i = 1, 2, \dots, n)$$

из которой, вообще говоря, можно определить неизвестный коэффициенты $p_1, p_2, \dots, p_n$.

Так как на основании формулы

$$y^{(k)} = Ay^{(k-1)} \qquad (k = 1, 2, \dots, n)$$

то координаты $y_1^{(k)}, y_2^{(k)}, \dots, y_n^{(k)}$ вектора $y^{(k)}$ последовательно вычисляются по формулам:

$$\begin{cases} y_i^{(1)} = \sum_{j=1}^{n} a_{ij} y_i^{(0)} \\ \dots \\ y_i^{(n)} = \sum_{j=1}^{n} a_{ij} y_i^{(n-1)} \end{cases}$$

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

$$y^{(0)} = \begin{bmatrix} y_1^{(0)} \\ y_2^{(0)} \\ \cdot \\ \cdot \\ \cdot \\ y_n^{(0)} \end{bmatrix}$$

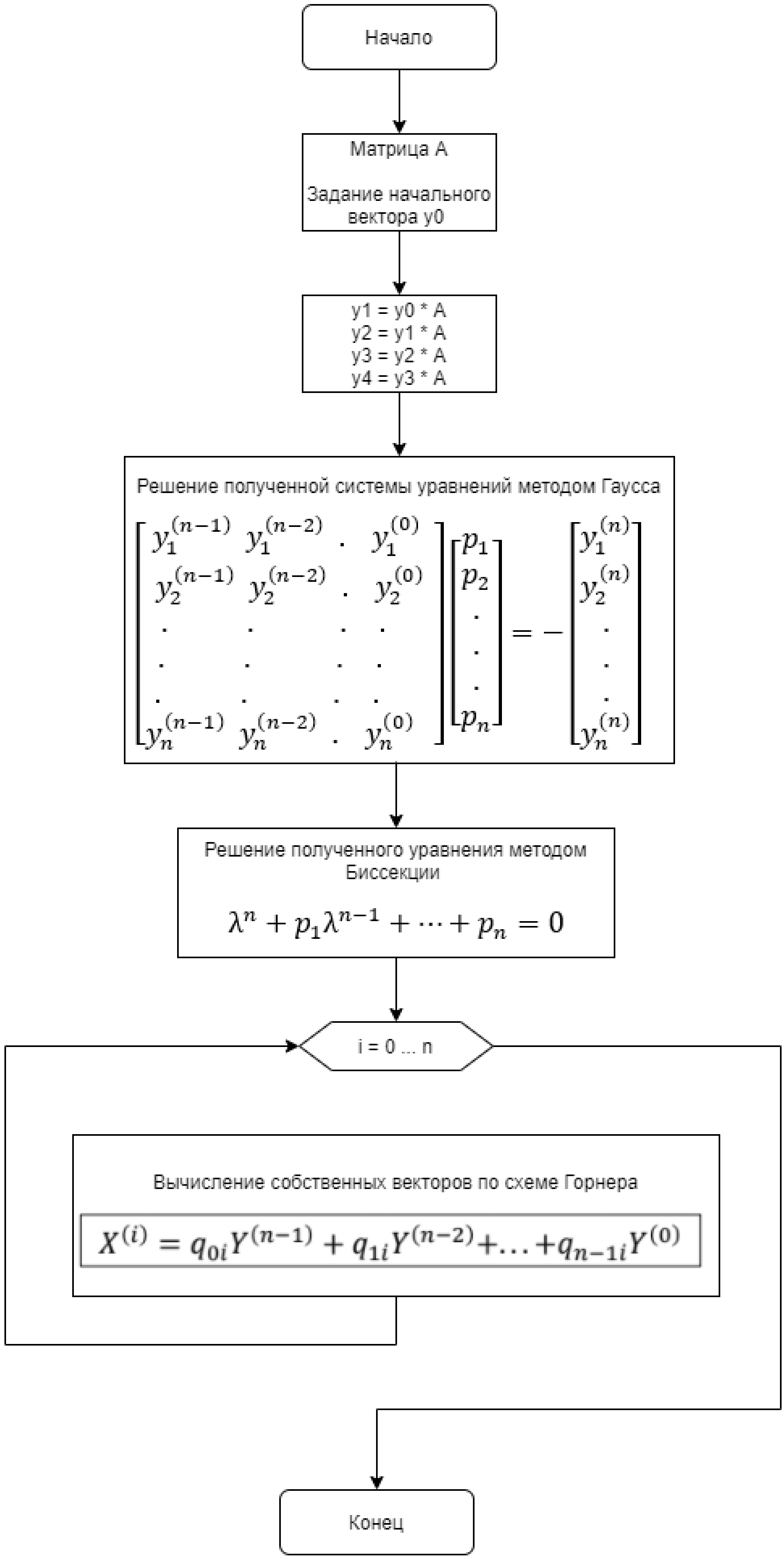
произвольны. Если система имеет единственное решение, то ее корни $p_1, p_2, \dots, p_n$ являются коэффициентами характеристического полинома. Это решение может быть найдено, например методом Гаусса. Если система не имеет единственного решения, то задача усложняется. В этом случае рекомендуется изменить начальный вектор.

Определение собственных векторов:

$$c_i \varphi_i(\lambda_i) x^{(i)} = y^{(n-1)} + q_{1,i} y^{(n-2)} + \dots + q_{n-1,i} y^{(0)} \quad (i = 1, 2, \dots, n)$$

Таким образом, если $c_i \neq 0$, то полученная линейная комбинация векторов $y^{(n-1)}, y^{(n-2)}, \dots, y^{(0)}$ дает собственный вектор $x^{(i)}$ с точностью до числового множителя. Коэффициенты $q_{j,\ i}$ $(j = 1,2, \dots, n-1)$ могут быть легко определены по схеме *Горнера*

$$\begin{cases} q_{0i} = 1 \\ q_{ji} = \lambda_i q_{j-1,i} + p_j \end{cases}$$


Начало
Матрица А
Задание начального вектора y0
y1 = y0 * A
y2 = y1 * A
y3 = y2 * A
y4 = y3 * A
Решение полученной системы уравнений методом Гаусса
$\begin{bmatrix} y_1^{(n-1)} & y_1^{(n-2)} & . & y_1^{(0)} \\ y_2^{(n-1)} & y_2^{(n-2)} & . & y_2^{(0)} \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ y_n^{(n-1)} & y_n^{(n-2)} & . & y_n^{(0)} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ . \\ . \\ . \\ p_n \end{bmatrix} = - \begin{bmatrix} y_1^{(n)} \\ y_2^{(n)} \\ . \\ . \\ . \\ y_n^{(n)} \end{bmatrix}$
Решение полученного уравнения методом Биссекции
$\lambda^n + p_1 \lambda^{n-1} + \cdots + p_n = 0$
i = 0 ... n
Вычисление собственных векторов по схеме Горнера
$X^{(i)} = q_{0i} Y^{(n-1)} + q_{1i} Y^{(n-2)} + \ldots + q_{n-1i} Y^{(0)}$
Конец

Расчетные данные

Значения, полученные в ходе лабораторной работы:

Собственные числа

```
Решения уравнения - собственные числа матрицы:
-2.588867
0.7119141
1.571289
4.604492
```

Собственные вектора

```
Собственный вектор V 1:
37.58152
-61.39006
16.56327
-24.70766

Собственный вектор V 2:
4.027664
3.269759
-1.184878
-2.795501

Собственный вектор V 3:
0.6544494
-0.6517608
-2.230562
1.121873

Собственный вектор V 4:
6.459005
1.891546
5.885626
9.044931
```

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 функции 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 А
		3)Перестановка первой строки и строки с максимальным
		  коэффициентом в векторе b
		4)Перестановка первого столбца и столбца с максимальным
		  коэффициентом в матрице А
		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 а
	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 а
        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

```
Определение собственных чисел и собственных векторов матрицы методом Крылова

Исходная матрица:
1       1.5     0.4     2
1.5     -1.2    1       -0.5
0.4     1       2       1.2
2       -0.5    1.2     2.5

Вектор y0:
0
1
0
0

Вектор y1:
1.5
-1.2
1
-0.5

Вектор y2:
-0.9
4.94
0.8
3.55

Вектор y3:
13.93
-8.253
10.44
5.565

Вектор y4:
16.8565
38.4561
24.877
58.427


Составленная система уравнений:
 13.93 * p1 +   -0.9 * p2 +    1.5 * p3 +      0 * p4 =-16.86
-8.253 * p1 +   4.94 * p2 +   -1.2 * p3 +      1 * p4 =-38.46
 10.44 * p1 +    0.8 * p2 +      1 * p3 +      0 * p4 =-24.88
 5.565 * p1 +   3.55 * p2 +   -0.5 * p3 +      0 * p4 =-58.43

Решение системы уравнений методом Гаусса
-4.3
-6.2
24.975
-13.346

Полученное уравнение P(z):
z^4 -4.3 * z^3 -6.2 * z^2 +24.975 * z -13.346 = 0

Решения уравнения - собственные числа матрицы:
-2.588867
0.7119141
1.571289
4.604492
```

```
Нахождение собсвтенных векторов:
Собственный вектор V 1:
37.58152
-61.39006
16.56327
-24.70766

Собственный вектор V 2:
4.027664
3.269759
-1.184878
-2.795501

Собственный вектор V 3:
0.6544494
-0.6517608
-2.230562
1.121873

Собственный вектор V 4:
6.459005
1.891546
5.885626
9.044931
```

Вывод

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