**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»**

**Журнал по практике**

Студент: Кузнецов Илья Игоревич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

##### Институт №3 “Системы управления, информатика и электроэнергетика”

###### Кафедра: №304 “Вычислительные машины, системы и сети”

##### Учебная группа: M30-109Б

Направление подготовки (специальность): 09.03.01 “Информатика и вычислительная техника”

*(шифр) (название направления, специальности)*

Вид практики: учебная

*(учебная, производственная, преддипломная или другой вид практики)*

Руководитель практики от МАИ

Секретарёв Виталий Евгеньевич/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

*(фамилия, имя, отчество) (подпись) (дата)*

Студент

Кузнецов Илья Игоревич /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

*(фамилия, имя, отчество) (подпись) (дата)*

Москва 2020 г.

**1.Место и сроки проведения практики**

*Наименование предприятия:* Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский авиационный институт (национальный исследовательский университет)»

*Название структурного подразделения:* Кафедра 304

*Сроки проведения практики:*

*-дата начала практики: 09.02.20*

*-дата окончания практики: 30.06.20*

1. **Инструктаж по технике безопасности**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

*(подпись проводившего)* *(дата проведения)*

1. **Индивидуальное задание студенту**

Разработать программы на языке С++ для решения задач с интерфейсом для выбора варианта ввода исходных данных и вывода результатов:

Задача 1.

На отрезке [0;2] методом Ньютона найти корень уравнения –x3-2x2-4x+10 = 0 с заданной точностью.

Задача 2.

1)Решите систему линейных уравнений методом *“Простой итерации”* с точностью 0.001, предварительно оценив число достаточных для этого итераций:

Где

2) Полученное решение используйте для вычисления невязки каждого уравнения.

3) Все полученные приближения решения системы привести в итоговом отчете.

4) Не забываем начинать отчет с формулировки задания.

В отчете для каждого задания необходимо привести:

1. постановку задачи.
2. описание используемого математического метода,
3. входные данные,
4. выходные данные,
5. словесное описание алгоритма решения задачи,
6. структурную схему алгоритма,
7. описание интерфейса программы,
8. текст программы с комментариями,
9. результаты проверки функционирования программы,
10. выводы.

Формируемые в ходе выполнения задания компетенции:

|  |  |  |
| --- | --- | --- |
| № пп | Шифр | Компетенция |
| 1 | ПСК-10 | Способен и умеет разрабатывать алгоритмическое и специальное ПО ВТ |
| 2 | ПСК-13 | Способность владеть языками процедурного и объектно-ориентированного программирования, навыками разработки программ. |

Достигаемые в ходе выполнения задания результаты освоения:

|  |  |  |
| --- | --- | --- |
| № пп | Шифр | Результат освоения |
| 1 | З-10-1  (ПСК-10) | Знать методы разработки алгоритмического и специального программного обеспечения вычислительной техники |
| 2 | В-10-1(ПСК-10) | Владеть методами разработки алгоритмического и специального программного обеспечения вычислительной техники |
| 3 | У-10-1(ПСК-10) | Уметь разрабатывать алгоритмическое и специальное программное обеспечение вычислительной техники |
| 4 | В-13-1 (ПСК-13) | Владеть языками процедурного и объектно-ориентированного программирования, навыками разработки и отладки программ не менее, чем на одном из алгоритмических языков |
| 5 | У-13-1 (ПСК-13) | Уметь работать с современными системами программирования, включая объектно-ориентированные |
| 6 | У-13-2 (ПСК-13) | Уметь ставить задачу и разрабатывать алгоритм ее решения, использовать прикладные системы программирования, разрабатывать основные программные документы |

**4.План выполнения индивидуального задания**

**План самостоятельной работы студента**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дата | Содержание или наименование проделанной работы | Продолжительность, часы | Компетенция | Подпись руководителя практики |
| 11.02 | Разработка структурной схемы алгоритма для решения задачи 1. | 12 | ПСК-10 |  |
| 18.02 | Проверка структурной схемы алгоритма для решения задачи 1**руководителем практики**. | 3 | ПСК-10 |  |
| 20.02 | Разработка программы 1 для решения задачи 1. Ввод и вывод данных с клавиатуры в интерактивном режиме. | 14 | ПСК-13 |  |
| 25.02 | Проверка программы 1 **руководителем практики.** | 5 | ПСК-13 |  |
| 27.02 | Отладка и тестирование. | 8 | ПСК-13 |  |
| 04.03 | Проверка результата работы программы 1 **руководителем практики** | 4 | ПСК-10  ПСК-13 |  |
| 12.03 | Разработка структурной схемы алгоритма для решения задачи 2. | 12 | ПСК-10 |  |
| 18.03 | Проверка структурной схемы алгоритма для решения задачи 2 **руководителем практики**. | 3 | ПСК-10 |  |
| 26.03 | Разработка программы 2 для решения задачи 2. Ввод и вывод данных с клавиатуры в интерактивном режиме. | 14 | ПСК-13 |  |
| 01.04 | Проверка программы 2 **руководителем практики.** | 5 | ПСК-13 |  |
| 09.04 | Отладка и тестирование. | 8 | ПСК-13 |  |
| 15.04 | Проверка результата работы программы 2 **руководителем практики** | 5 | ПСК-10  ПСК-13 |  |
|  | Итого: | 93 |  |  |

Кузнецов И.И./\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020г.

(*подпись студента-практиканта) (*дата)

Секретарёв В.Е./\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020г.

(*подпись руководителя практики от МАИ) (*дата)

**План работ практиканта по месту прохождения практики**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Дата | Содержание или наименование проделанной работы | Подразделение | Продолжительность, часы | Компетенция | Подпись руководителяпрактики |
| 1.07 | Корректировка отчета по практике. Проверка отчета руководителем практики. | Кафедра 304 | 3 | ПСК-10  ПСК-13 |  |
| 2.07 | Подготовка презентации для выступления на защите. Проверка презентации руководителем практики. | Кафедра 304 | 4 | ПСК-10  ПСК-13 |  |
| 03.07 | Защита отчетов по практике. | Кафедра 304 | 4 | ПСК-10  ПСК-13 |  |
| 04.07 | Защита отчетов по практике. | Кафедра 304 | 4 | ПСК-10  ПСК-13 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  | **Итого:** 15 |  |  |

Кузнецов И.И./\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

(*подпись студента-практиканта) (дата составления)*

Секретарёв В.Е./\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2020 г.

(*подпись руководителя практики от МАИ) (дата составления)*

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»**

**Отчет о прохождении практики**

Студента Кузнецов Илья Игоревич

##### Институт №3 “Системы управления, информатика и электроэнергетика”

###### Кафедра: №304 “Вычислительные машины, системы и сети”

##### Учебная группа: M30-109Б

Направление подготовки (специальность): 09.03.01 “Информатика и вычислительная техника”

*(шифр)(название направления, специальности)*

Вид практики: учебная

*(учебная, производственная, преддипломная или другой вид практики)*

Руководитель практики от МАИ

Секретарёв Виталий Евгеньевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(фамилия, имя, отчество) (подпись)*

*Наименование предприятия:* Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский авиационный институт (национальный исследовательский университет)»

*Название структурного подразделения (отдел, лаборатория):* Кафедра 304

Студент

Кузнецов Илья Игоревич /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ “\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

*(фамилия, имя, отчество) (подпись) (дата)*

Москва 2020 г.

**Вариант №7.**

**Задание 1.**

**Оглавление**

* Постановка задачи
* Входные данные
* Выходные данные
* Структурная схема алгоритма
* Описание интерфейса программы
* Код программы
* Тесты
* Вывод

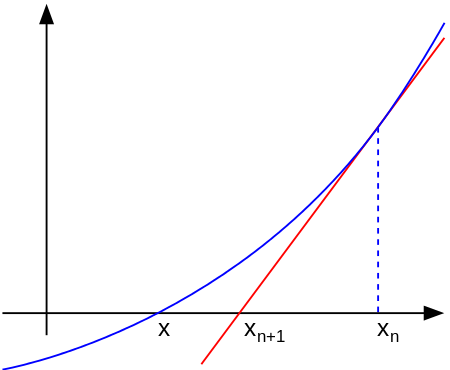
**Постановка задачи**

На отрезке [0;2] методом Ньютона найти корень уравнения –x3-2x2-4x+10 = 0 с заданной точностью.

**Описание математического метода.**

**Метод Ньютона.**

Основная идея **метода Ньютона** состоит следующем: задаётся начальное приближение вблизи предположительного корня, после чего строится касательная к графику исследуемой функции в точке приближения, для которой находится пересечение с осью абсцисс. Эта точка берётся в качестве следующего приближения. И так далее, пока не будет достигнута необходимая точность.



Как видно из графика, приближение **n+1** лучше предыдущего **n,** где синим цветом изображена функция **f(x),** ноль которой нужно найти, а красным – касательная очередного приближения.

**Достаточные условия сходимости метода**

Пусть функция f(x) имеет первую и вторую производную на отрезке **[a,b]**, причем выполнено условие знакопеременности функции **f(a)\*f(b)<0** , а производные **f ’(x)** и **f ’’(x)** сохраняют знак на отрезке **[a,b]**.

Тогда, исходя из начального приближения **x0∈[a,b]**, удовлетворяющего неравенству **f(x)\*f ’’(x)>0**, можно построить итерационную последовательность:

сходящуюся к единственному на **[a,b]** решению **ξ** уравнения **f(x)=0**.

Также не обязательно задавать отрезок **[a,b]**, содержащий корень уравнения, а достаточно лишь найти некоторое начальное приближение корня **x0 = x.**

Если же начальное приближение неизвестно, а известен лишь отрезок, тогда в качестве начально приближения можно выбрать точку **x0 = a,** для которого выполняется условие **f(x0) \* f ’’(x0) > 0,** иначе выбираем **x0 = b**.

Для окончания итерационного процесса может быть использовано условие:

где – требуемая точность.

Далее рассмотрим алгоритм решения.

**Алгоритм решения**

Входные данные: **f(x), f ’(x), f ’’(x), a, b, ε**.

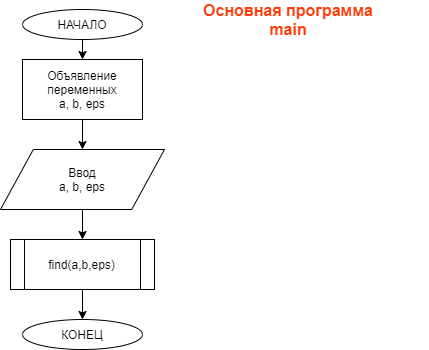
1. Если **a > b**, то меняем местами **a** и **b**.
2. Если **f(a) \* f(b) > 0** – *Вывод о ошибке*, прекращение работы программы.
3. Если **f(a) · f ’’(a) > 0,** то **x0 = a**, иначе **x0 = b**.
4. **xn=x0-f(x0) / f `(x0) –** первое приближение
5. **x0 = xn**
6. **xn=x0-f(x0) / f `(x0)**
7. Если **|x0 - xn | > eps**, то вернуться к 5

Выходные данные: **x** и номер итерации

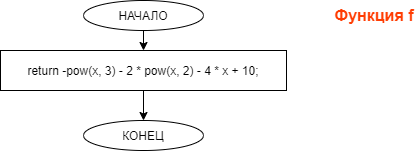
Значение **x** является решением с заданной точностью **ε**  уравнения вида **f(x) = 0**.

**Структурная схема алгоритма.**

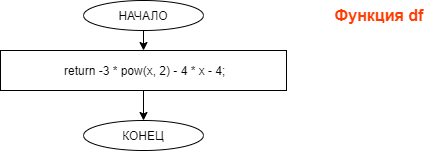
**Основная часть программы.**



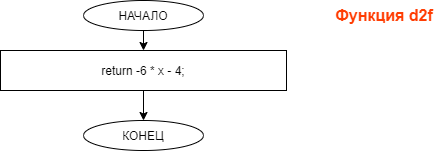
Функция double f(double x)  
Назначение: возвращает значение функции -x3-2\*x2-4\*x+10 в точке **х**   
Сигнатура функции: double f (double x)  
Параметры: double x – значение аргумента



Функция double df(double x)  
Назначение: возвращает значение функции -3x2-4x-4в точке **х**   
Сигнатура функции: double df (double x)  
Параметры: double x – значение аргумента



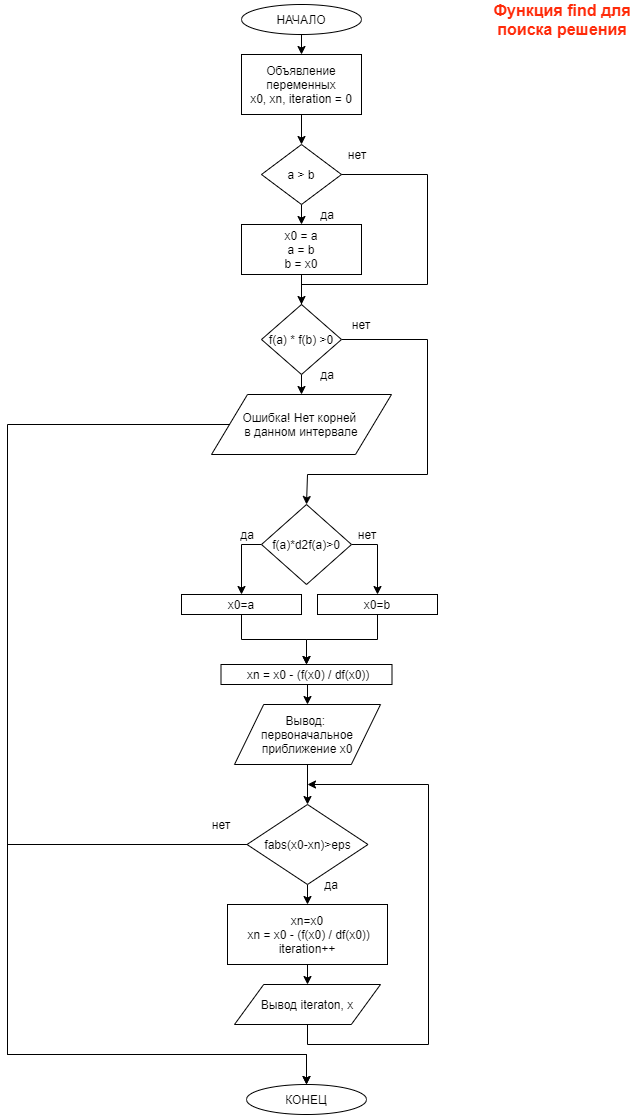
Функция double d2f(double x)  
Назначение: возвращает значение функции -6x-4 в точке **х**   
Сигнатура функции: double d2f (double x)  
Параметры: double x – значение аргумента



Функция void find(double a, double b, double eps)

Назначение: поиск решения методом ньютона с выводом текущего значения корня и итерации  
Сигнатура функции: void find(double a, double b, double eps)  
Параметры: double a, double b – граница отрезка, на котором ищем корень,

double eps - требуемая точность.



**Код программы**

#include <iostream>

#include <iomanip>

#include <math.h>

using namespace std;

double f(double x) {

//-x^3-2\*x^2-4\*x+10

return -pow(x, 3) - 2 \* pow(x, 2) - 4 \* x + 10;

}

double df(double x) {

//-3x^2-4x-4

return -3 \* pow(x, 2) - 4 \* x - 4;

}

double d2f(double x) {

//-6x-4

return -6 \* x - 4;

}

void find(double a, double b, double eps) {

int iteration = 0;

double x0; // первое приближение

double xn;

if (a > b) { //если пользователь перепутал границы местами

x0 = a; //x0 используется как временная переменная

a = b; //для замены местами концов отрезков

b = x0;

}

if ((f(a) \* f(b)) > 0) {

cout << "Ошибка! Нет корней в данном интервале!";

return;

}

if (f(a) \* d2f(a) > 0) {

x0 = a;

}

else {

x0 = b;

}

xn = x0 - (f(x0) / df(x0)); //высчитываем первое приближение

cout << "Первоначальное приближение x0 = " << xn << " " << endl;

while(fabs(x0-xn)>eps){

x0 = xn;

xn = x0 - (f(x0) / df(x0));

iteration++;

cout << "iteration: " << iteration;

cout << " |";

cout << " x: " << setprecision(15) << xn << endl;

}

}

int main()

{

system("color F0");

setlocale(LC\_ALL, "Russian");

double a;

double b;

double eps;

cout << "Введите a для отрезка [a,b]: " << endl;

cin >> a;

cout << "Введите b для отрезка [a,b]: " << endl;

cin >> b;

cout << "Введите необходимую точность: ";

cin >> eps;

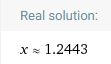
find(a,b,eps);

return 1;

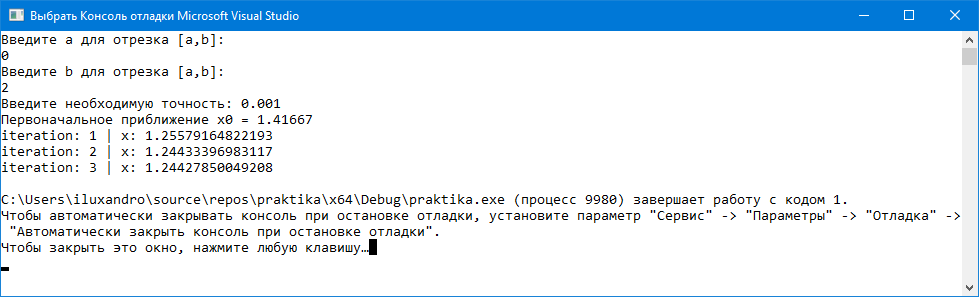
}

**Тест программы.**

Эталонный результат:



Полученный результат:



Работа программы корректна.  
Реальный результат совпал с ожидаемым

# **Вывод.**

Разработка программы успешно завершена, т. к. полученные результаты совпадают с ожидаемыми.

**Вариант №7.**

**Задание 2.**

**Оглавление**

* Постановка задачи
* Описание математического метода
* Входные данные
* Выходные данные
* Структурная схема алгоритма
* Код программы
* Тесты
* Вывод

**Постановка задачи**

1)Решите систему линейных уравнений методом *“Простой итерации”* с точностью 0.001, предварительно оценив число достаточных для этого итераций:

Где

2) Полученное решение используйте для вычисления невязки каждого уравнения.

3) Все полученные приближения решения системы привести в итоговом отчете.

4) Не забываем начинать отчет с формулировки задания.

**Описание математического метода.**

**Метод итерации**. Суть такого метода заключается в нахождении по приближённому значению величины следующего приближения (являющегося более точным). Метод позволяет получить значения корней системы с заданной точностью в виде предела последовательности некоторых векторов (итерационный процесс). Характер сходимости и сам факт сходимости метода зависит от выбора начального приближения корня **x0**.

**Описание метода**

Пусть дана СЛАУ вида: AX=B, где:

;

Предполагая, что {\displaystyle a\_{ii}} не равно 0, {\displaystyle i={\overline {1,n}}}. Выразим  {\displaystyle x\_{1}} через первое уравнение, {\displaystyle x\_{2}} — через второе и т. д.

Обозначим:

В матричном виде получим:

За нулевое приближение примем столбец свободных членов

– нулевое приближение.

– первое приближение

– второе приближение и т.д.

**Условие сходимости процесса**

Условие сходимости что называют условием диагонального преобладания исходной матрицы A.

**Словесный алгоритм**.

1. Вводим с проверкой на ошибки исходную матрицу
2. Вводим с проверкой на ошибки матрицу с диагональным преобладанием
3. Приводим матрицу (2) к новому виду
4. Проверяем сходимость, высчитывая норму матрицу
5. Высчитываем норму вектора столбца свободных членов
6. Высчитываем теоретическую нижнюю грань количества итераций
7. Ищем решение
   1. Ищем следующую итерацию n+1
   2. Ищем максимальную разницу между предыдущим и текущим решением
   3. Присваиваем предыдущему решение текущее
   4. Если Max(7.2) больше требуемой точности возвращаемся к (7.1)
8. Высчитываем невязку

**Входные данные**

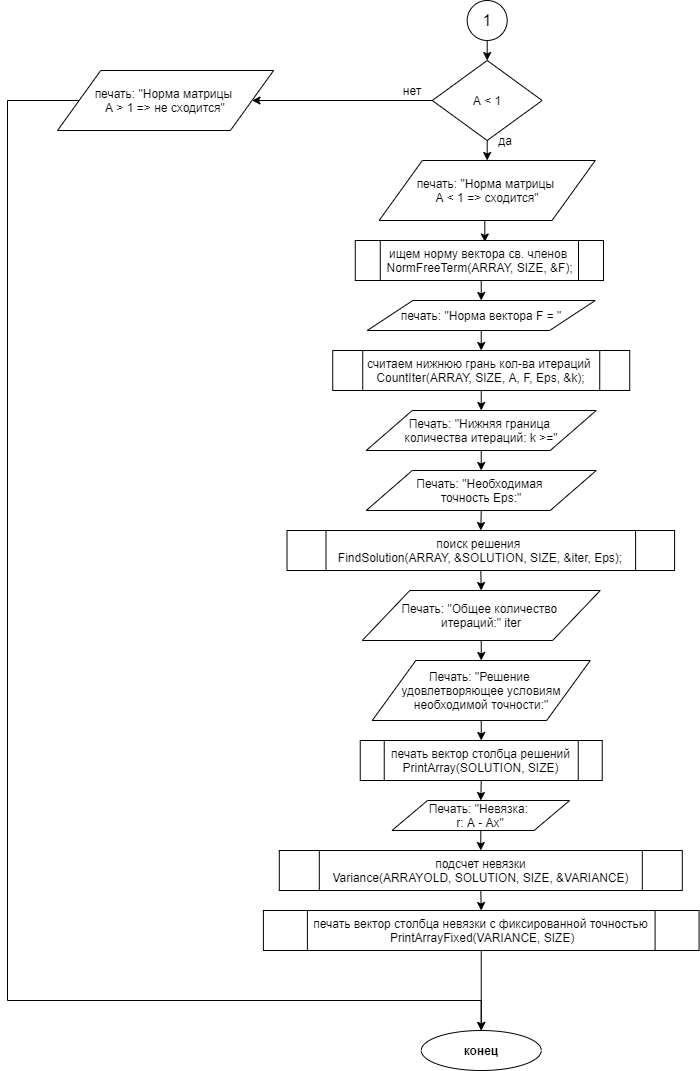
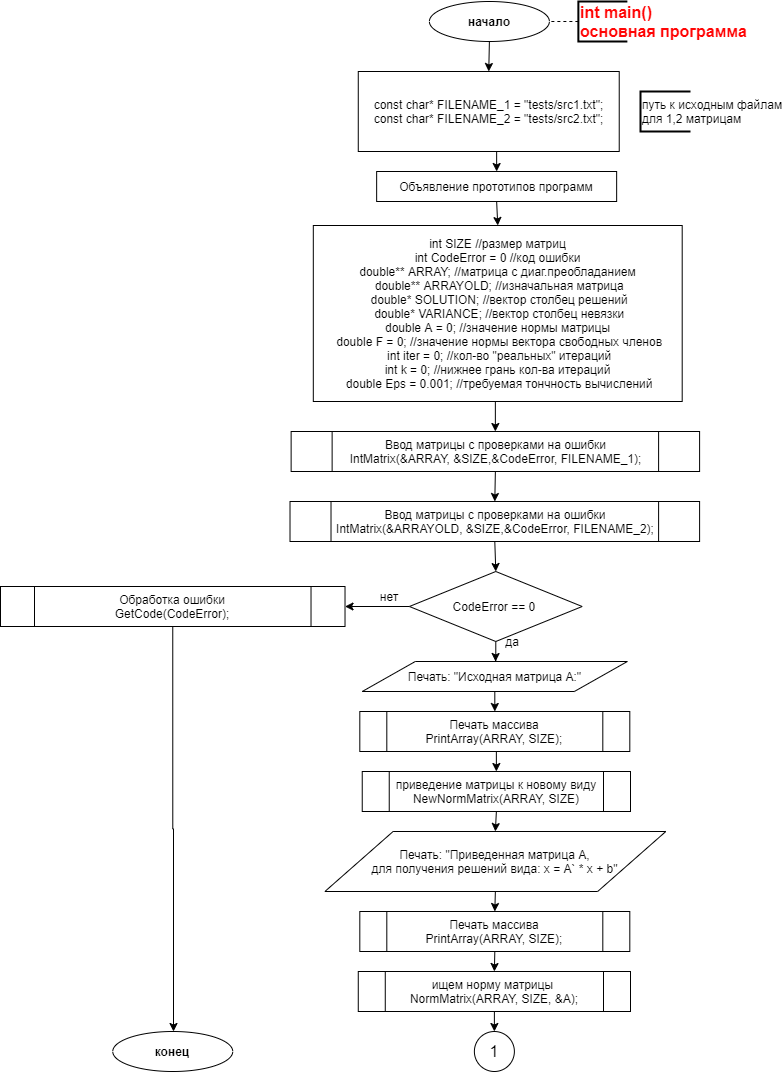
* Исходная СЛАУ и ее размеры
* СЛАУ приведенная к диагональному методу и ее размеры
* Требуемая точность
* Путь к исходным данным обеих матриц

**Выходные данные**

* Нижняя грань итераций, подсчитанная теоретически
* Норма матрицы
* Норма вектора
* Вектор решений
* Вектор невязки
* Количество реальных итераций

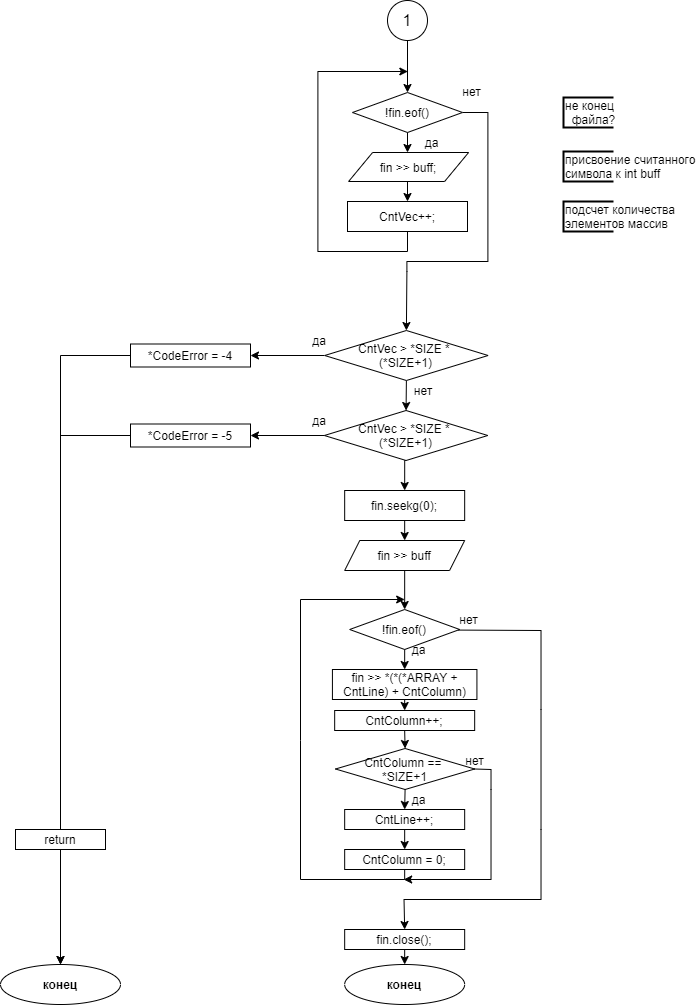
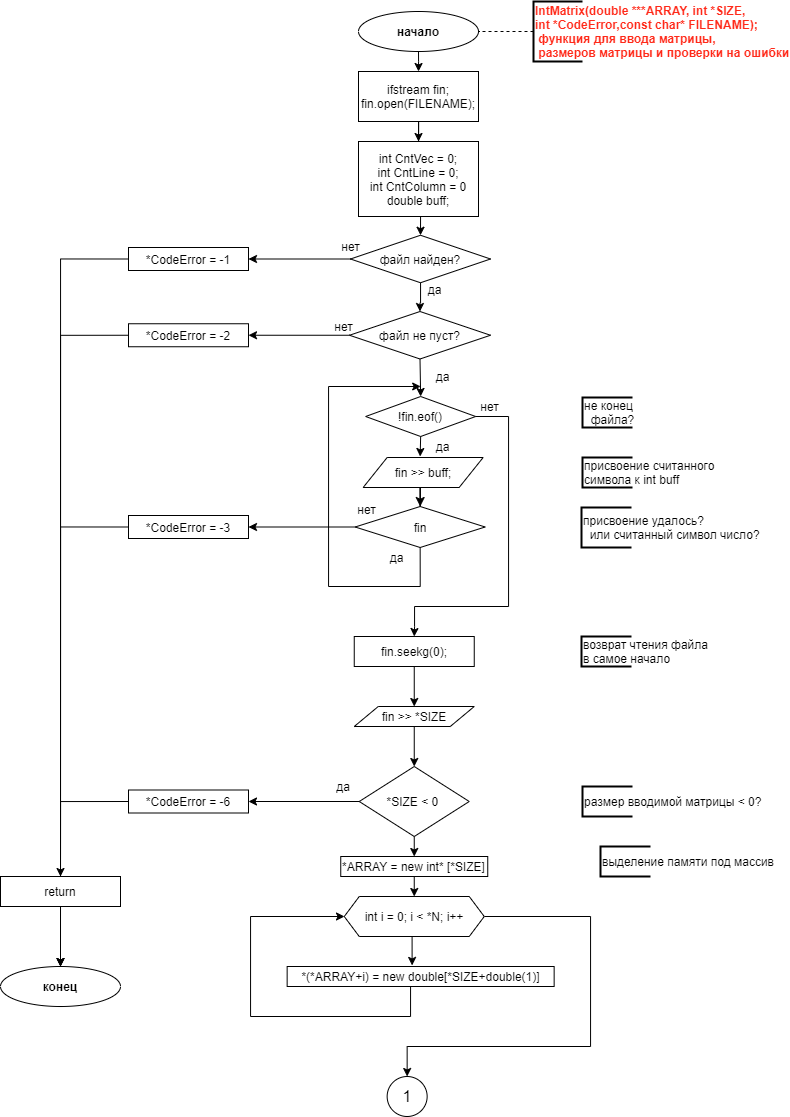
**Структурная схема алгоритма.**

**Основная часть программы**



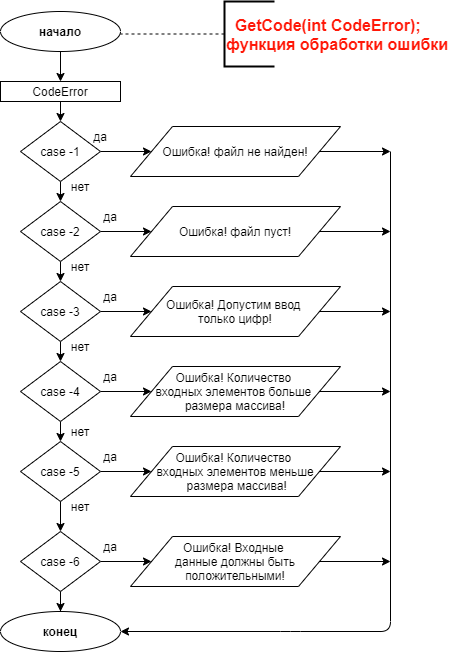
**Функция** void IntMatrix(double\*\*\* ARRAY, int\* SIZE, int\* CodeError, const char\* FILENAME)

**Назначение:** считывает матрицу из файла и проверяет корректность входных данных   
**Сигнатура функции:** void IntMatrix(double\*\*\* ARRAY, int\* SIZE, int\* CodeError, const char\* FILENAME)  
**Параметры:** double\*\*\* ARRAY – матрица, возвращаемая в программу, int\* SIZE – размеры матрицы, возвращаемые в программу, int\* CodeError – код ошибки, возвращаемый в программу, const char\* FILENAME – файл, из которого происходит ввод



**Функция** void GetCode(int CodeError)

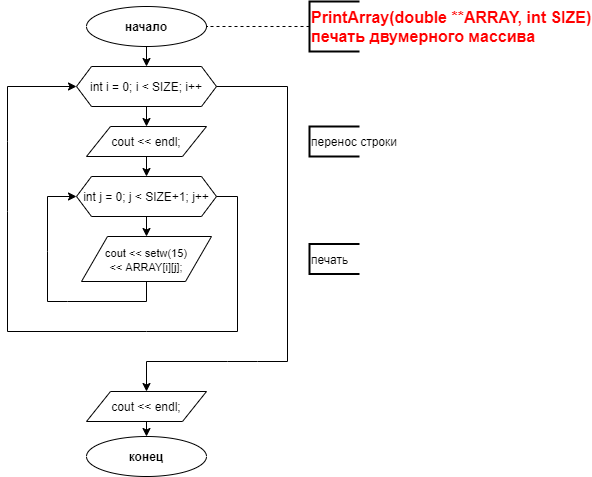
**Назначение:** вывод ошибку по его коду  
**Сигнатура функции:** void GetCode(int CodeError)  
**Параметры:** int CodeError – код ошибки



# 

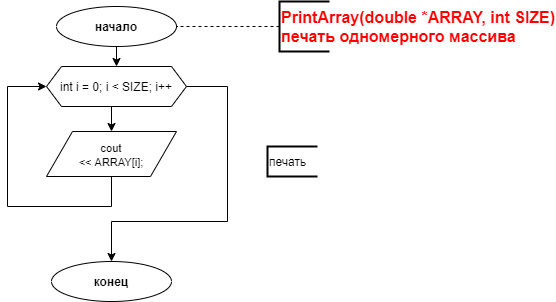
**Функция** void PrintArray(double\*\* ARRAY, int SIZE)

**Назначение:** вывод двумерного массива  
**Сигнатура функции:** void PrintArray(double\*\* ARRAY, int SIZE)  
**Параметры:** double\*\* ARRAY – выводимый массив, int SIZE – его размеры



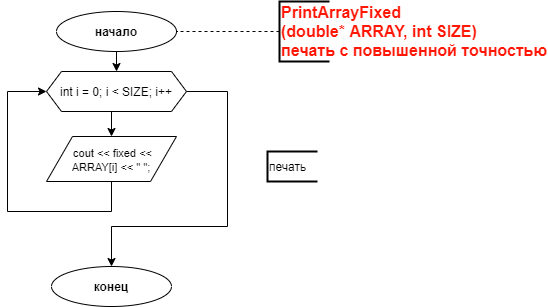
**Функция** void PrintArray(double\* ARRAY, int SIZE)

**Назначение:** вывод одномерного массива  
**Сигнатура функции:** void PrintArray(double\* ARRAY, int SIZE)  
**Параметры:** double\* ARRAY – выводимый массив, int SIZE – его размеры



**Функция** void PrintArrayFixed(double\* ARRAY, int SIZE)

**Назначение:** печать с повышенной точностью  
**Сигнатура функции:** void PrintArrayFixed(double\* ARRAY, int SIZE)  
**Параметры:** double\* ARRAY – выводимый массив, int SIZE – его размеры

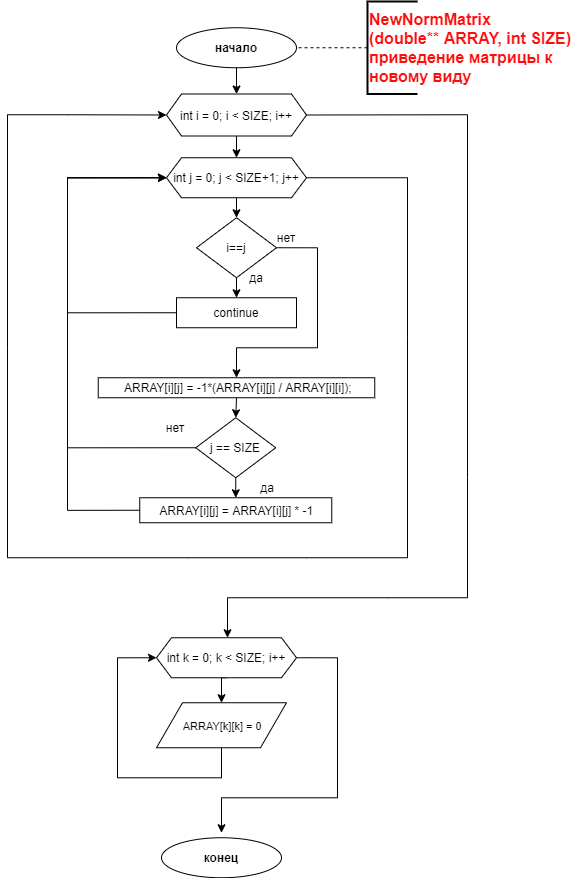


**Функция** void NewNormMatrix(double\*\* ARRAY, int SIZE)

**Назначение:** приведенной матрицы для получения решений вида x=A`x + b

**Сигнатура функции:** void NewNormMatrix(double\*\* ARRAY, int SIZE)

**Параметры:** double\*\* ARRAY – массив, int SIZE – его размеры

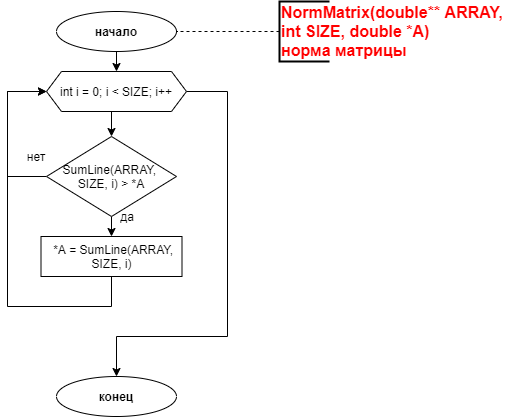


**Функция** void NormMatrix(double\*\* ARRAY, int SIZE, double \*A)

**Назначение:** подсчет нормы матрицы

**Сигнатура функции:** void NormMatrix(double\*\*ARRAY, int SIZE,double \*A)

**Параметры:** double\*\* ARRAY – массив, int SIZE – его размеры, double \*A - значение нормы матрицы

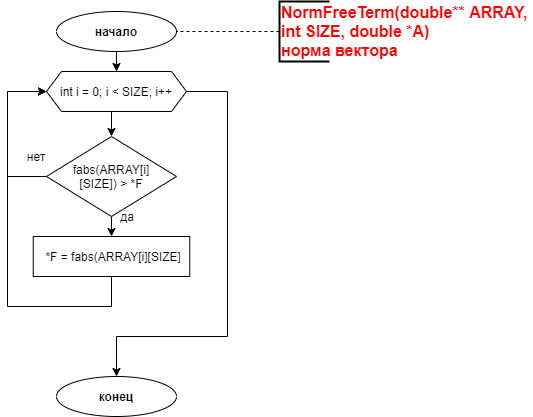


**Функция** void NormFreeTerm(double\*\* ARRAY, int SIZE, double \*F)

**Назначение:** подсчет нормы вектора

**Сигнатура функции:** void NormFreeTerm(double\*\*ARRAY, int SIZE, double \*F)

**Параметры:** double\*\* ARRAY – массив, int SIZE – его размеры, double \*F - значение нормы вектора

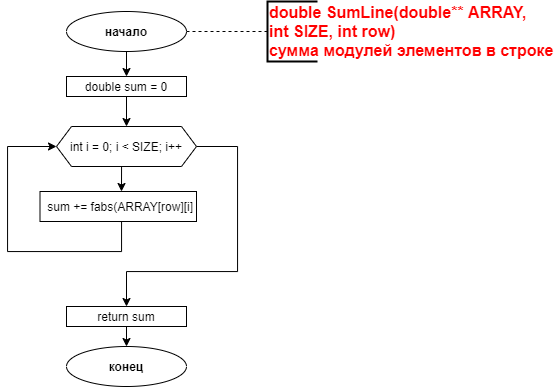


**Функция** double SumLine(double\*\* ARRAY, int SIZE, int row)

**Назначение:** функция суммирующая модули элементов всей строки

**Сигнатура функции:** double SumLine(double\*\* ARRAY, int SIZE, int row)

**Параметры:** double\*\* ARRAY – массив, int SIZE – его размеры, int row – столбец в котором ищется сумма

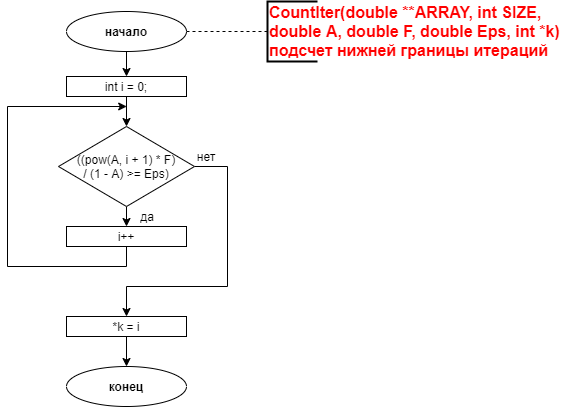


**Функция** void CountIter(double \*\*ARRAY, int SIZE, double A, double F, double Eps, int \*k)

**Назначение**: функция суммирующая модули элементов всей строки

**Сигнатура функции:** void CountIter(double \*\*ARRAY, int SIZE, double A, double F, double Eps, int \*k)

**Параметры:** double\*\* ARRAY – массив, int SIZE – его размеры, double A – значение нормы матрицы, double F – значение нормы вектора, double Eps – требуемая точность, int \*k – количество итераций

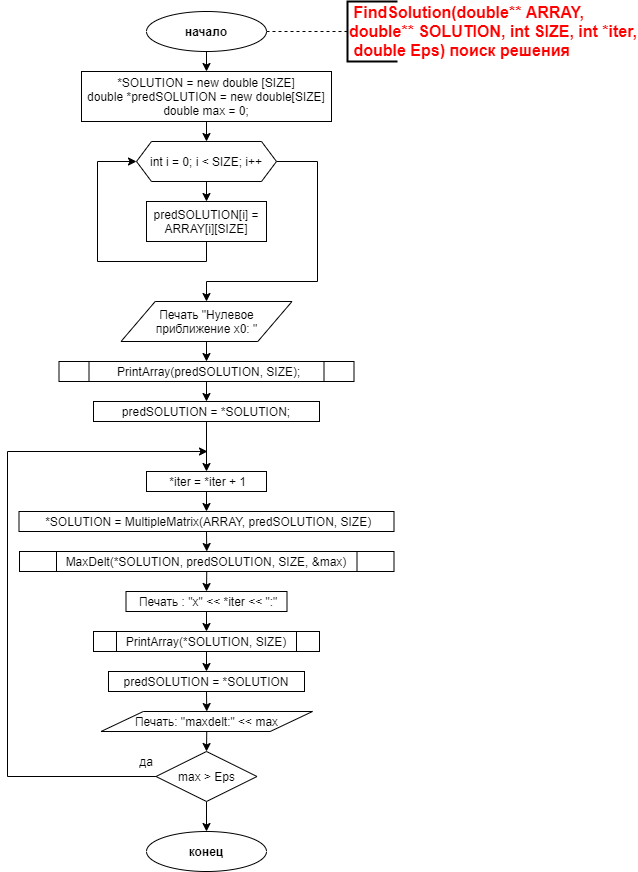


**Функция** void FindSolution(double\*\* ARRAY, double\*\* SOLUTION, int SIZE, int \*iter, double Eps)

**Назначение**: поиск решения

**Сигнатура функции:** void FindSolution(double\*\* ARRAY, double\*\* SOLUTION, int SIZE, int \*iter, double Eps)

**Параметры:** double\*\* ARRAY – массив, int SIZE – его размеры, double\*\* SOLUTION – вектор решений, int \*iter – количество реальных итераций, double Eps – требуемая точность

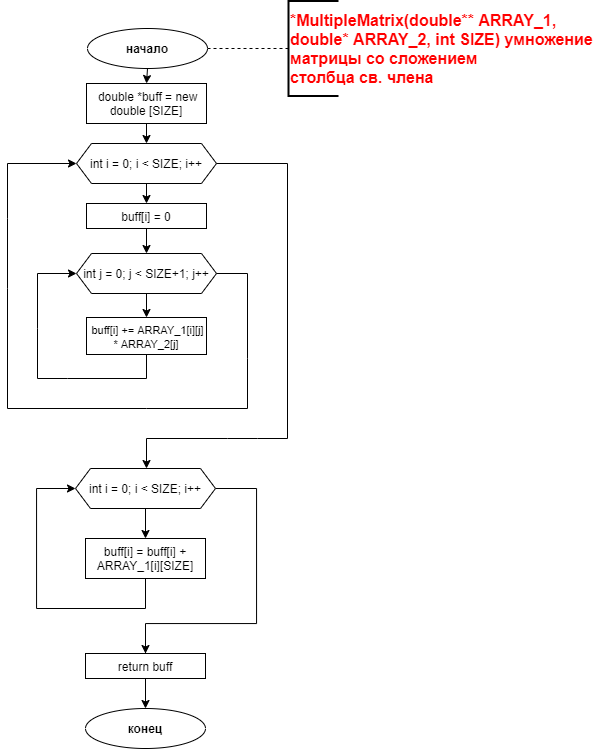


**Функция** double \*MultipleMatrix(double\*\* ARRAY\_1, double\* ARRAY\_2, int SIZE)

**Назначение**: умножение матрицы A` на матрицу решения для перехода на следующую итерацию. Также добавляется столбец из свободных членов по условию метода.

**Сигнатура функции:** double \*MultipleMatrix(double\*\* ARRAY\_1, double\* ARRAY\_2, int SIZE)

**Параметры:** double\*\* ARRAY\_1 – матрица A`, на которую умножается вектор решений, double\* ARRAY\_2 – вектор столбец из решений, int SIZE – размер матрицы

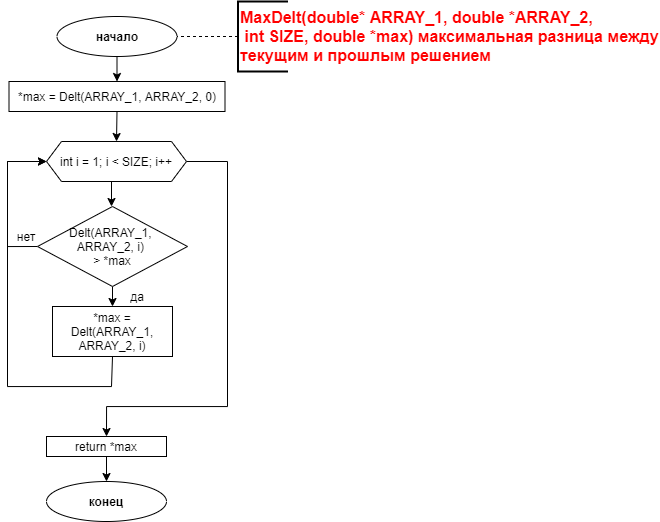


**Функция** double MaxDelt(double\* ARRAY\_1, double \*ARRAY\_2, int SIZE, double \*max)

**Назначение**: поиск максимальной разницы междутекущим и прошлым решением

**Сигнатура функции:** double MaxDelt(double\* ARRAY\_1, double \*ARRAY\_2, int SIZE, double \*max)

**Параметры:** double\*\* ARRAY\_1 – вектор текущего/пред. решения, double \*ARRAY\_2 -– вектор текущего/пред. решения, int SIZE – размеры матрицы, double \*max – максимальная разница, возвращаемая в функцию.

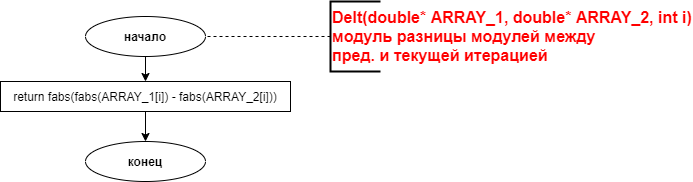


**Функция** double Delt(double\* ARRAY\_1, double\* ARRAY\_2, int i)

**Назначение**: поиск максимальной разницы между текущим и прошлым решением в столбце

**Сигнатура функции:** double Delt(double\* ARRAY\_1, double\* ARRAY\_2, int i)

**Параметры:** double\*\* ARRAY\_1 – вектор текущего/пред. решения, double \*ARRAY\_2 -– вектор текущего/пред. решения, int i – столбец, в котором ищется максимальная разница

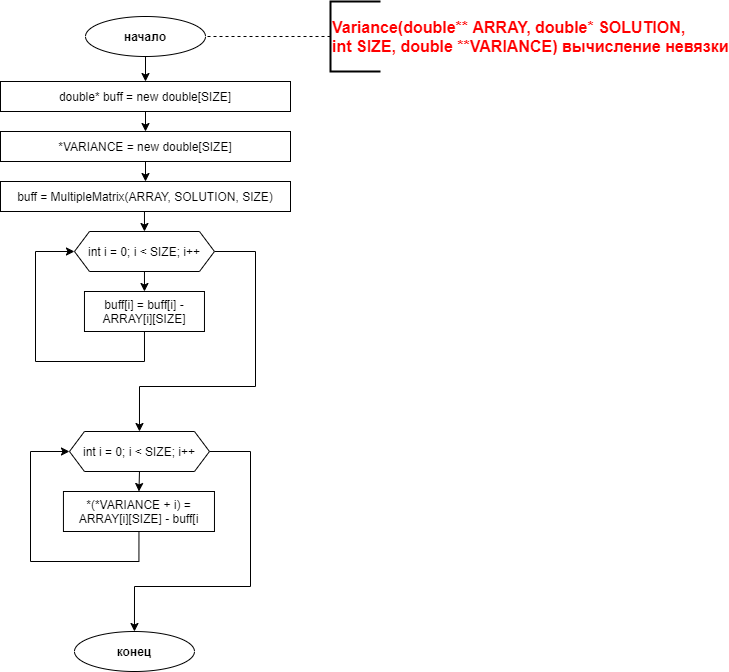


**Функция** void Variance(double\*\* ARRAY, double\* SOLUTION, int SIZE, double \*\*VARIANCE)

**Назначение**: поиск невязки, возвращаемая в функцию

**Сигнатура функции:** void Variance(double\*\* ARRAY, double\* SOLUTION, int SIZE, double \*\*VARIANCE)

**Параметры:** double\*\* ARRAY – изначальная матрица без преобразований, double\* SOLUTION – вектор столбец решений, int SIZE – размеры матрицы, double \*\*VARIANCE – вектор столбец невязки, возвращаемый в функцию.



**Код программы.**

#include <fstream>

#include <iomanip>

#include <cmath>

using namespace std;

const char\* FILENAME\_1 = "src.txt"; //путь к исходным файлам для матрицы с диагональным преобладанием

const char\* FILENAME\_2 = "src2.txt"; //путь к исходным файлам для исходной матрицы

void IntMatrix(double\*\*\* ARRAY, int\* SIZE, int\* CodeError, const char\* FILENAME);

//ввод матрицы проверка на ошибки

void GetCode(int CodeError);

//вывод ошибки по соотвествующему коду

void PrintArray(double\*\* ARRAY, int SIZE);

//печать двумерного массива

void PrintArray(double\* ARRAY, int SIZE);

//печать одномерного массива

void PrintArrayFixed(double\* ARRAY, int SIZE);

//печать с высокой точностью

void NewNormMatrix(double\*\* ARRAY, int SIZE);

//преобразование матрицы

void NormMatrix(double\*\* ARRAY, int SIZE, double \*A);

//норма матрицы

void NormFreeTerm(double\*\* ARRAY, int SIZE, double \*F);

//норма вектора свободных членов

double SumLine(double\*\* ARRAY, int SIZE, int row);

//сумма модулей элементо в строке

void CountIter(double\*\* ARRAY, int SIZE, double A, double F, double Eps, int \*k);

//поиск нижней границы количества итераций

void FindSolution(double\*\* ARRAY, double\*\* SOLUTION, int SIZE, int \*iter, double Eps);

//поиск решения

double \*MultipleMatrix(double\*\* ARRAY\_1, double\* ARRAY\_2, int SIZE);

//умножение матрицы на вектор столбец с добавлением столбца свободных членов

double MaxDelt(double\* ARRAY\_1, double\* ARRAY\_2, int SIZE, double \*max);

//макисмальная разница между пред. и текущей итерацией

double Delt(double\* ARRAY\_1, double\* ARRAY\_2, int i);

//поиск модули разница между модулями пред. и текущей итерации в некотором столбце

void Variance(double\*\* ARRAY, double\* SOLUTION, int SIZE, double\*\* VARIANCE);

//поиск невязки

int main() {

system("color F0"); //задаем цвет консоли

setlocale(LC\_ALL, "Russian"); //задаем язык

int SIZE; //размер матрицы

int CodeError = 0; //код ошибки

double\*\* ARRAY; //объявление матрицы с диаг.преобладанием

double\*\* ARRAYOLD; //объявление изначальной матрицы

double\* SOLUTION; //вектор столбец решений

double\* VARIANCE; //вектор столбец невязки

double A = 0; //значение нормы матрицы

double F = 0; //значение нормы вектора свободных членов

int iter = 0; //кол-во "реальных" итераций

int k = 0; //нижнее грань кол-ва итераций

double Eps = 0.001; //требуемая тончность вычислений

IntMatrix(&ARRAY, &SIZE, &CodeError, FILENAME\_1);

IntMatrix(&ARRAYOLD, &SIZE, &CodeError, FILENAME\_2);

if (CodeError == 0) {

cout << "Исходная матрица A: ";

PrintArray(ARRAY, SIZE);

NewNormMatrix(ARRAY, SIZE);

cout << endl << "---------------------------------------------------" << endl << endl;

cout << "Приведенная матрица A`:" << endl << "вида: x = A` \* x + b" << endl;

PrintArray(ARRAY, SIZE);

NormMatrix(ARRAY, SIZE, &A);

if (A < 1) {

cout << endl << "Норма матрицы A = " << A << " < 1 => сходится";

}

else {

cout << endl << "Норма матрицы A = " << A << " > 1 => не сходится";

return 1;

}

NormFreeTerm(ARRAY, SIZE, &F);

cout << endl << "Норма вектора F = " << F;

CountIter(ARRAY, SIZE, A, F, Eps, &k);

cout << endl << "Нижняя граница количества итераций: k >= " << k << endl;

cout << "Необходимая точность Eps: " << Eps << endl;

system("pause");

FindSolution(ARRAY, &SOLUTION, SIZE, &iter, Eps);

cout << "Общее количество итераций: " << iter << endl;

cout << endl << "Решение удовлетворяющее условиям необходимой точности: " << endl;

PrintArray(SOLUTION, SIZE);

cout << endl << endl << "Невязка: r: А - Ax" << endl;

Variance(ARRAYOLD, SOLUTION, SIZE, &VARIANCE);

PrintArrayFixed(VARIANCE, SIZE);

}

else {

GetCode(CodeError);

}

}

void IntMatrix(double\*\*\* ARRAY, int\* SIZE, int\* CodeError, const char\* FILENAME) {

/\*

функция для ввода матрицы, размеров матрицы и проверки на различные ошибки

\*/

ifstream fin; //создаем объект класса ifstream

fin.open(FILENAME); //открываем файл

int CntVec = 0; //счетчик реального размера массива

int CntLine = 0; //счетчик для ввода

int CntColumn = 0; //счетчик для ввода

double buff; // буффер для проверки корректности вводимых данных

if (!fin) // если файл не найден

{

\*CodeError = -1;

return;

}

if (fin.peek() == EOF) //если найденный файл пуст

{

\*CodeError = -2;

return;

}

while (!fin.eof()) { //проверка на ввод цифр

fin >> buff;

if (!fin) {

\*CodeError = -3;

return;

}

}

fin.seekg(0); //возврат чтения файла на самое начало

fin >> \*SIZE; //считываем количество строк

if (\*SIZE < 0) {

\*CodeError = -6;

}

\*ARRAY = new double\* [\*SIZE];

for (int i = 0; i < \*SIZE; i++) {

\*(\*ARRAY + i) = new double[\*SIZE+double(1)];

}

while (!fin.eof()) { //чтение элементов массива, пока элементы не закончатся

fin >> buff;

CntVec++; //подсчет количества элементов массив

}

if (CntVec > \*SIZE \* (\*SIZE+1)) { //проверка на выход за границы массива

\*CodeError = -4;

return;

}

if (CntVec < \*SIZE \* (\*SIZE + 1)) { //проверка на недостаток элементов

\*CodeError = -5;

return;

}

fin.seekg(0); //возврат чтения файла на самое начало

fin >> buff; //так как вернулись на самое начало, записываем в буффер размеры

while (!fin.eof()) {

fin >> \*(\*(\*ARRAY + CntLine) + CntColumn); //чтение массива из файла

CntColumn++;

if (CntColumn == \*SIZE+1) {

CntLine++;

CntColumn = 0;

}

}

fin.close(); // закрываем файл

}

void GetCode(int CodeError) {

switch (CodeError)

{

case -1:

cout << "Ошибка! файл не найден" << endl << endl;

break;

case -2:

cout << "Ошибка! файл пуст!" << endl << endl;

break;

case -3:

cout << "Ошибка! Допустим ввод только цифр!" << endl << endl;

break;

case -4:

cout << "Ошибка! Количество входных элементов больше размера массива!" << endl << endl;

break;

case -5:

cout << "Ошибка! Количество входных элементов меньше размера массива!" << endl << endl;

break;

case -6:

cout << "Ошибка! Входные данные должны быть положительными!" << endl << endl;

break;

}

}

void PrintArray(double\*\* ARRAY, int SIZE) { //печать двумерного массива

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

cout << endl;

for (int j = 0; j < SIZE+1; j++) {

cout << setw(15) << ARRAY[i][j];

}

}

cout << endl;

}

void PrintArray(double\* ARRAY, int SIZE) { //печать одномерного массива

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

cout << ARRAY[i] << " ";

}

}

void PrintArrayFixed(double\* ARRAY, int SIZE) { // печать с фиксированной

точностью используемая для печати невязки

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

cout << fixed << ARRAY[i] << " ";

}

}

void NewNormMatrix(double\*\* ARRAY, int SIZE) {

/\*

для получения решения нам нужно исходную матрицу преобразовать к виду x=A`x + b

то есть мы делим строчку на коэффиценты стоящие у диагонального x

после "переносим все в правую часть, то есть умножаем на -1

если попадется свободный столбик со свободным членом знак оставляем

после зануляем диагональ, т.к. мы выразили нужный нам x

\*/

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

for (int j = 0; j < SIZE+1; j++) {

if (i == j) { //если попался диагональный элемент - пропускаем,

зануление будет позже

continue;

}

ARRAY[i][j] = -1\*(ARRAY[i][j] / ARRAY[i][i]); //делим на коэффиценты

стоящие у диагонального x и "переносим"

if (j == SIZE) {

ARRAY[i][j] = ARRAY[i][j] \* -1; //возвращаем знак

свободного члена обратно

}

}

}

for (int k = 0; k < SIZE; k++) { //зануляем диагональ

ARRAY[k][k] = 0;

}

}

void NormMatrix(double\*\* ARRAY, int SIZE, double \*A) {

/\*

ищем норму матрицы:

нормой матрицы является максимум суммы среди всех строчек

сумма ищется как сумма модулей всех элементов

A=max{сумма по i;|a1i|...сумма по i;|ani|}

используется для определения сходимости

\*/

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

if (SumLine(ARRAY, SIZE, i) > \*A) {

\*A = SumLine(ARRAY, SIZE, i);

}

}

}

void NormFreeTerm(double\*\* ARRAY, int SIZE, double \*F) {

/\*

ищем норму вектора F

нормой вектора является максимальной элемент

среди столбца свободных членов

F=max{|bi|}

\*/

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

if (fabs(ARRAY[i][SIZE]) > \*F) {

\*F = fabs(ARRAY[i][SIZE]);

}

}

}

double SumLine(double\*\* ARRAY, int SIZE, int row) {

/\*

Функция суммирующая модули элементов всей строки,

где row - столбец в котором нужно посчитать сумма

прим.(столбец свободных членов пропускается)

\*/

double sum = 0;

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

sum += fabs(ARRAY[row][i]);

}

return sum;

}

void CountIter(double \*\*ARRAY, int SIZE, double A, double F, double Eps, int \*k) {

/\*

функция для подсчета нижней грани количества итераций

A^(k+1)\*F/(1-A)<=Eps

где k количество итераций

\*/

int i = 0;

while ((pow(A, i + 1) \* F) / (1 - A) >= Eps) {

i++;

}

\*k = i;

}

void FindSolution(double\*\* ARRAY, double\*\* SOLUTION, int SIZE, int \*iter, double Eps) {

/\*

Функция для поиска решения

алгоритм следующий:

первое приближение - столбик из свободных членов преобразованной матрицы(A`)

после высчитываем новое решение, ищем максимальную разницу между

пред. и текущим решением для проверки выхода из цикла

после присваем значения пред. итерации нынешней

\*/

\*SOLUTION = new double [SIZE];

double \*predSOLUTION = new double[SIZE];

double max = 0;

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

predSOLUTION[i] = ARRAY[i][SIZE]; //x0 - самое первое приближение это столбик из свободных членов

}

cout << "Нулевое приближение x0: " << endl;

PrintArray(predSOLUTION, SIZE);

cout << endl << endl;

predSOLUTION = \*SOLUTION; //x0=x1

do{

\*iter = \*iter + 1;

\*SOLUTION = MultipleMatrix(ARRAY, predSOLUTION, SIZE); //теперь получили x1

MaxDelt(\*SOLUTION, predSOLUTION, SIZE, &max);

cout << "x" << \*iter << ":";

PrintArray(\*SOLUTION, SIZE);

predSOLUTION = \*SOLUTION;

cout << endl << "maxdelt:" << max << endl << endl;

} while (max > Eps);

}

double \*MultipleMatrix(double\*\* ARRAY\_1, double\* ARRAY\_2, int SIZE) {

/\*

функция для умножения матрицы A на столбец решений для

перехода на следующую итерацию, x^(k+1) = A' \* x^k + F`

также добавляется столбец свободных членов по условию

перехода на следующую итерацию

функция возвращает x^k+1

\*/

double \*buff = new double [SIZE]; //объявляем матрицу буфер, в которой будет лежать результат

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

buff[i] = 0;

for (int j = 0; j < SIZE; j++) { //умножаем исходную матрицу A

buff[i] += ARRAY\_1[i][j] \* ARRAY\_2[j]; //на вектор столбец решений прошлых итераций

}//и сохраняем в buff матрицу

}

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

buff[i] = buff[i] + ARRAY\_1[i][SIZE];

}

return buff;

}

double MaxDelt(double\* ARRAY\_1, double \*ARRAY\_2, int SIZE, double \*max) {

/\*

функция для поиска максимальной дельты

максимальная дельта - это максимальная разница между

текущим и прошлым решением

данное число является условием для выхода из цикла

\*/

\*max = Delt(ARRAY\_1, ARRAY\_2, 0);

for (int i = 1; i < SIZE; i++) {

if (Delt(ARRAY\_1, ARRAY\_2, i) > \*max) {

\*max = Delt(ARRAY\_1, ARRAY\_2, i); //ищем самую большую дельту

}

}

return \*max;

}

double Delt(double\* ARRAY\_1, double\* ARRAY\_2, int i) {

/\*модуль разницы модулей между пред. и текущей итерацией \*/

return fabs(fabs(ARRAY\_1[i]) - fabs(ARRAY\_2[i]));

}

void Variance(double\*\* ARRAY, double\* SOLUTION, int SIZE, double \*\*VARIANCE) {

/\*

вычисление невязки - расхождение приближенного равенства

r = F - Ax, где x - конечный вектор столбец решений,

F - первоначальный вектор столбец свободных членов

A - первоначальная матрица

функция возвращает вектор столбец с невязками по каждому иксу

\*/

double\* buff = new double[SIZE];

\*VARIANCE = new double[SIZE];

buff = MultipleMatrix(ARRAY, SOLUTION, SIZE);

for (int i = 0; i < SIZE ; i++) { //т.к. в умножении матрицы мы добавляем столбец

buff[i] = buff[i] - ARRAY[i][SIZE]; //свободных членов вычитаем его для получения

} //верного результата

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

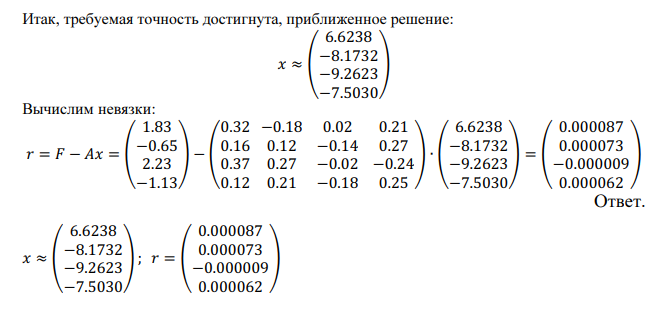
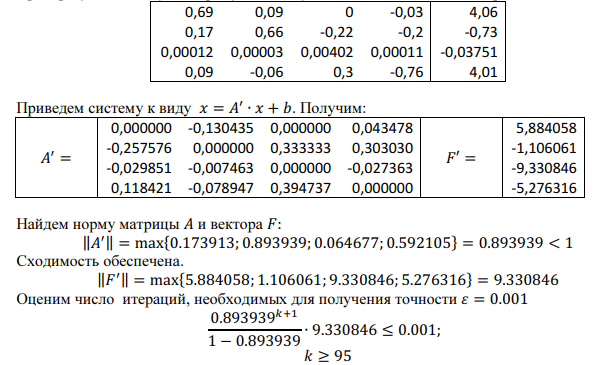
\*(\*VARIANCE + i) = ARRAY[i][SIZE] - buff[i];

}

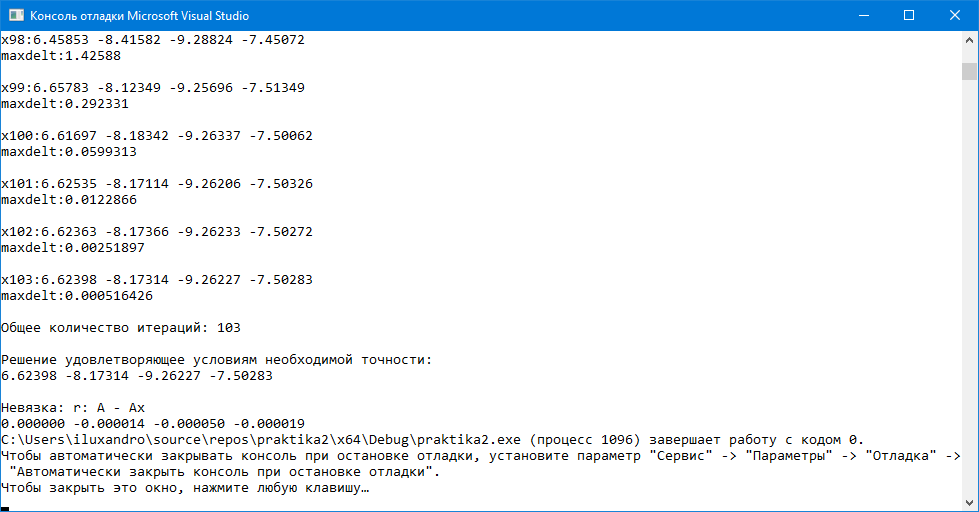
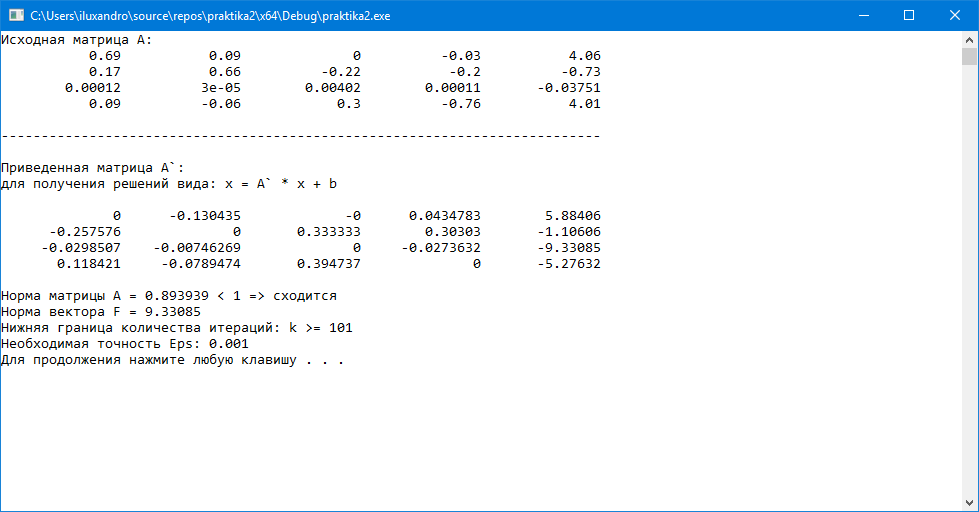
}

**Тест программы.**

**Эталонные результаты:**



**Полученный результат:**



Работа программы корректна.  
Реальный результат совпал с ожидаемым

# **Вывод.**

Разработка программы успешно завершена, т. к. полученные результаты совпадают с ожидаемыми.

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»**

**О Т З Ы В**

**РУКОВОДИТЕЛЯ ПРАКТИКИ**

Студент Кузнецов Илья Игоревич

Институт №3 “Системы управления, информатика и электроэнергетика”

###### Кафедра№304 “Вычислительные машины, системы и сети”

##### Учебная группа M30-109Б

Направление подготовки (специальность) 09.03.01 “Информатика и вычислительная техника”

*(шифр)(название направления, специальности)*

Вид практики учебная

*(учебная, производственная, преддипломная или другой вид практики)*

*Наименование предприятия:* Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский авиационный институт (национальный исследовательский университет)»

*Название структурного подразделения (отдел, лаборатория):* Кафедра 304

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*План работ выполнен: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(полностью/не полностью)*

*Соответствие практики образовательным компетенциям:*

ПСК-10, ПСК-13\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель практики от МАИ

Секретарёв Виталий Евгеньевич/ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(фамилия, имя, отчество) (подпись)*

Москва 2020 г.