Санкт-Петербургский Политехнический Университет Петра Великого

Институт прикладной математики и механики

Кафедра прикладной математики

Курсовой проект (курсовая работа)   
по дисциплине "Численные методы"

Выполнил студент гр. 23631/3 **Кузин А.В.**

Преподаватель: **Павлова Л.В.**

Санкт-Петербург

2018

Оглавление

[1. Решение алгебраических и трансцендентных уравнений: метод Ньютона и метод половинного деления 4](#_Toc533471969)

[1.1. Формулировка задачи и её формализация 4](#_Toc533471970)

[*1.2.* Алгоритмы методов 4](#_Toc533471971)

[1.2.1. Метод половинного деления 4](#_Toc533471972)

[1.2.2. Метод Ньютона 5](#_Toc533471973)

[*1.3.* Предварительный анализ задачи 5](#_Toc533471974)

[1.3.1. Алгебраическое уравнение 5](#_Toc533471975)

[1.3.2. Трансцендентное уравнение 6](#_Toc533471976)

[1.4. Тестовый пример с детальными расчётами 6](#_Toc533471977)

[1.4.1. Метод половинного деления 6](#_Toc533471978)

[1.4.2. Метод Ньютона 6](#_Toc533471979)

[1.5. Иллюстрация в MATLAB 7](#_Toc533471980)

[1.6. Модульная структура программы 8](#_Toc533471981)

[1.7. Анализ решения задачи 9](#_Toc533471982)

[1.7.1. Алгебраическое уравнение 9](#_Toc533471983)

[1.7.2. Трансцендентное уравнение 9](#_Toc533471984)

[1.8. Выводы 10](#_Toc533471985)

[2. Решение СЛАУ прямыми методами: Метод квадратного корня. 10](#_Toc533471986)

[2.1. Формулировка задачи и её формализация 10](#_Toc533471987)

[2.2. Алгоритм метода 10](#_Toc533471988)

[2.3. Предварительный анализ задачи и подготовка матрицы 11](#_Toc533471989)

[2.4. Тестовый пример с ручными расчётами 12](#_Toc533471990)

[2.5. Модульная структура программы 13](#_Toc533471991)

[2.6. Анализ решения задачи 13](#_Toc533471992)

[2.7. Вывод 14](#_Toc533471993)

[3. Решение СЛАУ итерационными методами: метод простых итераций 15](#_Toc533471994)

[3.1. Формулировка задачи и её формализация 15](#_Toc533471995)

[3.2. Алгоритм метода 15](#_Toc533471996)

[3.3. Предварительный анализ задачи и подготовка матрицы 15](#_Toc533471997)

[3.4. Тестовый пример 15](#_Toc533471998)

[3.5. Структура программы 16](#_Toc533471999)

[3.6. Численный анализ решения задачи 17](#_Toc533472000)

[3.7. Вывод 17](#_Toc533472001)

[4. Отчёт по лабораторной работе «Решение алгебраической проблемы собственных значений» 19](#_Toc533472002)

[4.1. Формулировка задачи и её формализация 19](#_Toc533472003)

[4.2. Алгоритм метода 19](#_Toc533472004)

[4.3. Предварительный анализ задачи и подготовка матрицы 19](#_Toc533472005)

[4.4. Тестовый пример 19](#_Toc533472006)

[4.5. Модульная структура программы 20](#_Toc533472007)

[4.6. Численный анализ решения 20](#_Toc533472008)

[4.7. Вывод 22](#_Toc533472009)

# Решение алгебраических и трансцендентных уравнений: метод Ньютона и метод половинного деления

## Формулировка задачи и её формализация

Решить два уравнения: алгебраическое и трансцендентное. С помощью двух методов:

* Метод половинного деления
* Метод Ньютона

Для алгебраического уравнения аналитически найти промежутки для его корней по теореме о верхней границе. Исследовать зависимость количества итераций методов от заданной точности.

**Алгебраическое уравнение:**

**Трансцендентное уравнение:**

Число 𝑥∗ называется приближенным решением уравнения, если |𝑥− 𝑥∗|≤ 𝜀.   
Необходимо найти такие решения.

## Алгоритмы методов

### Метод половинного деления

Если функция непрерывна на промежутке [a; b] и знаки значений функции на концах промежутка различны, то находим и выбираем новый промежуток, чтобы знаки на концах промежутка отличались, повторяем до тех пор, пока , где k – число итераций.

### Метод Ньютона

Задаётся начальное приближение вблизи предположительного корня из границ выбранного промежутка по условию , 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.

Тогда следующая итерация

Выполняется с условием того, что .

Итерационный процесс завершается, если ,

Где

## Предварительный анализ задачи

### Алгебраическое уравнение

Вычислим границы интервалов, на которых располагаются корни по теореме о верхней границе:

**Найдём верхнюю и нижнюю границы действительных корней.**

1. Верхняя граница  
   Делим полином на старший коэффициент, после среди коэффициентов ищем максимальный коэффициент по абсолютному значению и индекс первого отрицательного коэффициента. – формула для реализации на Си.
2. Нижняя граница  
   Вычисляем значения функции при –x. Далее аналогично верхней границе, кроме итоговой формулы -Итоговый интервал - [-5.05378, 17.43316]

### Трансцендентное уравнение

Рассмотрим уравнение

Из области значений косинуса понятно, что корни могут существовать только на промежутке [-1,1].

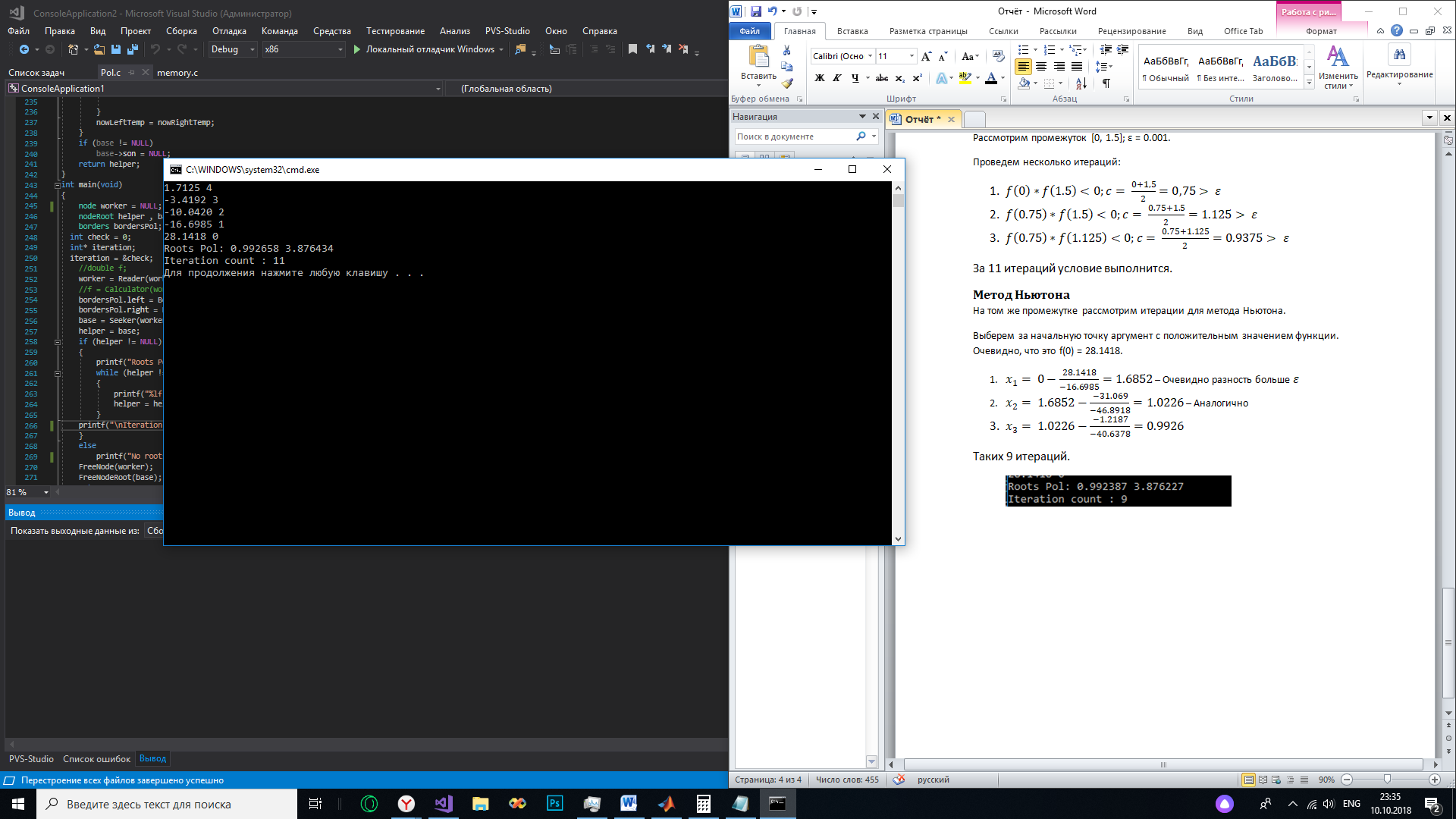
## Тестовый пример с детальными расчётами

### Метод половинного деления

Рассмотрим промежуток [0, 1.5]; ε = 0.001.

Проведем несколько итераций:

За 11 итераций условие выполнится.

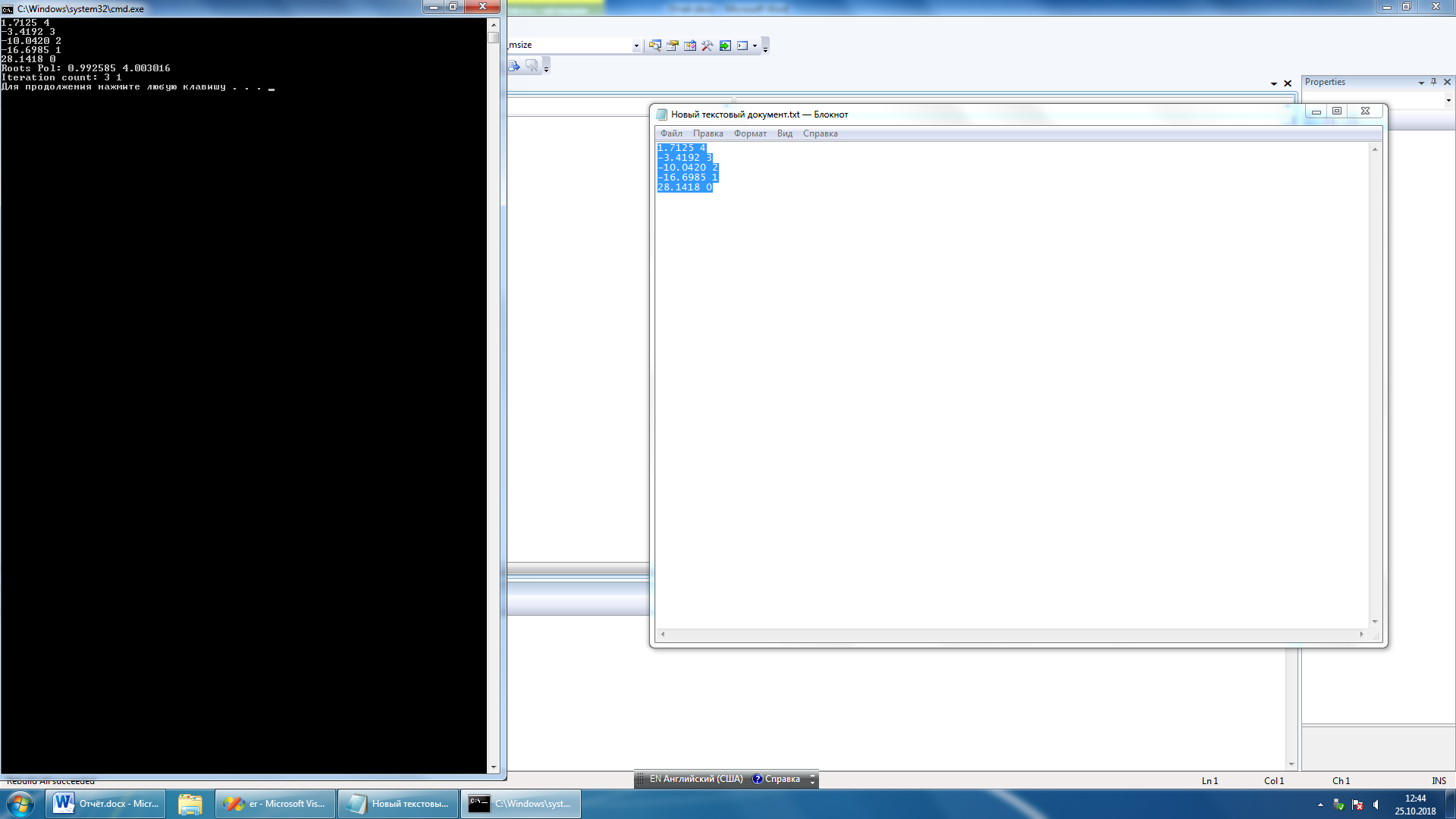


### Метод Ньютона

На том же промежутке рассмотрим итерации для метода Ньютона.

Выберем за начальное приближение аргумент с положительным значением функции.  
Очевидно, что это f(0) = 28.1418.

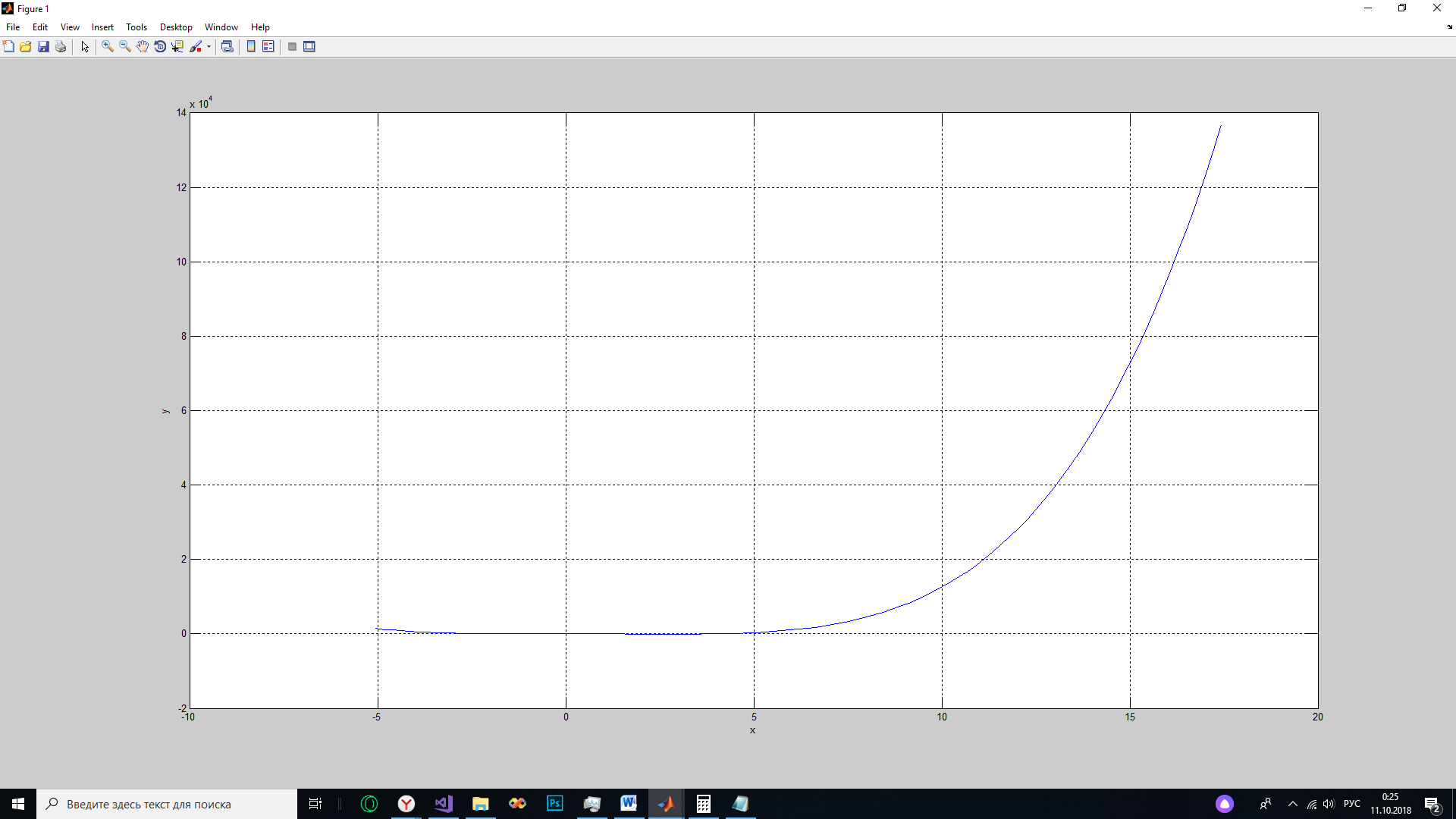
1. – Очевидно разность больше
2. – Аналогично



Аналогичные расчёты с трансцендентной функцией.

## Иллюстрация в MATLAB

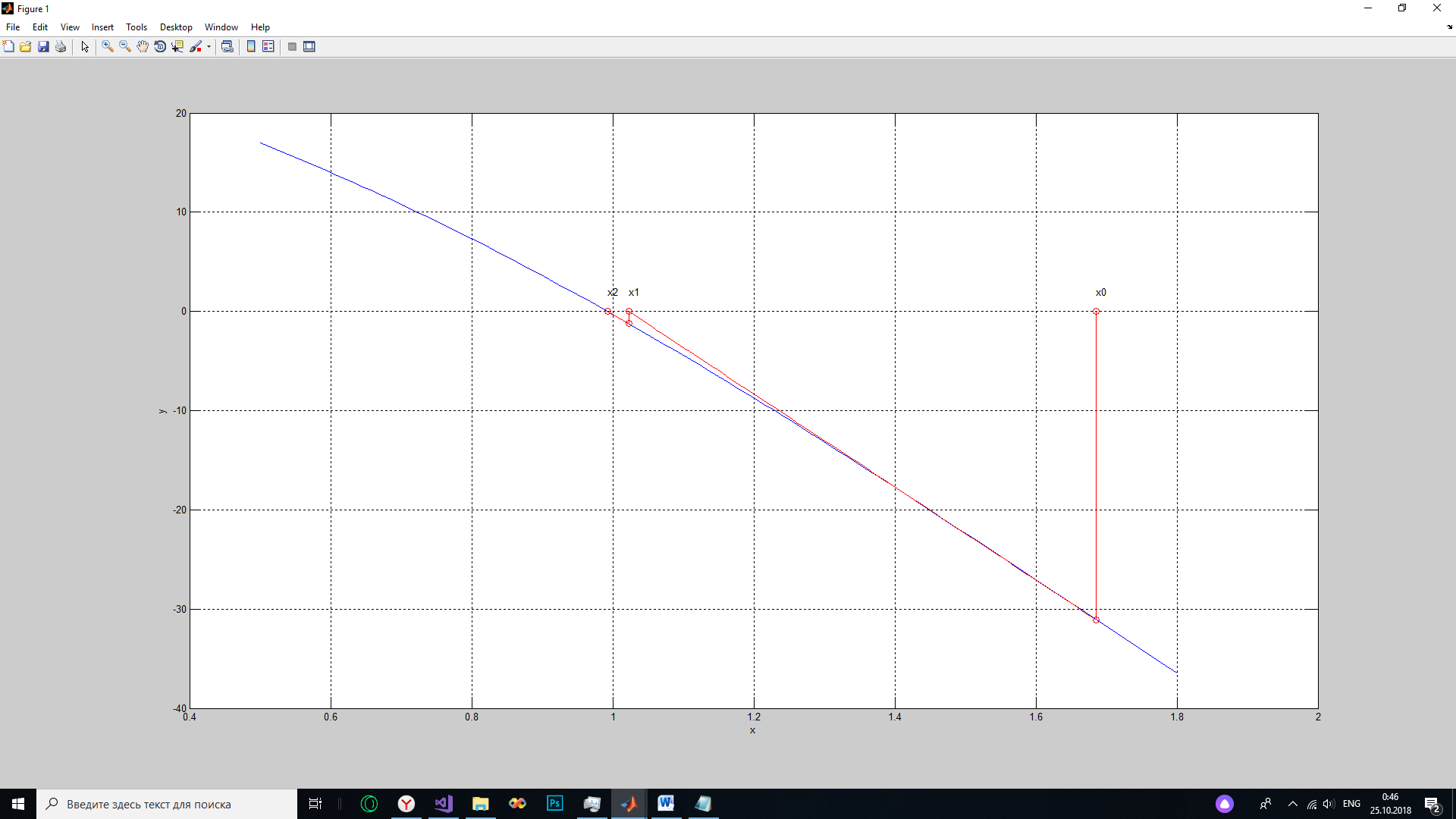
График полинома на интервале найденных границ



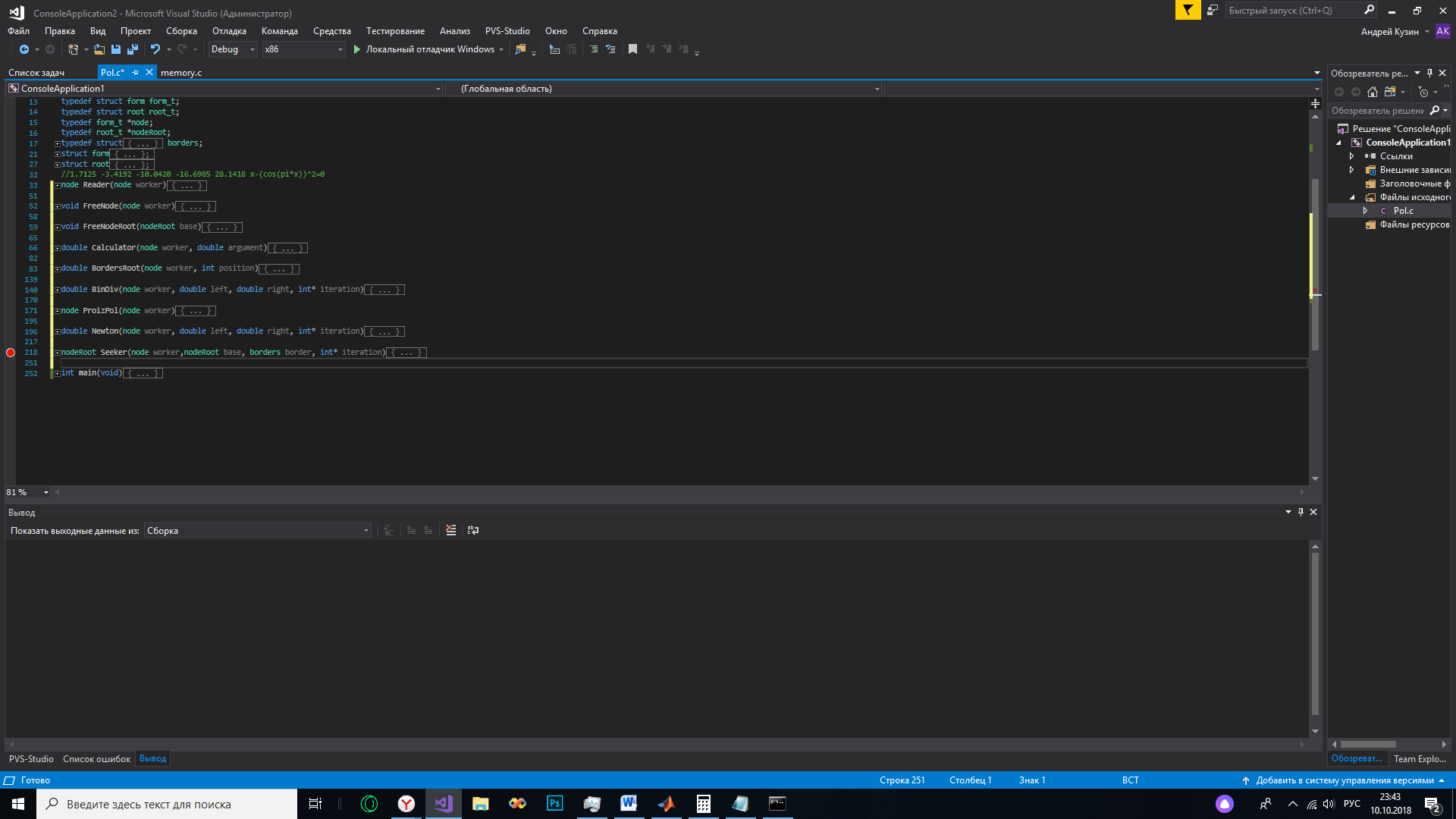
Возьмём интервал [0.5, 1.8] и построим более наглядный график



Теперь, используя метод Ньютона, построим первые итерации



## Модульная структура программы



* Функция Reader() считывает коэффициенты полинома и записывает их в структуру списка.
* FreeNode() служит для очищения памяти, выделенной под список коэффициентов полинома.
* FreeNodeRoot() служит для очищения памяти, выделенной под список корней.
* Calculator() использует список и аргумент для вычисления значения полинома в точке.
* BordersRoot() поиск верхней и нижней границ. Второй аргумент отвечает за выбор границы.
* BinDiv() – реализация метода половинного деления. Основана на рекурсивном выполнении с указателем для инкременации счётчика итераций в поиске корня с заданной точностью.
* ProizPol() – используя список, создаёт новый список, производную от полинома.
* Newton() – реализация метода Ньютона с аналогичным счётчиком
* Seeker() – функция хождения по заданному промежутку на часть длины промежутка в поисках разнознаковых концов

## Анализ решения задачи

### Алгебраическое уравнение

1. X1 = 0.992387 (Полученный результат в реализации Х1 = 0.99238)

|  |  |  |
| --- | --- | --- |
|  | Число итераций | |
| МПД | Метод Ньютона |
| 10-2 | 6 | 3 |
| 10-3 | 9 | 3 |
| 10-5 | 16 | 4 |

1. X2 = 3.87623 (Полученный результат в реализации Х2 = 3.87623)

|  |  |  |
| --- | --- | --- |
|  | Число итераций | |
| МПД | Метод Ньютона |
| 10-2 | 6 | 1 |
| 10-3 | 9 | 3 |
| 10-5 | 16 | 4 |

### Трансцендентное уравнение

1. X1 = 0.311516 (Полученный результат в реализации Х1 = 0.311516)

|  |  |  |
| --- | --- | --- |
|  | Число итераций | |
| МПД | Метод Ньютона |
| 10-2 | 4 | 1 |
| 10-3 | 8 | 1 |
| 10-5 | 14 | 2 |

1. X2 = 0.894916 (Полученный результат в реализации Х2 = 0.894916)

|  |  |  |
| --- | --- | --- |
|  | Число итераций | |
| МПД | Метод Ньютона |
| 10-2 | 4 | 1 |
| 10-3 | 8 | 1 |
| 10-5 | 14 | 2 |

1. X3 = 1 (Полученный результат в реализации при высокой точности Х3 = 1)

|  |  |  |
| --- | --- | --- |
|  | Число итераций | |
| МПД | Метод Ньютона |
| 10-2 | 4 | 2 |
| 10-3 | 8 | 3 |
| 10-5 | 14 | 4 |

## Выводы

Методу Ньютона требуется меньше итераций, чем методу дихотомии даже на первых примерах – это разумное следствие из-за квадратичной сходимости метода Ньютона. Отсюда следует, что целесообразнее использовать метод Ньютона, но для задач, где хватает точности вычислений в несколько знаков после запятой – оба метода пригодны для использования.

# Решение СЛАУ прямыми методами: Метод квадратного корня.

## Формулировка задачи и её формализация

Найти решения СЛАУ методом квадратного корня. Проверить вычислительную ошибку (сравнивая с точным решением) для матриц с разными числами обусловленности. Размерность системы не менее 10.

## Алгоритм метода

1. A = AT, если А – симметричная положительно определённая матрица
2. Представим матрицу А в виде LDLT, где

Матрица D в данном случае позволяет снять ограничение на условие положительной определённости исходной матрицы и избежать мнимых элементов в матрице L.

Для элементов матрицы L справедливы следующие формулы

Далее представляя решаем , где D не вызовет сложностей в умножении, так как подразумевает смену знака в столбце при необходимости.

Теперь когда нашли

## Предварительный анализ задачи и подготовка матрицы

В MATLAB’е строим по формулам:

*В матрице D ненулевые элементы стоят на главной диагонали, благодаря чему получается симметричная матрица.*

*Для вычислительной ошибки:*

## Тестовый пример с ручными расчётами

L11 = L22 = L33 =

L21 = L31 = L32=

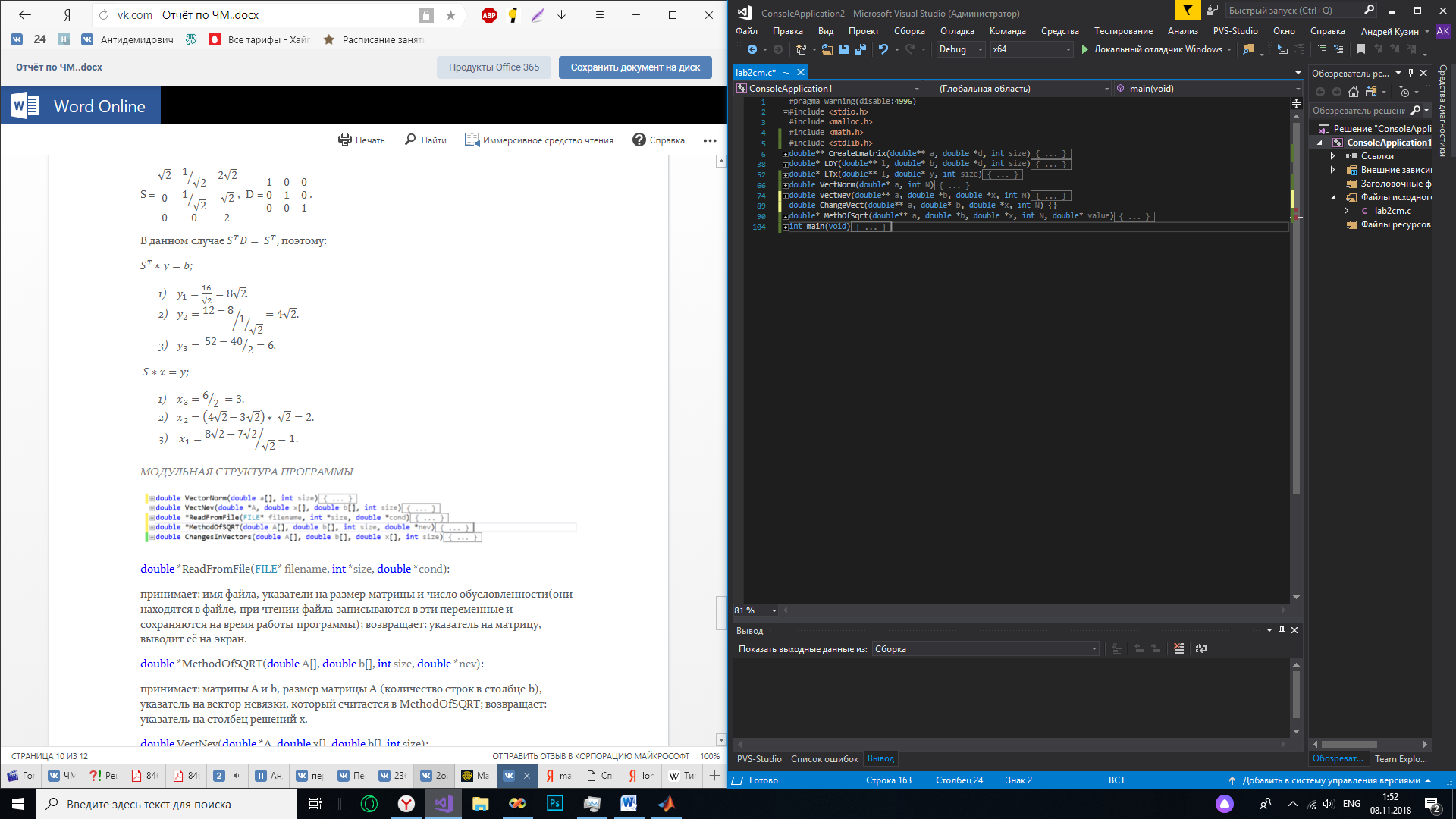
L12 = 0 L13 = 0 L23 = 0

D1 = 1 D2 = 1D3 = 1

Так как , то

* y1 =
* y2 =
* y3=
* X3 =
* X2 =
* X1 =

## Модульная структура программы



CreateLMatrix() – принимая на вход исходную матрицу, размерность и массив для матрицы D, строит нижнетреугольную L матрицу, возвращая соответственно её же и неявно указатель на заполненную матрицу D.

LDY() – принимает: матрица L, матрица свободных коэффициентов b, вспомогательная матрица D, размерность матрицы. Выполняется поиск вектора y из формулы LDy = b. Возвращает вектор у.

LTx() – принимает: матрица L, вектор у, размерность матрицы. Выполняется поиск вектора корней из формулы . Возвращает вектор корней х.

VectNorm() – принимая вектор и его размер, считает и возвращает вторую норму

VectNev() – принимая матрицу A, вектора х и b, считает вектор невязки и вызывает функцию поиска нормы для её возвращения

ChangeVect() - принимает: матрицы A, вектора b, x и их размерность. Возвращает коэффициенты возмущений и выводит их на экран.

MethOfSqrt() – принимает матрицу А, вектор b и пустой вектор x, размерность и указатель на вектор невязки. Возвращает вектор корней и неявно вектор невязок

main() – выполняется чтение размерности, матрицы и коэффициента обусловленности для дальнейшего вызова основного алгоритма и вывода решения.

## Анализ решения задачи

Подготовим две матрицы – с маленьким числом обусловленности (А1) и большим (А2)

Для хорошо обусловленной матрицы:

Для плохо обусловленной матрицы:

*=* 464429084895.70801

## Вывод

При внесении возмущений в коэффициенты матриц А ошибка вычислений больше в матрице с большим числом обусловленности. При внесении возмущений в векторе свободных коэффициентов b ошибка в двух вариантах одного порядка. Коэффициенты не превосходят числа обусловленности матриц, также заметим, что для плохо обусловленной матрицы возрастает в тысячи раз.

# Решение СЛАУ итерационными методами: метод простых итераций

## Формулировка задачи и её формализация

Решение СЛАУ итерационным методом – методом простых итераций. Исследовать точность решения, когда определитель матрицы близок к 0.

(\***Метод итерации** – это численный метод решения математических задач, приближённый метод решения СЛАУ. 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).)

## Алгоритм метода

Для сходимости итерационного процесса является достаточным заданное условие .

1. Если исходная матрица симметричная и положительно-определённая, то А = Аисх, иначе матрица , \* b;
2. За x0 выбираем вектор b;
3. ;
4. ;

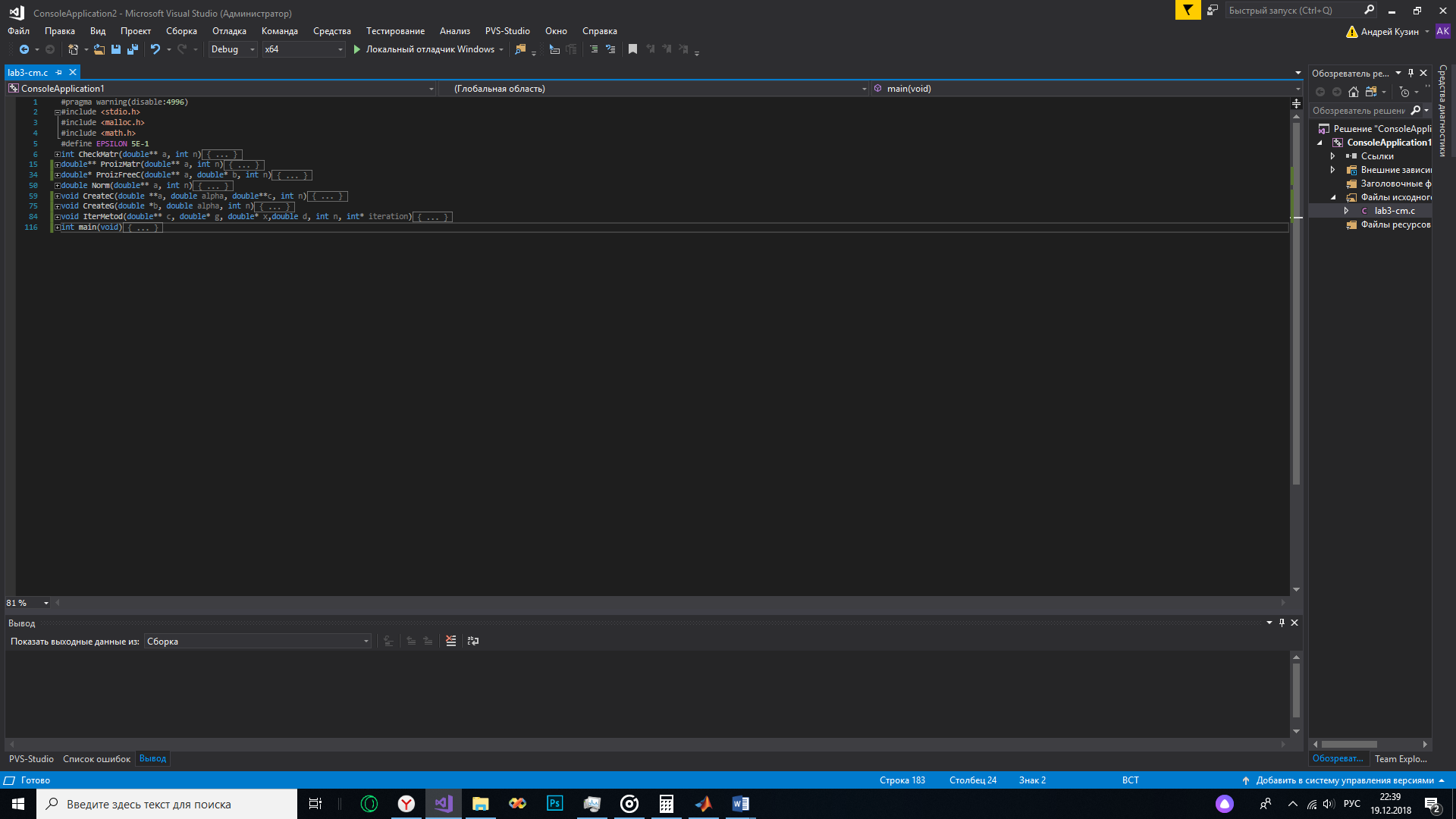
## Предварительный анализ задачи и подготовка матрицы

С помощью матлаба, используя создатель матриц из 2 работы, легко создаём матрицу А с произвольным определителем и вектор свободных коэффициентов b.

## Тестовый пример

1. *Домножим А и b на транспонированную матрицу А*
2. − 3;

## Структура программы



int CheckMatr(double\*\* a, int n)

Принимает: матрицу А и её размер; возвращает: 1 или 0 в зависимости от симметричности матрицы.

double\*\* ProizMatr(double\*\* a, int n)

Принимает: матрицу А и её размер; возвращает: матрицу А, умноженную слева на АТ.

double\* ProizFreeC(double\*\* a, double\* b, int n)

Принимает: матрицу А и её размер, вектор b; возвращает: вектор b, умноженный на AT.

double Norm(double\*\* a, int n)

Принимает: матрицу А и её размер; возвращает: евклидову норму А.

void CreateC(double \*\*a, double alpha, double\*\*c, int n)

Принимает: матрицу А и её размер, alpha и блок памяти под C; возвращает: вычисленную матрицу С.

void CreateG(double \*b, double alpha, int n)

Принимает: вектор b и его размер, alpha; возвращает: вектор b, умноженный на alpha.

void IterMetod(double\*\* c, double\* g, double\* x,double d, int n, int\* iteration)

Принимает: матрицу С и её размер, вектор g и x, число d, указатель на счётчик количества итераций;

Выполняется основной алгоритм и неявно возвращается вектор х.

int main(void)

Открывается и читается файл, вызов основный функций и вывод результатов на экран.

## Численный анализ решения задачи

Рассмотрим зависимость количества итераций от определителя матрицы (с малым и большим определителем)

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Норма вектора невязки. |
|  | 23 | 0.003486 |
|  | 28 | 0.000222 |
|  | 32 | 0.000028 |

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Норма вектора невязки. |
|  | 25 | 1.17419e-7 |
|  | 29 | 1.37925e-8 |
|  | 34 | 9.6421e-10 |

## Вывод

Исследован МПИ при , чем ближе определитель к нулю, тем больше итераций. Также из-за способа уменьшения определителя в программе (деление всех элементов исходной матрицы на константу) было обнаружено, что число итераций почти не меняется, а порядок вектора невязки уменьшается. Была установлена взаимосвязь сходимости от заданной точности - чем меньше 𝜀, тем больше итераций, но вектор невязки при этом уменьшается.

# Решение алгебраической проблемы собственных значений: метод Якоби

## Формулировка задачи и её формализация

Решение алгебраической проблемы собственных значений итерационным методом - итерационный метод Якоби. Исследовать сходимость при хорошей и плохой отделимости искомого собственного числа.

(\***Собственный вектор** — это ненулевой вектор, определяемый для квадратной матрицы или произвольного линейного преобразования, умножение матрицы на который или применение к которому преобразования даёт коллинеарный вектор — тот же вектор, умноженный на некоторое скалярное значение, называемое **собственным числом** **(значением)** матрицы или линейного преобразования.)

## Алгоритм метода

1. Формируем единичную матрицу, по размерности совпадающую с А;
2. Находим максимальный по модулю элемент среди элементов выше главной диагонали и запоминаем его индексы;
3. Находим угол поворота
4. Строим ортогональную матрицу простого поворота. Угол поворота определен так, чтобы в результате ортогонального преобразования элементы матрицы, стоящие позициях (i, j) и (j, i) обратились в нуль.
6. Если то повторяем всё со 2 шага, иначе выходим из итерационного процесса.

## Предварительный анализ задачи и подготовка матрицы

A – вещественная симметричная матрица, которую создаём при помощи пакета MatLab.

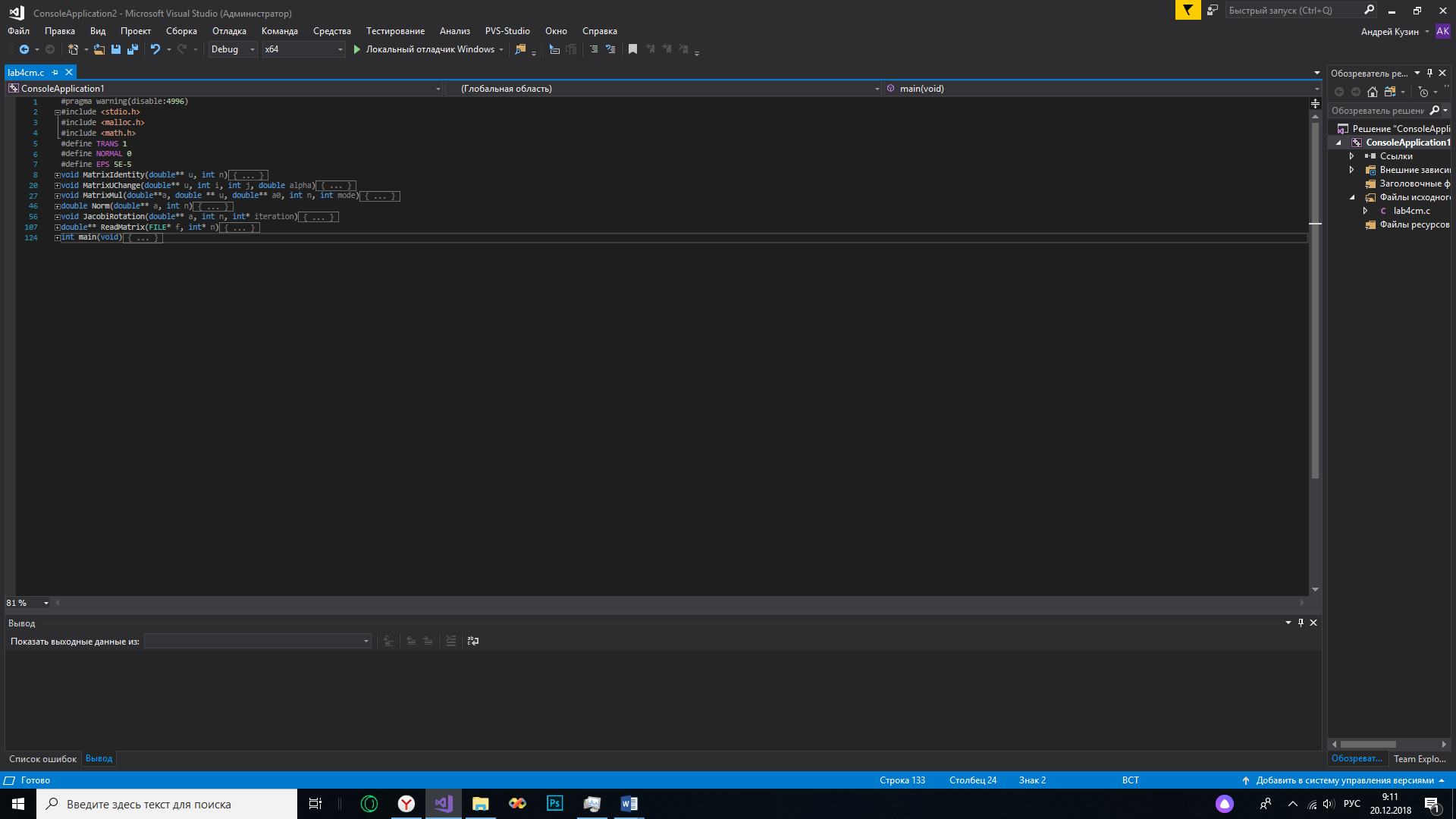
## Тестовый пример

1. *Находим максимальный недиагональный элемент*

*;*

1. *;*
2. *;*
3. *;*

## Модульная структура программы



void MatrixIdentity(double\*\* u, int n)

Принимает: указатель на блок памяти для матрицы NxN и размер N; возвращает: единичную матрицу.

void MatrixUChange(double\*\* u, int i, int j, double alpha)

Принимает: единичную матрицу U, индексы i и j, угол поворота alpha; возвращает: матрицу поворота.

void MatrixMul(double\*\*a, double \*\* u, double\*\* a0, int n, int mode)

Принимает: две матрицы данных и вспомогательную, размерность и режим умножения (в зависимости от умножения – на транспонированную или обычную матрицу); возвращает: итоговую матрицу после умножения.

double Norm(double\*\* a, int n)

Принимает: матрицу А и её размерность; возвращает: сумму квадратов недиагональных элементов.

void JacobiRotation(double\*\* a, int n, int\* iteration)

Функция итерационной реализации алгоритма, принимающая матрицу А и её размерность, а также указатель на количество итераций.

double\*\* ReadMatrix(FILE\* f, int\* n)

Происходит чтение матрицы и её размерности из файла.

## Численный анализ решения

Теперь изучим зависимость количества итераций от отделимости собственных чисел матрицы и числа её обусловленности на примере матриц 10x10.

1. В данной матрице собственные числа хорошо отделимы друг от друга и матрица имеет «хорошее» число обусловленности, т.е

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Максимальная норма |
|  | 64 | 1.302659e-2 |
|  | 79 | 1.387523e-3 |
|  | 87 | 2.086557e-4 |

1. В следующей матрице

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Максимальная норма |
|  | 1 | 5.260010e-4 |
|  | 1 | 5.260010e-4 |
|  | 15 | 1.462876e-4 |

1. В следующей матрице

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Максимальная норма |
|  | 63 | 1.188753e-2 |
|  | 80 | 2.157432e-3 |
|  | 85 | 1.867412e-4 |

1. В следующей матрице

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Максимальная норма |
|  | 2 | 1.452746e-2 |
|  | 3 | 2.128964e-3 |
|  | 14 | 1.435687e-5 |

## Вывод

В реализации метода Якоби выясняется, что для плохо отделимых собственных чисел метод работает быстрее, но дает меньшую точность в сравнении с хорошо отделимыми собственными числами.

Замечено, что число обусловленности матрицы почти не влияет на количество итераций и точность для получения решения => метод устойчив.

Чем меньше отличаются между собой собственные числа (чем они менее отделимы), тем быстрее сходится метод.

Минусы метода состоят в том, что возможно использовать только для вещественных симметричных матриц, а также в цене за один шаг в O(n3).