Санкт-Петербургский государственный политехнический университет

Кафедра “Прикладная математика”

Отчет по курсовой работе по дисциплине

“Численные методы”

Отчет по курсовой работе по дисциплине

“Алгоритмы и структуры данных”

# Отчёт

# по лабораторной работе «Решение алгебраических и трансцендентных уравнений»(методы половинного деления и хорд)

## Формулировка задачи и её формализация

Решить 2 уравнения (трансцендентное и алгебраическое) с помощью двух методов: метода половинного деления и метода хорд. Для алгебраического уравнения аналитически найти промежутки для его корней по теореме о верхней границе. Средствами пакета MATLAB реализовать графическую интерпретацию для одного из методов (3-4 итерации). Сравнить результаты вычислений с результатами, полученными с помощью fzero. Исследовать влияние заданной точности вычислений и начального приближения на число итераций для двух уравнений.

Алгебраическое уравнение:

Трансцендентное уравнение:

Определение корня: называется корнем уравнения, если при подстановке обращает уравнение в тождество.

## Алгоритмы методов

### Метод половинного деления

Если знаки на концах промежутка различны, то на промежутке есть корень.

Блок-схема метода:

C =

f(c) = 0

f(c) ≠ 0

b = c;

a = c;

f(a)\*f(c)<0

нет

да

нет

да

Print c

|b – a| < ε

Таким образом, , где k – число итераций.

В случае, когда два корня попадают в один отрезок:

*1)*

*2)*

*3)* ,

тогда делим отрезок на N частей длиной , и к левой границе отрезка прибавляем поочередно длины частей и проверяем на двух образовавшихся отрезках выполнение условий метода половинного деления.

### Метод хорд

Рассмотрим отрезок [a; b]. Если

1)

*2)*

*3)*

*4)*

то тогда можно выделить две точки: неподвижную и нестационарную

Тогда повторяем итерацию: до тех пор, пока погрешность находится вне допустимых пределов.

Оценкой погрешности в этом случае будет:

, где .

В случае, когда два корня попадают в один отрезок:

*1)*

*2)*

*3)* ,

тогда делим отрезок на N частей длиной , и к левой границе отрезка прибавляем поочередно длины частей и проверяем на двух образовавшихся отрезках выполнение условий метода половинного деления.

## Предварительный анализ задачи

### Алгебраическое уравнение

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

1. Нет отрицательных коэффициентов в уравнении, поэтому нет положительных корней.
2. Для верхней границы отрицательных корней сделаем замену тогда – максимальный по модулю отрицательный коэффициент; следовательно, – формула для вычисления границы. Получаем *y* = 10,4179, тогда *x* = -0.0959.
3. Для нижней границы отрицательных корней используем замену ; тогда по той же формуле получаем *y* = 10,4179; тогда *x* = -10,4179.

Интервал, на котором существуют корни, определяется как [-10,4179; -0.0959].

Существуют промежутки, на которых выполняются два условия:

Например, можно рассмотреть промежутки [-2; -1] и [-0.7; 0].

(*f(-*2) = 32.96058; *f(-*1) = -10.75461; *f*(-0.7) = -10.19094153; *f*(0) = 12.6488; корнем производной является число -0.896874, поэтому на этих промежутках первая производная знакопостоянна).

На них выполняются эти условия, следовательно, на каждом промежутке существуют по одному единственному корню.

### Трансцендентное уравнение

Рассмотрим производную: ; корнями данной производной являются *x* = 2πn.

Рассмотрим промежуток [1;2]: *f*(1) = -0.091; *f*(2) =0.8407; корни производной в этом промежутке не встречаются, поэтому на нем существует единственный корень

## Тестовый пример с ручными расчетами

### Метод половинного деления

Рассмотрим промежуток [1.1; 1.2] для трансцендентного уравнения; ε = 0.001.

Проведем несколько итераций:

1. *f*(1.1) ∙ *f*(1.2) < 0; c = > ε;
2. *f(1.15) ∙f*(1.2*) > 0;* c *=* 1.175 *>* ε;
3. *f*(1.15*) ∙f*(1.175*) < 0;* c *= >* ε;

Таких итераций должно быть 7; корень с точностью 0.0029.

### Метод хорд

На промежутке [-1.1; -1] рассмотрим итерации метода хорд для полинома.

Выберем стационарную точку: *f* ’’(-1.1) ∙ *f*(-1.1) = 0.4088 > 0, поэтому выбираем эту точку за стационарную и

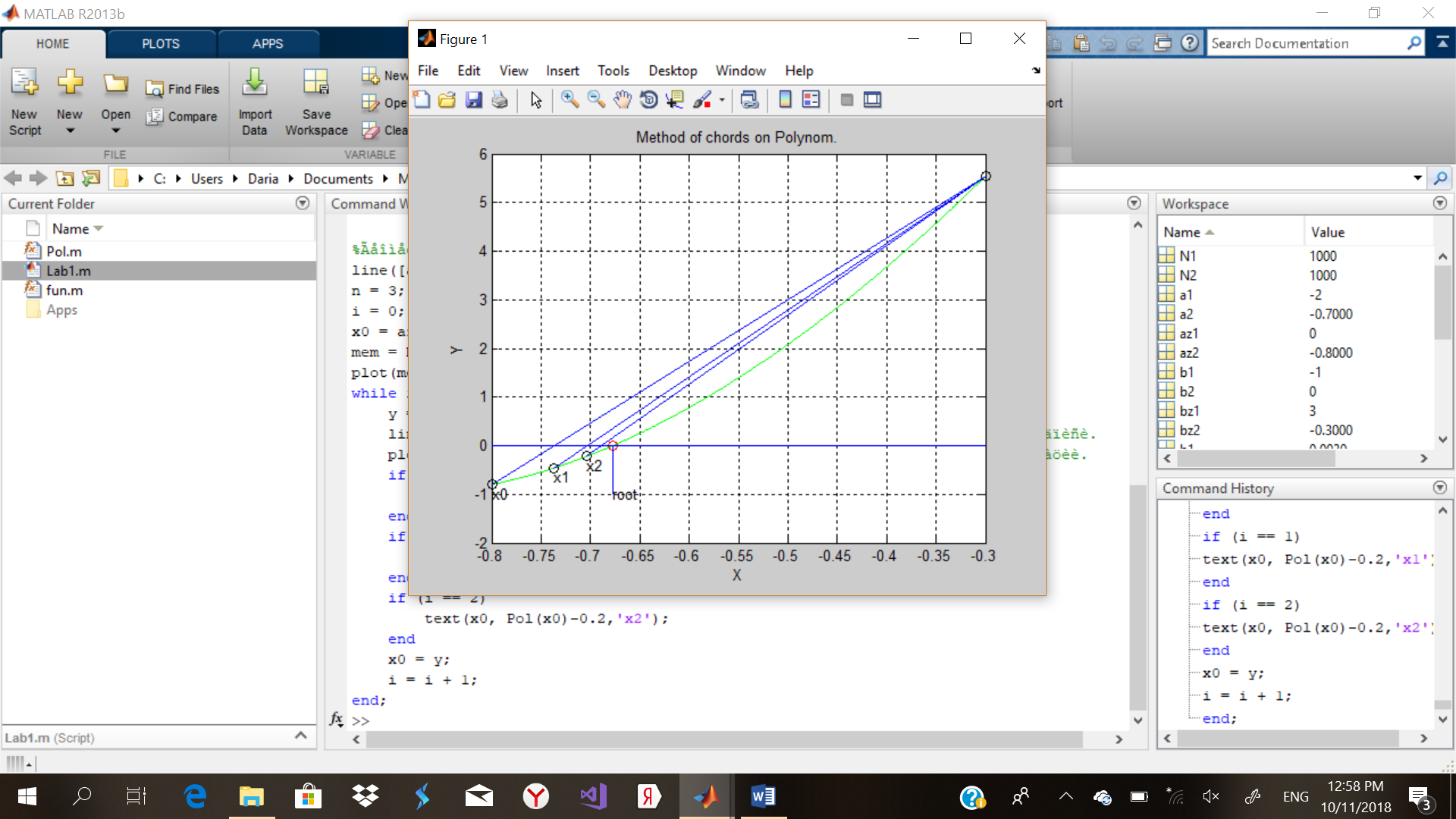
1. = -1.0986;
2. = -1.09903;

После этой итерации проверка |-1.1 – x|> ε ∙ не выполняется.

После последней итерации ε = 0.00381.

## Иллюстрация в MATLAB

Проиллюстрируем один из методов, метод хорд, при работе с полиномом (2 итерации). Отрезок, выбранный для иллюстрации, удовлетворяет условиям метода.



Проверим также получение корней с помощью функции fzero.

a1=-2; %Define borders of interval.

b1=-1;

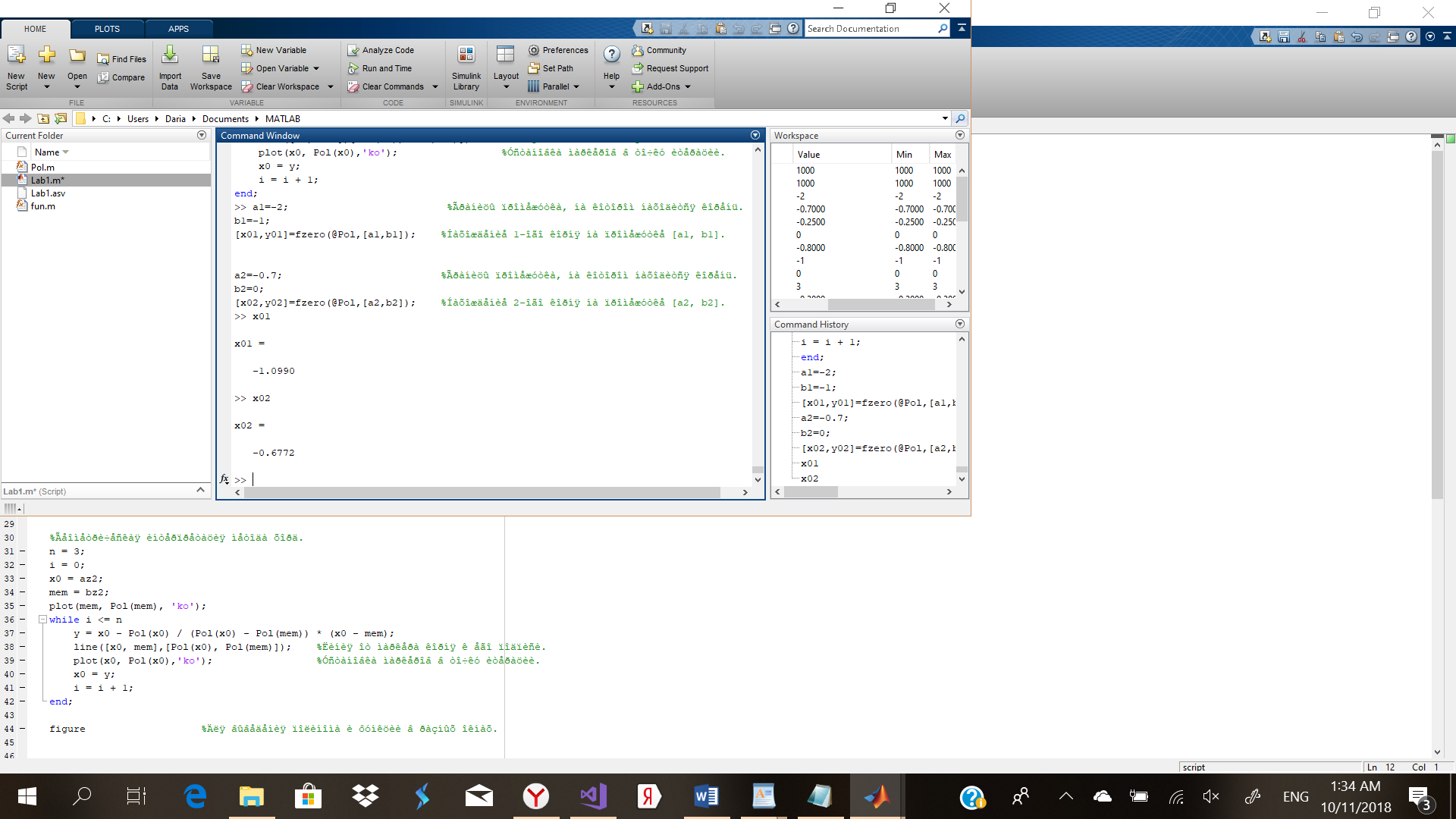
[x01,y01]=fzero(@Pol,[a1,b1]); %Find root on interval [a1, b1] with function "fzero".

a2=-0.7; %Define borders of the second interval.

b2=0;

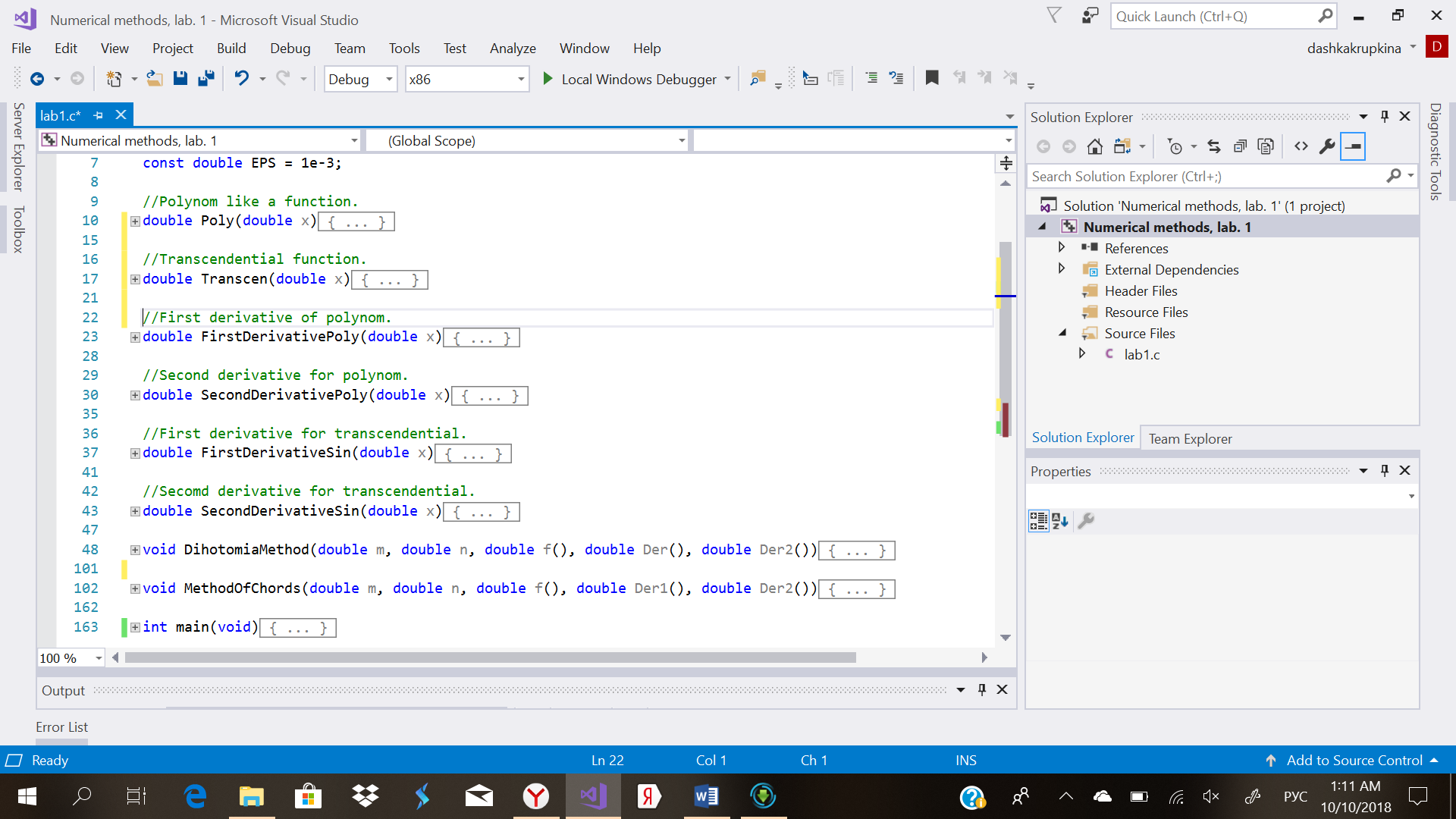
[x02,y02]=fzero(@Pol,[a2,b2]); %Find root on interval [a2, b2] with function "fzero".

Корни на этих двух промежутках получаются такими(можно сравнить с результатами вычисления в СИ в таблице ниже):



Аналогично для трансцендентного уравнения: на отрезке [1;2] получаем корень 1.1712.

## Модульная структура программы



Poly() и Transcen(): input – x, output -f(x) (value of functions) .

DihotomiaMethod(): input – borders m, n; function and its derivatives(1st and 2nd).

MethodOfChords(): input – borders m, n; function and its derivatives(1st and 2nd).

FirstDerivativePoly() && FirstDerivativeSin: input – x, output – value of 1st derivative.

SecondDerivativePoly() && SecondDerivativeSin: input – x, output – value of 2nd derivative.

## Численный анализ решения задачи

В таблице представлены зависимости точности корней и числа итераций от ε(задана заранее в качестве константы).

### Полином.

Корень из промежутка [-1.1; -1]: -1.09745.

|  |  |  |
| --- | --- | --- |
| ε | Метод половинного деления. | Метод хорд. |
|  | 4 | 2 |
|  | 7 | 2 |
|  | 14 | 3 |

Корень из промежутка [-0.7; -0.6]: -0.6772.

|  |  |  |
| --- | --- | --- |
| ε | Метод половинного деления. | Метод хорд. |
|  | 4 | 2 |
|  | 6 | 3 |
|  | 14 | 5 |

### Трансцендентное уравнение.

Корень из промежутка [1.1; 1.2]: 1.1723.

|  |  |  |
| --- | --- | --- |
| ε | Метод половинного деления. | Метод хорд. |
|  | 4 | 2 |
|  | 7 | 2 |
|  | 13 | 3 |

## Выводы

В моем случае с увеличением точности метод хорд лучше метода половинного деления по числу итераций. Метод хорд больше подходит для данного трансцендентного уравнения (больше крутизна графика, поэтому проще попасть в точку при построении хорд), метод половинного деления одинаково работает и с данным полиномом, и с трансцендентной функцией.

# Отчёт

# по лабораторной работе «Решение СЛАУ прямыми методами»(метод квадратного корня)

## Формулировка задачи и её формализация

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

## Алгоритм метода

Матрица симметрична, т.е. , и .

Тогда можно представить в виде: , где:

S = – верхняя треугольная матрица.

D = , где

Тогда общие формулы для метода:

С использованием матрицы D исключена возможность появления мнимых коэффициентов в матрице S.

## Предварительный анализ задачи и подготовка матрицы

При построении в MATLAB по формулам:

*D = - диагональная матрица.*

Уже учитываем условия применимости.

## Тестовый пример с ручными расчетами

*A = , b = .*



Тогда

S = , D = .

В данном случае поэтому:

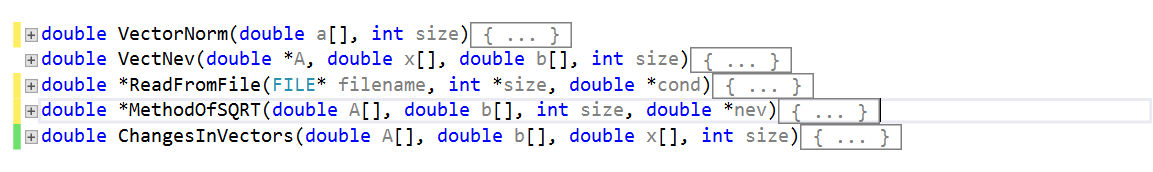
*;*

1. *.*
2. *.*

*;*



## Модульная структура программы

**

double \*ReadFromFile(FILE\* filename, int \*size, double \*cond):

принимает: имя файла, указатели на размер матрицы и число обусловленности(они находятся в файле, при чтении файла записываются в эти переменные и сохраняются на время работы программы); возвращает: указатель на матрицу, выводит её на экран.

double \*MethodOfSQRT(double A[], double b[], int size, double \*nev):

принимает: матрицы А и b, размер матрицы А (количество строк в столбце b), указатель на вектор невязки, который считается в MethodOfSQRT; возвращает: указатель на столбец решений x.

double VectNev(double \*A, double x[], double b[], int size):

принимает: матрицы A, b, x и их размер (количество строк в столбцах b и x); возвращает вектор невязки.

double VectorNorm(double a[], int size):

принимает: вектор и его размер; возвращает: его вторую норму.

double ChangesInVectors(double A[], double b[], double x[], int size):

принимает: матрицы A, b, x и их размер (количество строк в столбцах b и x); возвращает коэффициенты возмущений и выводит их на экран.

## Численный анализ решения задачи

Известно, что:

Вычислим тогда:

Для хорошо обусловленной матрицы:

Cond(A) = 3;

Для плохо обусловленной матрицы:

Cond(A) = 43243200;

## Выводы

В моём случае для любой матрицы из двух коэффициентов большим был тот, который был получен при внесении возмущений в матрицу А. Полученные коэффициенты не превосходят числа обусловленности матриц. При этом остаётся примерно одного порядка для любых коэффициентов матрицы А и b, а зависит от числа обусловленности, но при этом отличается от него на несколько порядков. приближается в своем значении к числу обусловленности при большой разнице в порядках между A и b.

# Отчёт

# по лабораторной работе «Решение СЛАУ итерационными методами»(метод Зейделя)

## Формулировка задачи и её формализация

Решение СЛАУ итерационными методом: методом Зейделя. Исследовать зависимость точности решения и количества итераций в случае, когда определитель матрицы близок к 0.

## Алгоритм метода

Иначе возможно представить:

Для всех

## Предварительный анализ задачи и подготовка матрицы

1. Для сходимости метода для произвольной матрицы А достаточно |||| < 1.

В ином случае матрицу С также можно получить из представления А в виде :

, ,

Тогда:

В качестве выберем .

В таком случае матрица удовлетворяет условиям сходимости.

Завершается итерационный процесс в случае(:

1. Если же и положительно определена, то метод Зейделя сходится в любом случае. Тогда нет необходимости проверять норму матрицы С.

Эти условия достигаются при домножении на : .

В таком случае условие выхода из итерационного процесса: || < .

1. Для получения матрицы с определителем, близким к 0, воспользуемся генератором из лаб. 2:

*D = - диагональная матрица, где – малые числа, стремящиеся к 0.*

## Тестовый пример с ручными расчетами

*, b =*

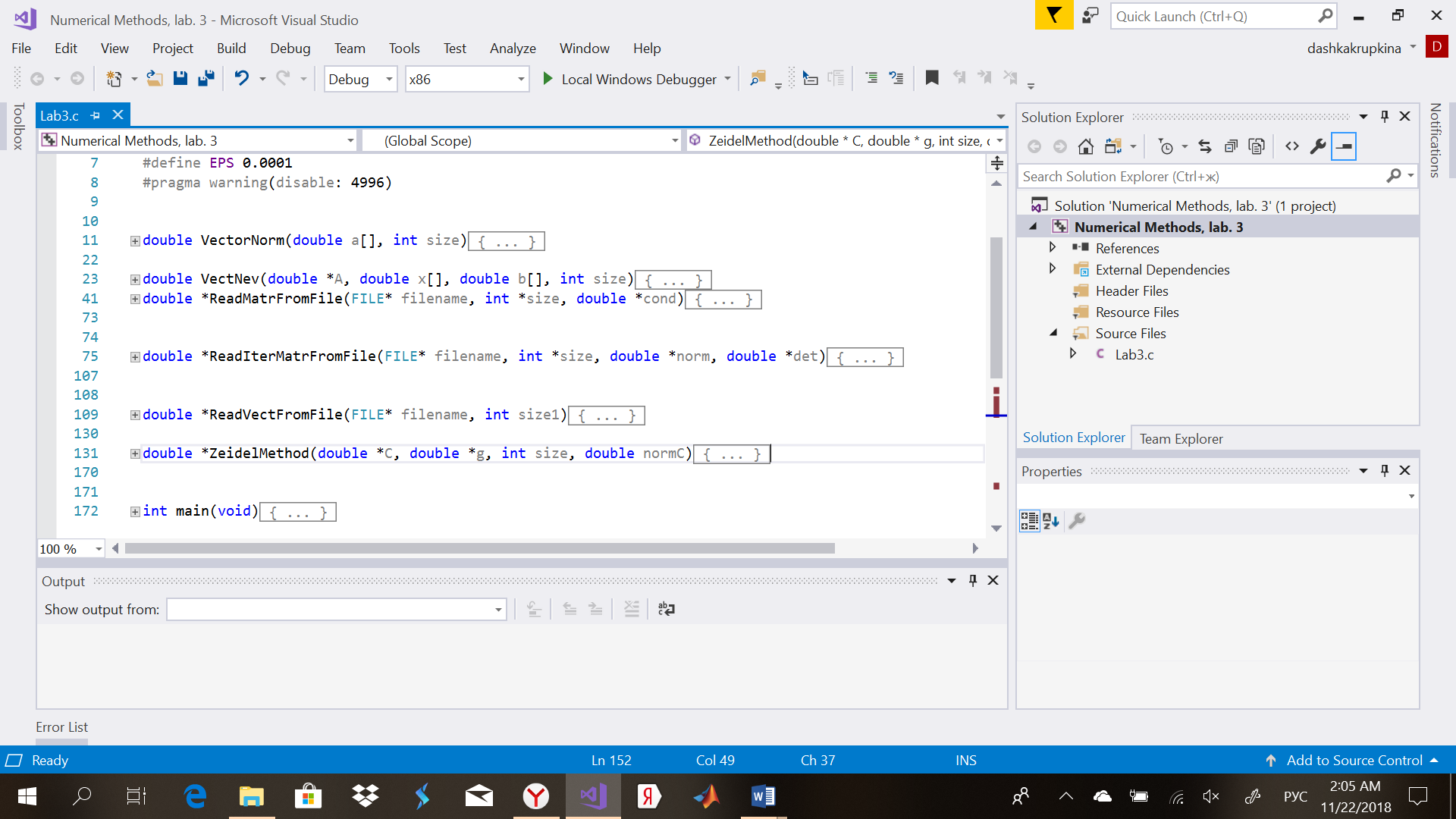
Приведем к виду, удобному для итераций(здесь СЛАУ симметричная):

, *g = (рассчитаем с точностью, стремящейся к 0, т.к СЛАУ элементарна).*

*= , = , = .*

Получим точное решение; 1.4 и 0.2 за 3 итерации*.*

## Модульная структура программы



double VectorNorm(double a[], int size):

принимает: вектор и его размер;

возвращает: его вторую норму.

double VectNev(double \*A, double x[], double b[], int size):

принимает: матрицы A, b, x и их размер (количество строк в столбцах b и x); возвращает вектор невязки.

double \*ReadMatrFromFile(FILE\* filename, int \*size, double \*cond):

принимает: имя файла, указатели на размер матрицы и число обусловленности(они находятся в файле, при чтении файла записываются в эти переменные и сохраняются на время работы программы);

возвращает: указатель на матрицу(исходную), выводит её на экран.

double \* ReadIterMatrFromFile (FILE\* filename, int \*size, double \*norm, double \*det):

принимает: имя файла, указатели на размер матрицы в виде, удобном для итераций(формируется в Matlab), норму матрицы и её определитель(они находятся в файле, при чтении файла записываются в эти переменные и сохраняются на время работы программы);

возвращает: указатель на матрицу(исходную), выводит её на экран.

double \* ReadVectFromFile (FILE\* filename, int size1):

принимает: имя файла, указатели на размер вектора(формируется в Matlab)(он находится в файле, при чтении файла записывается в эту переменную и сохраняется на время работы программы);

возвращает: указатель на вектор, выводит его на экран.

double \*ZeidelMethod(double \*C, double \*g, int size, double normC):

принимает: матрицы C и g, размер матрицы C (количество строк в столбце g), норму матрицы С;

возвращает: указатель на столбец решений x.

## Численный анализ решения задачи

Рассмотрим зависимость количества итераций от определителя матрицы(рассмотрим две матрицы размерности 10×10: с малым и большим определителем).

Det(A) = 9165.000000.

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Норма вектора невязки. |
|  | 29 | 0.027163 |
|  | 35 | 0.0026869 |
|  | 42 | 0.00046840 |

Det(A) = 2.0000e-06.

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Норма вектора невязки. |
|  | 34 | 2.456398e-05 |
|  | 41 | 3.604881e-06 |
|  | 48 | 3.166118e-06 |

Сравним теперь результаты для малых определителей в зависимости от их порядка(зафиксируем ε = ):

|  |  |  |
| --- | --- | --- |
| Определитель. | Число итераций. | Норма вектора невязки. |
| 1.1719e-05 | 29 | 3.087407e-05 |
| 1.1111e-07 | 62 | 1.870387e-05 |
| 2.2500e-09 | 139 | 1.825065e-05 |

## Вывод

В моем случае метод Зейделя работает ощутимо медленнее при определителе, стремящемся к 0, при этом точность работы метода возрастает при уменьшении определителя.

Также стоит отметить, что число итераций полностью определяется порядком малости определителя, а порядок нормы вектора невязки скорее связан с точностью ε.

# Отчет по лабораторной работе «Решение алгебраической проблемы собственных значений»(метод Якоби)

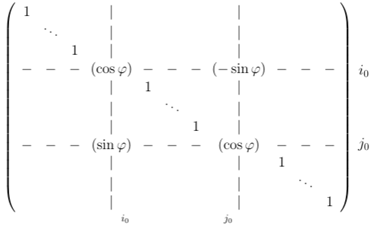
## Формулировка задачи и её формализация

Необходимо решить алгебраическую проблему собственных значений итерационным методом Якоби, а также исследовать сходимость и количество итераций при хорошей и плохой отделимости искомых собственных чисел.

## Алгоритм метода

Для этого метода: *A* = *A* – вещественная.

1. *.*
2. *Угол поворота:*
3. *Матрица вращения:*

**

*j-я строка*

*i-я строка*

*Как только выполняется , итерационный процесс останавливается.*

*Тогда собственные числа:*

*Cобственные вектора:*

## Предварительный анализ задачи и подготовка матрицы

Для получения симметричной матрицы используем Matlab:

*D = - диагональная матрица, где – собственные числа ,которые должны быть получены в результате метода Якоби.*

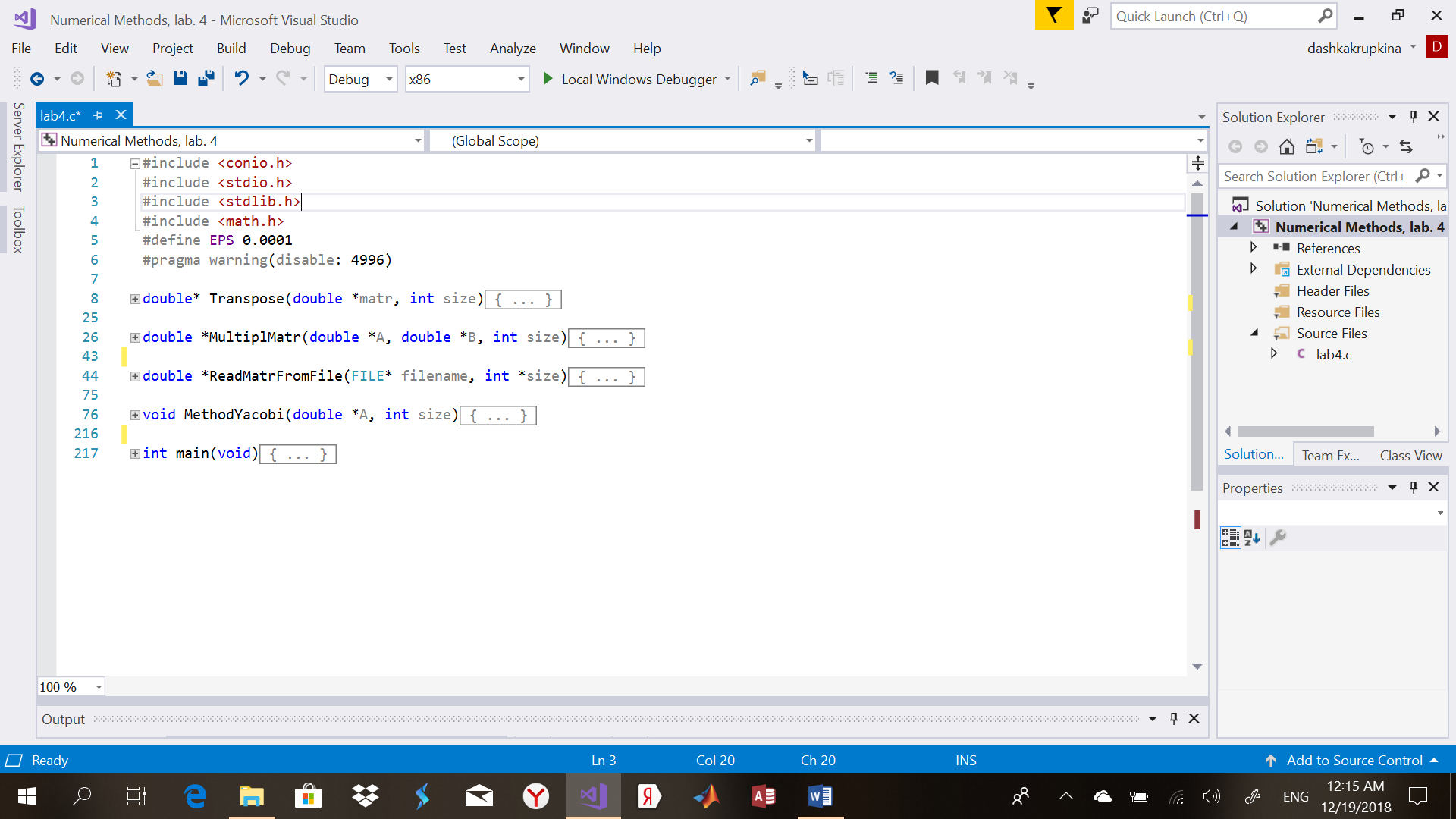
## Тестовый пример с ручными расчетами

,

1. Максимальный недиагональный элемент 1. Тогда
2. Тогда

1. На следующей итерации
2. Тогда

## Модульная структура программы



double \*ReadMatrFromFile(FILE\* filename, int \*size):

принимает: имя файла, указатели на размер матрицы и число обусловленности(они находятся в файле, при чтении файла записываются в эти переменные и сохраняются на время работы программы);

возвращает: указатель на матрицу(исходную), выводит её на экран.

double \*Transpose(double \*matr, int size):

принимает: указатель на матрицу, размер матрицы;  
возвращает: указатель на транспонированную матрицу.

double \*MultiplMatr(double \*A, double \*B, int size):

принимает: указатели на две матрицы и их размер(одинаков по построению метода);  
возвращает: указатель на матрицу, являющуюся произведением двух данных.

void MethodYacobi(double \*A, int size):

принимает: указатель на матрицу, размер матрицы;  
в процессе выполнения: находит и выводит на экран значения собственных чисел и собственных векторов матрицы , а также находит нормы вектора , где и – собственное число и собственный вектор, соответствуюшие друг другу. Также выводит на экран количество итераций.

## Численный анализ решения задачи

Рассмотрим зависимость количества итераций от того, насколько хорошо собственные числа матрицы отделимы друг от друга, а также от числа обусловленности матрицы. Для этого рассмотрим две матрицы(размерности 10×10).

Рассмотрим для начала матрицы с хорошим числом обусловленности.

В данной матрице собственные числа хорошо отделимы друг от друга, т.е , где i ≠j.

Cond(A) = 6.1250.

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Максимальная норма |
|  | 42 | 2.211183e-04 |
|  | 53 | 2.176976e-06 |
|  | 74 | 6.280076e-09 |

В следующей матрице , где i ≠j.

Cond(A) = 1.0010.

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Максимальная норма |
|  | 23 | 1.018547e-03 |
|  | 49 | 1.012328e-05 |
|  | 64 | 1.790073e-08 |

Теперь обратимся к матрицам с плохим числом обусловленности.

В данной матрице собственные числа хорошо отделимы друг от друга, т.е , где i ≠j.

Cond(A) = 2.8000e+07.

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Максимальная норма |
|  | 48 | 1.196548e-04 |
|  | 58 | 7.757340e-06 |
|  | 76 | 7.963761e-08 |

В следующей матрице , где i ≠j.

Cond(A) = 1.0027e+07.

|  |  |  |
| --- | --- | --- |
| ε | Число итераций. | Максимальная норма |
|  | 25 | 9.483338e-04 |
|  | 51 | 8.214497e-06 |
|  | 64 | 3.640905e-08 |

Теперь исследуем количество итераций и точность для разной степени отделимости собственных чисел.

Зафиксируем ε = .

|  |  |  |
| --- | --- | --- |
| . | Число итераций. | Норма вектора невязки. |
|  | 53 | 9.852393e-08 |
|  | 21 | 1.006391e-07 |
|  | 7 | 7.286665e-07 |

## Вывод

В моей реализации метод Якоби дает такой результат: для плохо отделимых собственных чисел метод работает быстрее, однако дает худшую точность в сравнении с хорошо отделимыми собственными числами.

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

Также стоит отметить, что метод достаточно устойчив, т.к даёт практически одинаковые по точности и числу итераций результаты для матриц с различными числами обусловленности.

Существенным минусом метода является то, что он действует лишь для симметричных матриц.