Санкт-Петербургский государственный

политехнический университет

Кафедра “Прикладная математика”

Отчет по курсовой работе по дисциплине

“Численные методы”

Выполнила

Студентка группы № 23631/1

Королевская Ксения Дмитриевна

Руководитель

Павлова Людмила Владимировна

Санкт-Петербург 2018

Оглавление

[Часть 1. Решение алгебраических и трансцендентных уравнений. 4](#_Toc532904719)

[Формулировка задачи: 4](#_Toc532904720)

[Этапы решения: 4](#_Toc532904721)

[Алгоритмы методов и условия их применимости 4](#_Toc532904722)

[Предварительный анализ задачи 4](#_Toc532904723)

[Проверка условий применимости метода 5](#_Toc532904724)

[Тестовый пример 6](#_Toc532904725)

[Контрольные тесты 6](#_Toc532904726)

[Модульная структура программы 7](#_Toc532904727)

[Анализ численного решения задач 7](#_Toc532904728)

[Краткие выводы 8](#_Toc532904729)

[Часть 2. Решение СЛАУ прямыми методами 8](#_Toc532904730)

[Формулировка задачи 8](#_Toc532904731)

[Алгоритм метода и условия применимости 8](#_Toc532904732)

[Предварительный анализ задачи 8](#_Toc532904733)

[Тестовый пример 8](#_Toc532904734)

[Модульная структура программы 9](#_Toc532904735)

[Анализ численного решения задач 10](#_Toc532904736)

[Вывод 10](#_Toc532904737)

[Часть 3. Решение СЛАУ итерационными методами. 10](#_Toc532904738)

[Формулировка задачи 10](#_Toc532904739)

[Алгоритм метода и условия применимости 10](#_Toc532904740)

[Предварительный анализ задачи 11](#_Toc532904741)

[Тестовый пример 11](#_Toc532904742)

[Модульная структура программы 11](#_Toc532904743)

[Анализ численного решения задач 12](#_Toc532904744)

[Вывод 12](#_Toc532904745)

[Вывод по 2 и 3 части 12](#_Toc532904746)

[Часть 4. Решение алгебраической проблемы собственных значений. 12](#_Toc532904747)

[Формулировка задачи. 12](#_Toc532904748)

[Алгоритм метода и условия применимости 12](#_Toc532904749)

[Тестовый пример 13](#_Toc532904750)

[Модульная структура программы 13](#_Toc532904751)

[Анализ численного решения задач 14](#_Toc532904752)

[Вывод 14](#_Toc532904753)

[Приложение 14](#_Toc532904754)

[Часть 1. 14](#_Toc532904755)

[Часть 2. 17](#_Toc532904756)

[Часть 3. 17](#_Toc532904757)

[Часть 4. 18](#_Toc532904758)

# Часть 1. Решение алгебраических и трансцендентных уравнений.

Формулировка задачи:

- найти корни уравнения вида f(x)=0.

Этапы решения:

- нахождение границ, в которых лежат корни

- отделение корней (нахождение отрезков, содержащих один корень)

- уточнение корней с некоторой точностью

Алгоритмы методов и условия их применимости

* Метод половинного деления

Функция f(x) определена и непрерывна при всех х на отрезке [а, b], и f(a)\*f(b) <0.

Тогда уравнение будет иметь хотя бы один корень.

Алгоритм:

1. Находим середину отрезка c =
2. Если f(c)=0, то полагаем, что корень равен с. Если f(a)\*f(c) <0, полагаем b=c, иначе a=c.
3. Выполняем пункты 1-2, пока не будет достигнуто |a-b| <2 \* Ɛ

* Метод секущих

Функция f(x) определена и дважды дифференцируема на отрезке [а, b], f(a)\*f(b) <0. . <> 0, <> 0 на [а, b]. И f()\*()<0. Тогда последовательность {} сходится к корню.

Оценка погрешности: || < , где = min{||}, = max{||}.

Алгоритм:

1. Выбираем и : если f(а)\*(а)<0, то = а, = а + Ɛ, иначе = b, = b – Ɛ.
2. Вычисляем в цикле новый х по формуле: = - .
3. Используем апостериорную оценку для выхода из цикла: || < , где = min{||}, = max{||}.

Предварительный анализ задачи

Исследуем функцию f(x) = 9.2214\* – 8.9079\* + 7.7486\* – 1.3684\*х – 0.9893. Функция непрерывна, не имеет точек разрыва. = 36.8856\* – 26.7237\* + 15.4972\*х – 1.3684.

= 110.6568\* – 53.4474\*х + 15.4972. и определены и непрерывны на всей оси. У функции есть два корня в окрестностях точек -0.25 и 0.5.

Теорема о верхней границе положительных корней полинома: х < 1 + , где m – номер первого отриц. коэф-та, = . Для нахождения нижней границы пол. корня делаем замену: х = , для нахождения границ отриц. корня делаем замены: х = -y и х =- .

Полученные формулы:

a=9.2214;

b=-8.9079;

c=7.7486;

d=-1.3684;

e=-0.9893;

y=a\*x.^4+b\*x.^3+c\*x.^2+d\*x+e;

X\_min\_otr = (-1) \* ((abs(e)/a) ^ (1/4) +1);

X\_max\_otr = 1 / ((-1) \* (a/abs(e) + 1));

X\_max\_pol = abs(b)/a + 1;

X\_min\_pol = 1 / ((a/abs(e)) ^ (1/2) + 1);

Получилось, что положительный корень лежит в промежутке [0.246728;1.966003], отрицательный корень лежит в промежутке [-1.572312; -0.096889].

Для метода половинного деления: f(a)\*f(b) <0.f(0.246728) \* f(1.966003) = -91.9929 <0 и f(-1.572312)\*f(-0.096889) = -86.2652 <0.

Для метода секущих: функция дважды дифференцируема. . Также выполняется условие: для первого корня f(а)\*(а) <0 : f(0.246728)\*(0.246728)= -8.7176 <0, следовательно = а, = а + Ɛ, и для второго : f(b)\*(b) <0 : f(-1.572312)\*(-1.572312) = -15.3314 <0, следовательно = b, = b - Ɛ.

Исследуем функцию f(x) = x – 3\*. Функция непрерывна, не имеет точек разрыва.

= 1 + 6.24\*cos(1.04\*x)\*sin(1.04\*x), = -1\*3.3746\*cos(2.08\*x). Обе функции определены и непрерывны. Функция имеет три корня в окрестности точек 1, 2.75, 3.

Для метода половинного деления: функция определена и непрерывна при всех х на промежутках [0.8;1.5], [2.3;2.8], [2.9;3.5]. Также выполняется условие: f(a)\*f(b) <0. f(0.8) \* f(1.5) = -0.8404<0 ,

f(2.3)\*f(2.8) = -0.0309<0 и f(2.9) \* f(3.5) = -0.0627<0.

Для метода секущих: функция дважды дифференцируема. и . Также для первого промежутка: f(a)\*(a) <0 : f(0.8)\*(0.8) = -0.1760<0, для второго промежутка: f(a)\*(a) <0 : f(2.3)\*(2.3) = -0.1672<0, для третьего промежутка: f(b)\*(b) <0 : f(3.5)\*(3.5) = -2.1723<0, следовательно, для третьего промежутка = b, = b – Ɛ, а для первого и второго = а, = а + Ɛ.

Проверка условий применимости метода

Произведем проверку условий метода половинного деления:

1) Нарушим условие непрерывности. Рассмотрим функцию 1/𝑥=0 на промежутке [-1; 1]. Метод дал результат - -2.2204e-016. Полученное решение соответствует точке разрыва функции.

2) Нарушим условие 𝑓(𝑎)∗𝑓(𝑏)<0 и наличие корней. Для этого возьмем функцию для +1=0 на промежутке [-1; 1]. Данная функция определена и непрерывна при всех х, принадлежащих отрезку [-1, 1], а также не имеет корней на всей области определения. В этом случае метод не даст никакого ответа, поскольку программа зациклится.

3) Нарушим условие единственности корня на отрезке. Рассмотрим функцию + −2∗𝑥−1=0 на отрезке

[-2, 2] (условие 𝑓(𝑎)∗𝑓(𝑏)<0 выполняется). Метод даст результат 1.2470, что является одним из трех корней уравнения на данном отрезке.

Произведем проверку условий метода Секущих:

1) Нарушим условие 𝑓(𝑎)∗𝑓(𝑏) <0 и наличие корней. Рассмотрим +1=0 на промежутке [0; 4]. В этом случае ответ будет равняться 0.3274 с количеством итераций i = 36446846, хотя у функции и нет корней.

2) Нарушим условие существования производных. Рассмотрим 1/𝑥=0 на промежутке [-1; 1], программа зациклилась и не дала результата.

3) Нарушим знакопостоянство первой производной. Рассмотрим −2∗𝑥−0.5=0 на промежутке [-1; 0.5]. Метод даст правильный ответ -0.2587.

4) Нарушим знакопостоянство второй производной. Рассмотрим −2∗𝑥−0.5=0 на промежутке [-0.6; 0.5]. Метод даст правильный ответ -0.2587.

5) Нарушим единственность корня. Возьмем функцию sin(𝑥)=0 на отрезке [−𝜋/2,3𝜋/2] (условие 𝑓(𝑎)∗𝑓(𝑏) <0 не выполняется). Программа даст ответ 0, то есть мы нашли один правильный корень из двух существующих. Рассмотрим 𝑦=𝑠𝑖𝑛(𝑥) на отрезке [−3𝜋/2,3𝜋/2] (условие 𝑓(𝑎)∗𝑓(𝑏) <0 выполняется). Метод даст результат 0, что является одним из двух корней уравнения на данном отрезке.

6)Пусть первые две точки не лежат по одну сторону от корня. Возьмем функцию 3\*-3 =0 на промежутке [0;3]

И первые две точки равны 0 и 3 соответственно. Метод даст правильный корень 1.

Тестовый пример

Метод половинного деления для трансцендентной функции у = x – 3\* на отрезке [0.8;1.5].

1. с==1.15, f(a)\*f(c) = -0.4191 < 0, следовательно, b=c=1.15
2. с==0.975 , f(a)\*f(c) = -0.0769 < 0, следовательно, b=c=0.975
3. с==0.8875 , f(a)\*f(c) = 0.1148> 0, следовательно, a=c=0.8875
4. с==0.93125 , f(a)\*f(c) = 0.0065> 0, следовательно, a=c= 0.93125
5. Продолжаем алгоритм, пока не дойдем до корня, удовлетворяющего заданной точности.

Метод Секущих для трансцендентной функции у = x – 3\* на отрезке [0.8;1.5].

1. = 0.8, = 0.801
2. = 0.999 - =0.9375
3. = 0.9375 - =0.9393
4. Продолжаем алгоритм, пока не дойдем до корня, удовлетворяющего заданной точности.

Контрольные тесты

Корень полинома на промежутке [0.246728;1.966003]

|  |  |  |
| --- | --- | --- |
| Точность | Кол-во итераций МПД | Кол-во итераций МС |
| 10^-3 | 11 | 7 |
| 10^-6 | 21 | 9 |
| 10^-9 | 31 | 10 |

Корень fzero: 0.55686254

Корень полинома на промежутке [-1.572312; -0.096889]

|  |  |  |
| --- | --- | --- |
| Точность | Кол-во итераций МПД | Кол-во итераций МС |
| 10^-3 | 11 | 5 |
| 10^-6 | 21 | 7 |
| 10^-9 | 31 | 8 |

Корень fzero: -0.24835190

Корень трансцендентной функции на промежутке [0.8;1.5].

|  |  |  |
| --- | --- | --- |
| Точность | Кол-во итераций МПД | Кол-во итераций МС |
| 10^-3 | 10 | 3 |
| 10^-6 | 20 | 4 |
| 10^-9 | 30 | 5 |

Корень fzero: 0.93935537

Корень трансцендентной функции на промежутке [2.3;2.8].

|  |  |  |
| --- | --- | --- |
| Точность | Кол-во итераций МПД | Кол-во итераций МС |
| 10^-3 | 10 | 5 |
| 10^-6 | 20 | 7 |
| 10^-9 | 30 | 8 |

Корень fzero: 2.72500028

Корень трансцендентной функции на промежутке [2.9;3.5].

|  |  |  |
| --- | --- | --- |
| Точность | Кол-во итераций МПД | Кол-во итераций МС |
| 10^-3 | 10 | 6 |
| 10^-6 | 20 | 8 |
| 10^-9 | 30 | 9 |

Корень fzero: 2.99837389

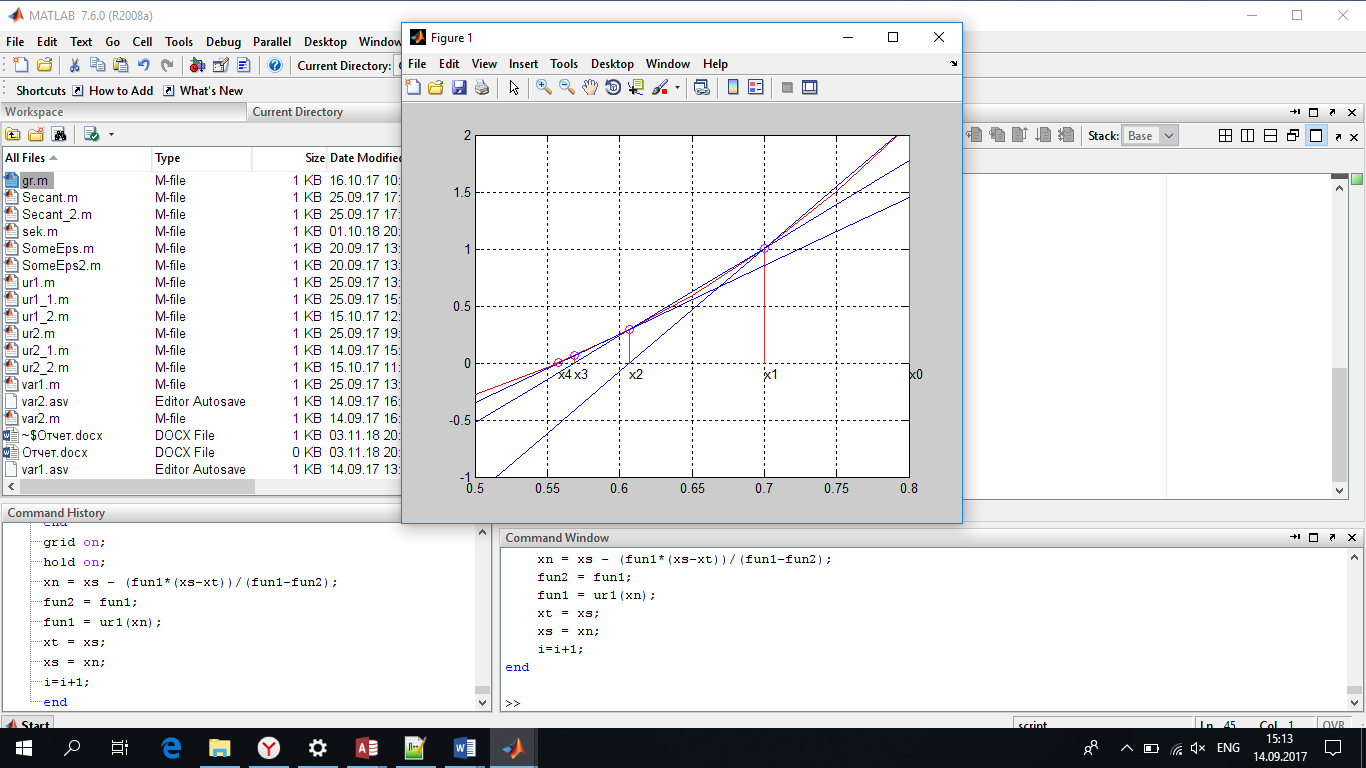
Модульная структура программы

Программа состоит из 20 модулей:

* «ur1.m», «ur2.m» - полином и трансцендентная функция соответственно.
* «ur1\_1.m», «ur1\_2.m», «ur2\_1.m», «ur2\_2.m» - первая и вторая производная соответствующих функций
* «var1.m» - модуль, реализующий метод половинного деления, рисует график обеих функций, вызывает для каждой функции на заданном промежутке «SomeEps.m» и «SomeEps2.m» соответственно для полинома и трансцендентной функции.
* «SomeEps.m», «SomeEps2.m» - функции с циклом, в котором меняется эпсилон, вызывают соответственно «FindRootsHD.m» и «FindRootsHD2.m». Также находит встроенный корень с помощью fzero. Принимают на вход границы поиска корня.
* «FindRootsHD.m», «FindRootsHD2.m» - функции, реализующие метод половинного деления, на входе промежуток, содержащий корень. На выходе- корень и количество итераций.
* «var2.m» - модуль, реализующий метод секущих, рисует график обеих функций, находит первые две точки, вызывает для каждой функции на заданном промежутке «eps1.m» и «eps2.m» соответственно для полинома и трансцендентной функции.
* «eps1.m», «eps2.m» - функции с циклом, в котором меняется эпсилон, вызывают соответственно «Secant.m» и «Secant\_2.m». На вход принимаются первые две точки, и границы поиска корня для вызова функции «Derivative.m» и «Derivative\_2.m». Вычисляется апостериорная оценка.
* «Derivative.m», «Derivative\_2.m» - функции вычисляющие минимум первой производной и максимум второй для полинома и трансцендентной функции соответственно.
* «Secant.m», «Secant\_2.m» - функции, реализующие метод секущих. На вход принимаются первые две точки и эпсилон, на выходе – последние две точки и количество итераций.
* «gr.m» - графическая интерпретация метода секущих для первых 4 итераций. Вызывает функцию «sek.m».
* «sek.m» - функция, составляющая уравнение секущей.

Анализ численного решения задач

Была поставлена задача найти корни трансцендентного и алгебраического уравнений вида 𝑓(𝑥)=0 двумя разными методами. Метод половинного деления достаточно легко реализуется и является наиболее универсальным среди итерационных методов уточнения корней. Его применение гарантирует получение решения для любой непрерывной функции 𝑓(𝑥), если найден интервал, на котором она изменяет знак. В том случае, когда корни не отделены, будет найден один из корней уравнения. Метод достаточно медленный, корень с точностью 10−9 достигается с помощью 30 итераций для трансцендентной функции, 31 итераций для полинома. Метод Секущих обладает высокой скоростью сходимости, с его помощью удалось найти корень с точностью до 10−9 всего за 5, 8 и 9 итерации для трансцендентной функции, за 10 для положительного корня полинома и за 8 для отрицательного. Таким образом, мы изучили алгоритмы этих методов для нахождения корней. Графическая интерпретаций метода секущих (4 итерации):



Краткие выводы

С помощью данной работы мы изучили два метода для нахождения корней уравнения, посмотрели на скорость их сходимости и реализовали графическую интерпретацию метода Секущих.

# Часть 2. Решение СЛАУ прямыми методами

Формулировка задачи

- найти корни СЛАУ вида Ax=b методом Квадратного Корня, сравнить решения СЛАУ с матрицами разной обусловленности.

Алгоритм метода и условия применимости

Матрица А: А = ,det(A)≠0 и (Ах, х)>0 для Ɐ х, х ≠ 0.

Алгоритм:

1. Представляем матрицу в виде А = U\*, используя формулы: = () ;

= (j, j>i); =0 (j<i)

1. Находим у из уравнения \*y = b, по формуле: = , ()
2. Находим х из уравнения U\*x = y, по формуле: = , ()

Алгоритм создания симметричной матрицы с заданным числом обусловленности:

Е ϵ Mn; w ϵ R(n);

H = Е –2\* ; - Н – матрица отражения. сond(H) = 1.

D – диагональная матрица. Тогда на главной диагонали находятся собственные числа (с.ч). Выберем максимальное и минимальное с.ч. так, чтобы их отношение было заданным по условию числом.

A = H\*R\*Hт.

Предварительный анализ задачи

Все главные миноры матриц отличны от нуля, матрицы положительно определенные.

Тестовый пример

А = ; b =

det(A) = 3\*3\*2 + 2 + 2 - 3 – 3 – 2\*2\*2 = 8.

1. = , = , =

= = , = 0, = =

= = , = 0, = 0;

1. \* U = \*
2. \* y = b

= ; = = ; = =

1. U\*x = y

= = ; = = ; = =

= -5.125; = 2.875; = 4.625

Модульная структура программы

Программа состоит из 14 модулей:

1. void Create (double A[M][M], double B[M][M]) – функция, создающая хорошо обусловленную матрицу и плохо обусловленную.
2. Int MainJob (double A[M][M], double b[M], double x[M]) – основная функция (печатает матрицу, решение, вектор невязки и т.д.)
3. int Det (double B[M][M]) – функция, считающая все главные миноры.
4. void FindMatrix (double A[M][M], double B[M][M]) – функция, которая находит матрицу U.
5. void FindY (double A[M][M], double B[M][M], double y[M], double b[M]) – функция, которая находит столбец у.
6. void FindX (double A[M][M], double B[M][M], double y[M], double b[M], double x[M]) – функция, которая находит столбец решений Х.
7. void VectorNevyazki (double A[M][M], double X[M], double b[M], double V[M]) – функция, вычисляющая вектор невязки.
8. void Mulmatrix (double A[M][M], double B[M][M], double C[M][M]) – функция, умножающая две матрицы.
9. void Trans (double A[M][M], double B[M][M]) – функция, вычисляющая транспонированную матрицу.
10. double Norma (double A[M][M]) – функция, вычисляющая норму матрицы.
11. void ReverseMatrix (double A[M][M], double AA[M][M]) – функция, находящая обратную матрицу.
12. double Cond (double A[M][M]) – функция, считающая число обусловленностей матрицы.
13. double Compare1(double x[M], double x1[M], double b[M], double b1[M],double A[M][M]) – функция, вычисляющая коэффициент к1.
14. double Compare2(double A[M][M], double A1[M][M], double x[M],double x1[M]) - функция, вычисляющая коэффициент к2.

Анализ численного решения задач

Известны неравенства: ; .

Наша задача – найти коэффициенты K1 и K2, связывающие правые и левые части этих неравенств.

Коэффициент К1 получается путем внесения возмущений в столбец b и вычисляется по формуле: =𝑘1∗ .

Коэффициент К2 получается путем внесения возмущений в матрицу А и вычисляется по формуле: =𝑘2∗ .

Полученные данные:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Число обусловленностей | Значение коэф-та | Вектор невязки |
| K1 | 3,200273 | 0,578348 | 1.8263е-07 |
| К2 | 0,649711 | 3.273956е-07 |
| К1 | 267072020,541625 | 1,391294 | 3.827109е-07 |
| К2 | 12,775601 | 1.384596е-06 |

Вывод

Коэффициент К2, полученный путем внесения возмущений в матрицу А, был всегда больше коэффициента К1.

Полученные коэффициенты должны быть всегда меньше числа обусловленностей матрицы.

Для плохо обусловленной матрицы коэффициент К2 был намного больше, чем у хорошо обусловленной.

# Часть 3. Решение СЛАУ итерационными методами.

Формулировка задачи

Найти корни СЛАУ вида Ax=b методом Зейделя, исследовать точность решения, когда определитель матрицы близок к 0.

Суть итерационного метода: нахождение по приближенному значению величины следующего приближения, которое является более точным. Метод позволяет получить значения корней системы с заданной точностью в виде предела последовательности некоторых векторов. Характер сходимости и сам факт сходимости зависит от выбора начального приближения.

Алгоритм метода и условия применимости

Условия применимости: det(A)≠0.

Для того, чтобы метод сходился нужно, чтобы система была представима в виде x = αx + β.

Суть метода: при нахождении i-ой компоненты (k+1)-го приближения сразу используются уже найденные компоненты (k+1)-го приближения с меньшими номерами. В матричной форме получается:

= L\* + U\* + β, L – нижняя треугольная матрица, U – верхняя треугольная матрица.

Критерий окончания: || - || < \*ε.

Достаточное условие сходимости: <1 ; достаточное и необходимое условие сходимости: | (α)| < 1.

Метод Зейделя всегда сходится для нормальных СЛАУ, т.е. для систем, в которых матрица А симметричная и положительно определенная. Любую невырожденную матрицу можно преобразовать к нормальной, если слева умножить ее на матрицу ( Система \*А\*х = \*b является нормальной). Но при таком преобразовании увеличивается число обусловленностей и уменьшается точность вычислений.

Алгоритм:

1. \*А\*х = \*b.
2. Преобразовать систему к виду x = αx + β: = , = 0, = .
3. Задать начальное приближение решение ( чаще всего кладут β).
4. Вычислить по формуле:
5. Повторять пункт (4), пока || - || < ε.

Предварительный анализ задачи

det(A)≠0, <1 и | (α)| < 1.

Тестовый пример

А = ; b =

1. Выразим столбец х:
2. α = , = β = ;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| k |  |  |  | |||| |
| 0 | 1.2 | 1.3 | 1.4 | - |
| 1 | 0.93 | 0.974 | 1.0192 | 0.3808 |
| 2 | 1.00068 | 0.997944 | 1.0002752 | 0.07068 |
| 3 | 1.00017808 | 0.999936864 | 0.9999770112 | 0.0019928 |
| 4 | 1.0000086125 | 1.0000005764 | 0.9999981622 | 0.0001694675 |

С точностью корни уравнения: х = .

Модульная структура программы

Программа состоит из 6 модулей:

1. double Norma (double alfa[M][M]) – считает норму матрицы.
2. void Create (double A[M][M], double b[M], double alfa[M][M], double betta[M]) – функция, создающая матрицу α и столбец β.
3. int Zeidel (double alfa[M][M], double betta[M], double x1[M]) – функция, реализующая метод Зейделя.
4. void Trans (double A[M][M], double B[M][M]) – находит транспонированную матрицу.
5. void Mulmatvec (double A[M][M], double B[M], double C[M]) – умножает матрицу на вектор.
6. void Mulmatrix (double A[M][M], double B[M][M], double C[M][M]) – умножает две матрицы.

Анализ численного решения задач

Исследовались матрицы различной размерности с разными определителями и разным числом обусловленностей.

Полученный данные:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер матрицы | Определитель | Обусловленность | Кол-во итераций | Вектор невязки |
| 3х3 | 1.3454e-008 | 23 | 60 | 7.43156е-008 |
| 3х3 | 2.1017e-023 | 7.4578e+004 | 3213 | 1.1378е-007 |
| 5х5 | 1.8590e-008 | 3.4606e+003 | 1123 | 1.514216е-006 |
| 10х10 | 6.7857e-016 | 840 | 1067 | 5.37735е-009 |
| 10х10 | 2.9471e-020 | 6.2609e+003 | 2718 | 1.149841е-007 |

Вывод

Метод Зейделя хорошо сходится для матриц с небольшим числом обусловленностей и определителем, не близким к нулю. Чем определитель ближе к нулю, тем большее количество итераций требуется, и чем больше число обусловленностей матрицы, тем больше погрешность вычислений.

Вывод по 2 и 3 части

Нами были изучены два типа методов решения СЛАУ: прямые и итерационные. Прямые методы больше направлены на точность решения и являются более трудоемкими, в то время как итерационные методы находят некоторое приближение к точному решению. Прямые методы позволяют получить решение за конечное число шагов и не зависят от выбора начального приближения, погрешность решения определяется выбранным эпсилон. Недостатками же такого метода является большое количество хранимых данных. Итерационные методы тоже позволяют найти решение с заданной степенью точности, эффективно используют оперативную память, но не гарантируют сходимость в общем случае, также область применения конкретного метода зависит от свойств сходимости. Приходим к следующему выводу: прямые методы не целесообразно применять при решении СЛАУ большой размерности, т.к. требуется большой объем памяти и велико количество арифметических операций. Итерационные методы в свою очередь подходят для систем большой размерности, потому что они требуют меньше оперативной памяти и выполнения арифметических операции, позволяют регулировать величину погрешности.

# Часть 4. Решение алгебраической проблемы собственных значений.

Формулировка задачи.

Найти собственные числа λ и соответствующие им собственные векторы X методом Якоби.

Число λ называется собственным числом матрицы А, если существует ненулевой вектор Х такой, что A\*Х = λ\*Х. При этом вектор Х называется собственным вектором матрицы А, соответствующим собственному числу λ.

Алгоритм метода и условия применимости

Метод Якоби для собственных значений – итерационный алгоритм для вычисления собственных значений и собственных векторов вещественной симметричной матрицы.

Условия применимости: А =

Алгоритм:

1. || = (Выбираем максимальный по модулю элемент)
2. :
3. =
4. Выполнять пункты 1-4, пока > ε (сумма недиагональных элементов)
5. , , … ,
6. U = …

= (,

= (,

= (.

Тестовый пример

А =

1. = 0.7854
2. =
3. =
4. = -0.6979
5. =
6. =
7. U =

Модульная структура программы

Программа состоит из 7 модулей:

1. void CreateU (double A[M][M], double U[M][M], int k, int l) – функция создает матрицу U.
2. double SumEl (double A[M][M]) – функция возвращает сумму внедиагональных элементов.
3. void Mulmatrix (double A[M][M], double B[M][M], double C[M][M]) – функция, перемножающая матрицы.
4. void Trans (double A[M][M], double B[M][M]) – функция, находящая транспонированную матрицу.
5. void Dub (double U[M][M], double U1[M][M]) – функция, дублирующая матрицу.
6. void MaxElem (double A[M][M], int x[2]) – функция, находящая положение максимального элемента.
7. void Yacobi (double A[M][M]) – функция, реализующая метод Якоби

Анализ численного решения задач

|  |  |  |  |
| --- | --- | --- | --- |
| Число обусловленностей | Отделимость собственных чисел | Вектор невязки | Кол-во итераций |
| 60.3547 | Хорошая |  | 103 |
| 6.4084e+003 | Плохая | 1.992016e-07 | 148 |
| 16 | Плохая | 1.487233e-08 | 114 |
| 2.2201e+004 | Хорошая | 3.655878e-07 | 123 |

Вывод

Метод Якоби нахождения собственных чисел и соответствующих им собственных векторов подходит для матриц с разными собственными числами. Чем хуже отделимость собственных чисел, тем большее количество итерации потребуется. Таким образом этот метод решает полную проблему собственных значений и собственных векторов для матриц малой размерности.

# Приложение

Часть 1.

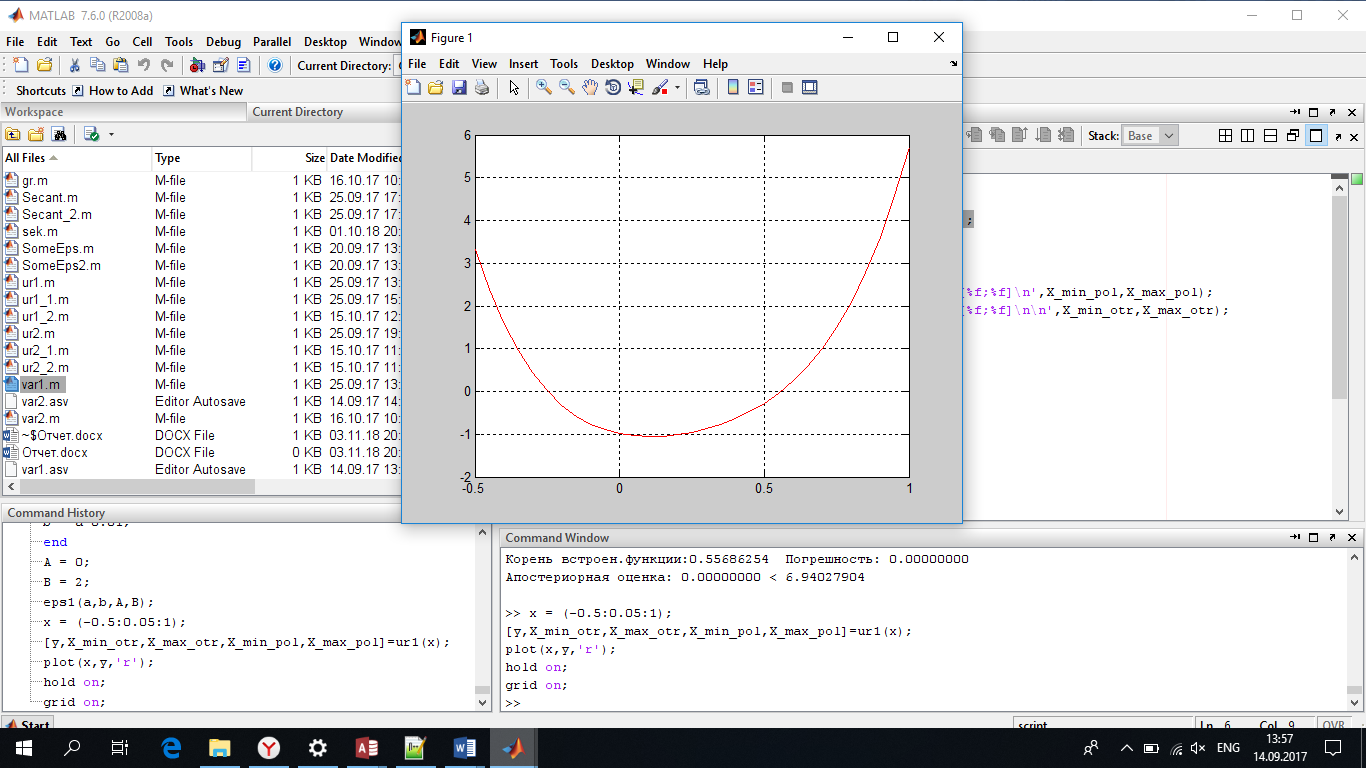
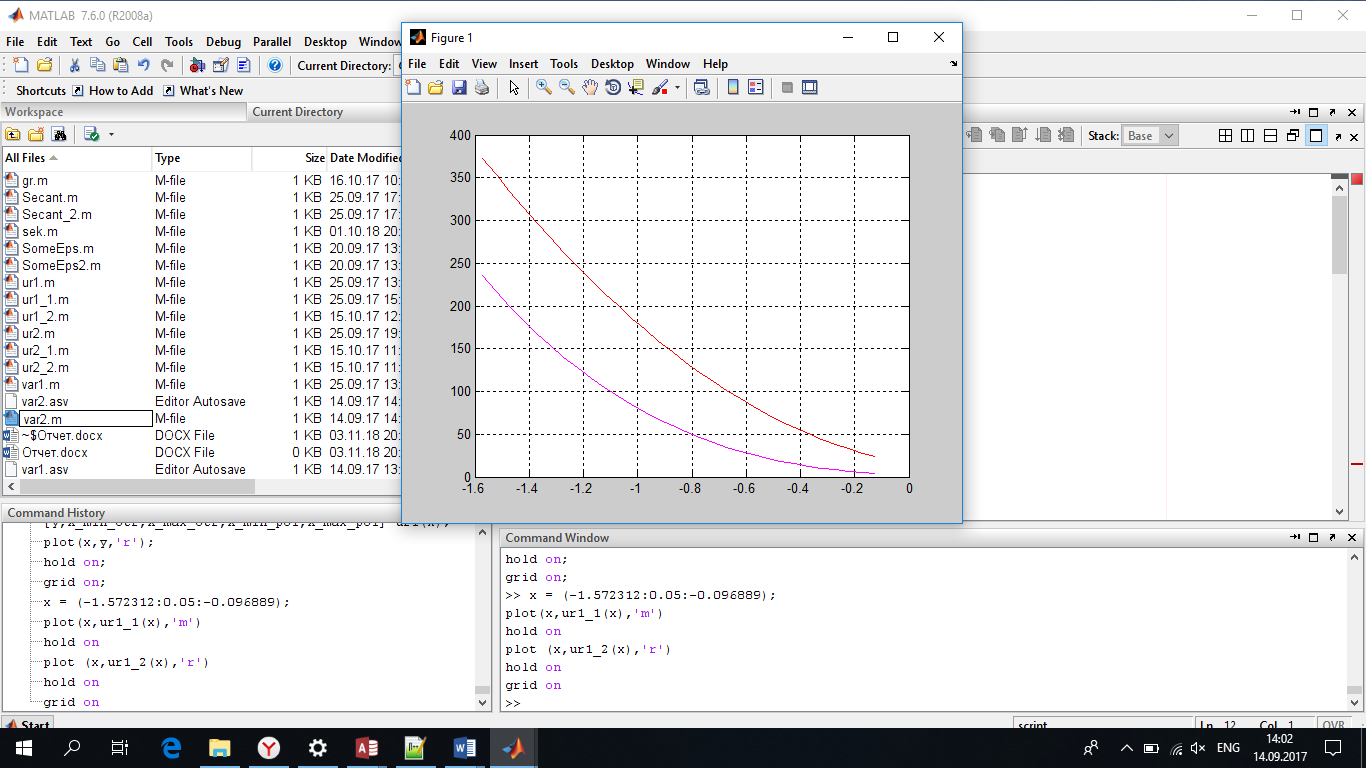
 

Рисунок 1.График полинома Рисунок 2.График первой и второй производной для . отрицательного корня полинома

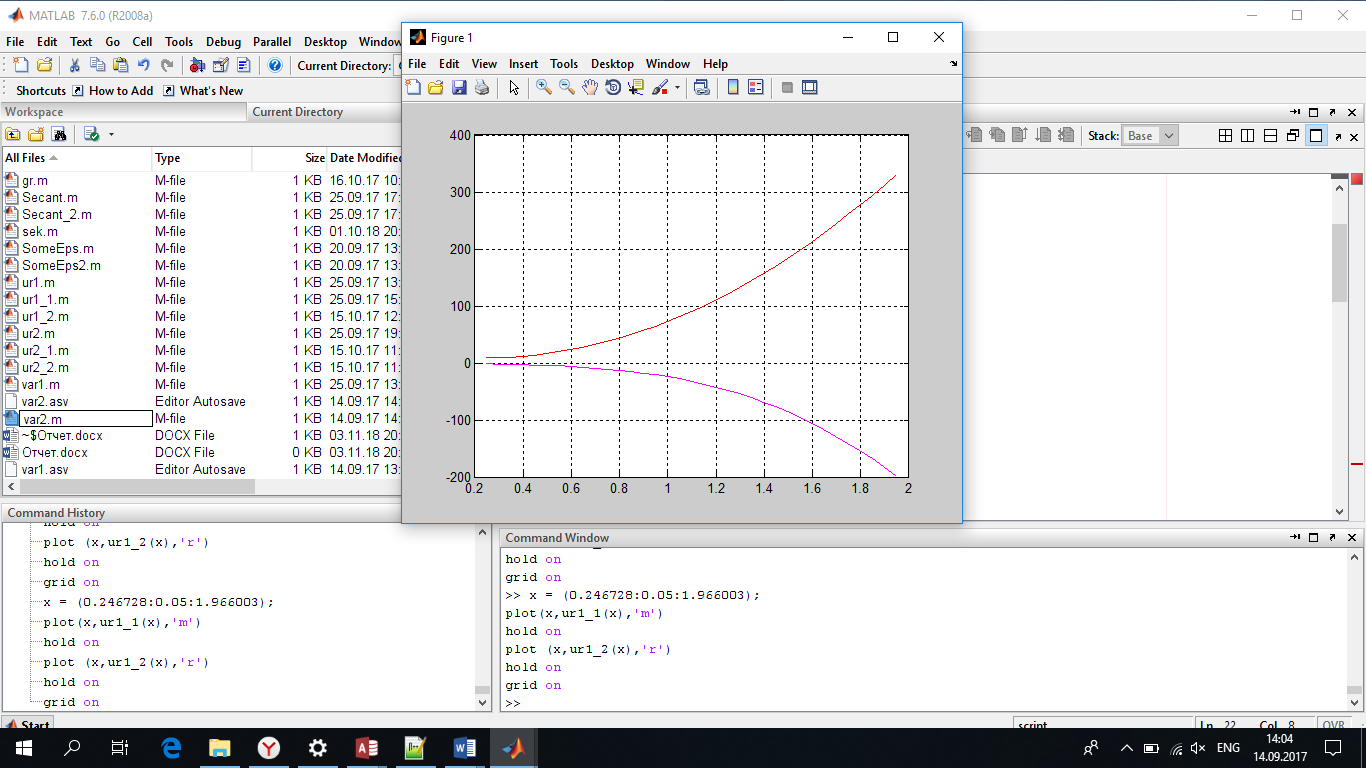
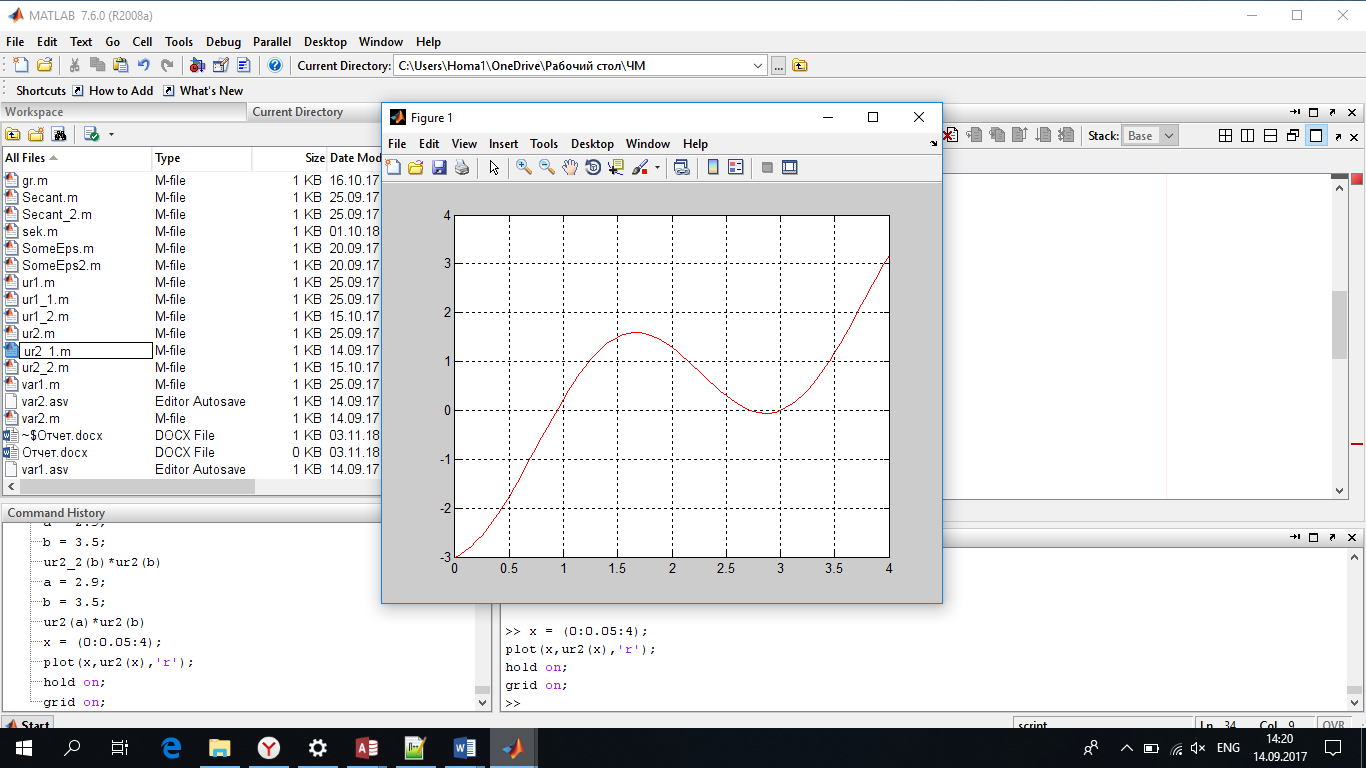
 

Рисунок 3.График первой и второй производной для Рисунок 4.График трансцендентной функции.

положительного корня полинома.

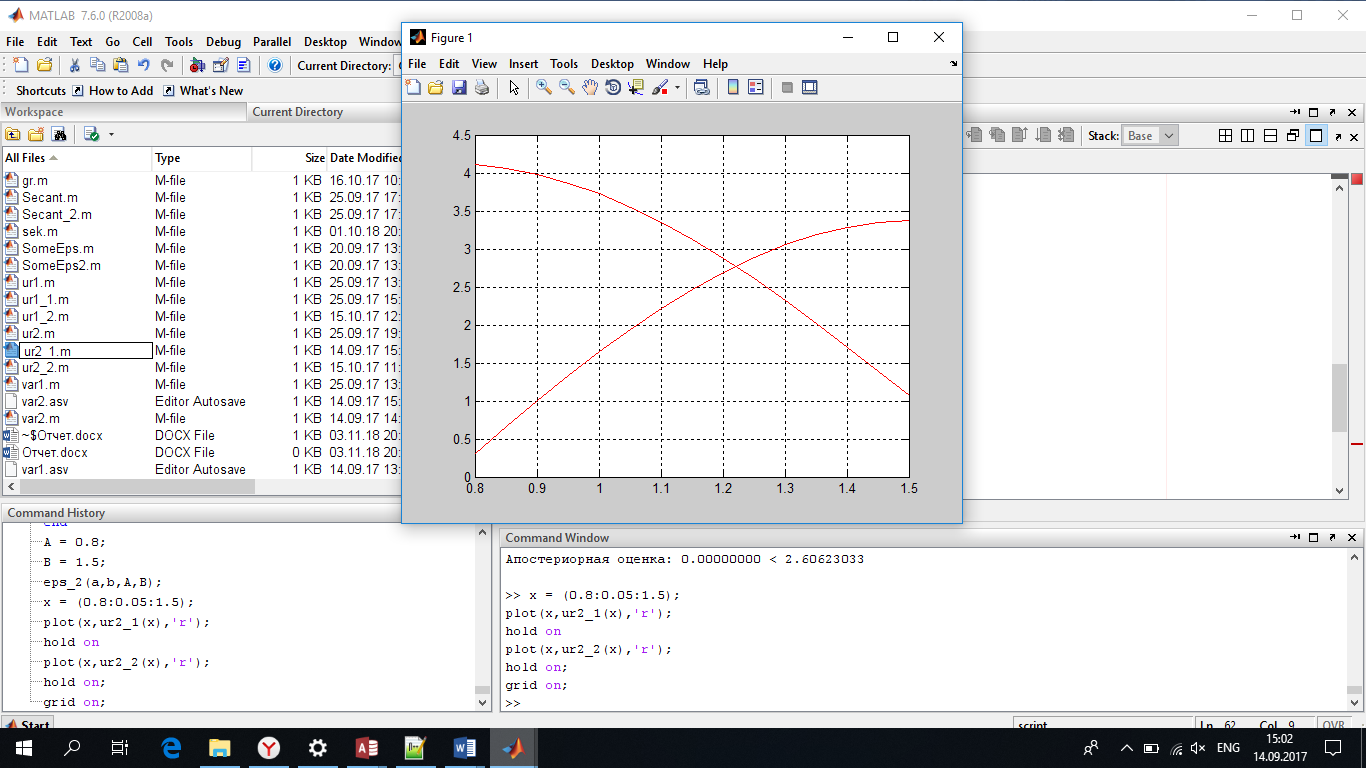
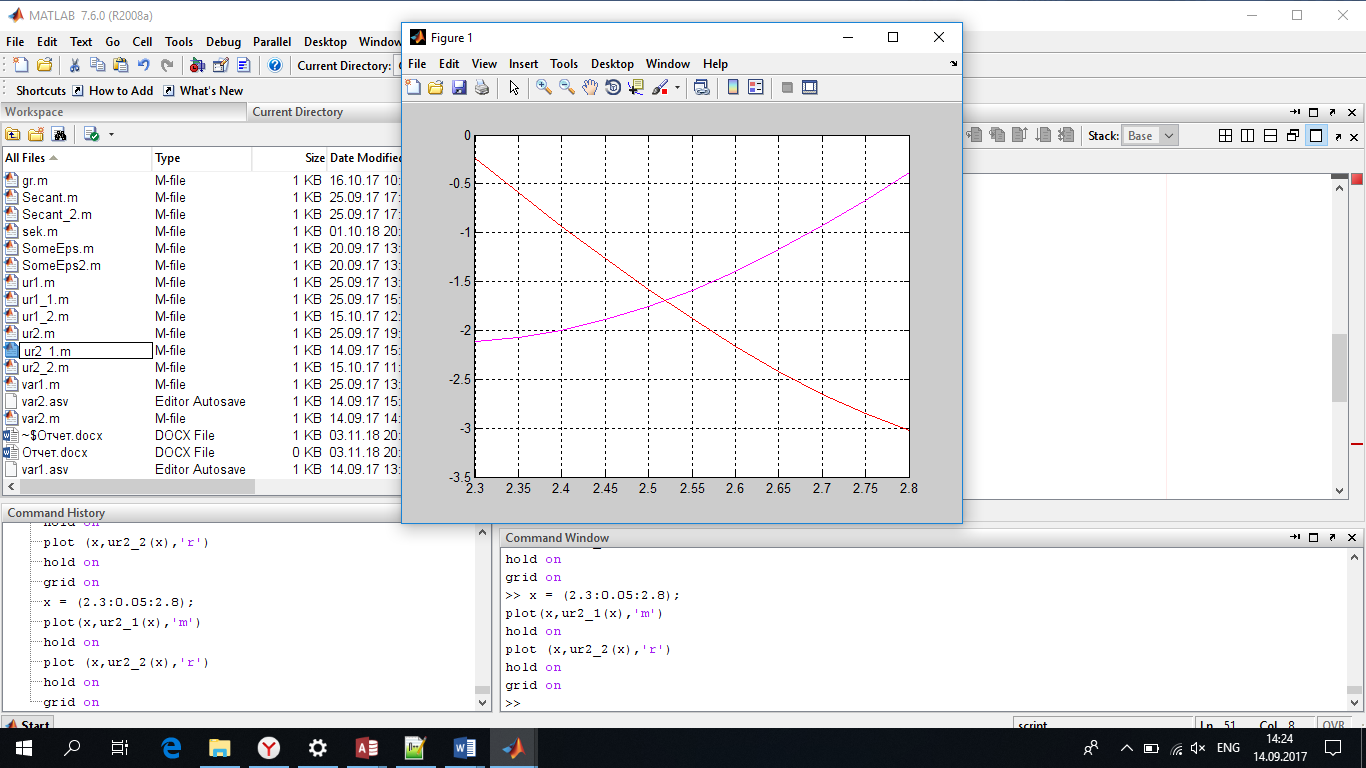
 

Рисунок 5.График первой и второй производных для Рисунок 6.График первой и второй производной для второго

первого корня корня

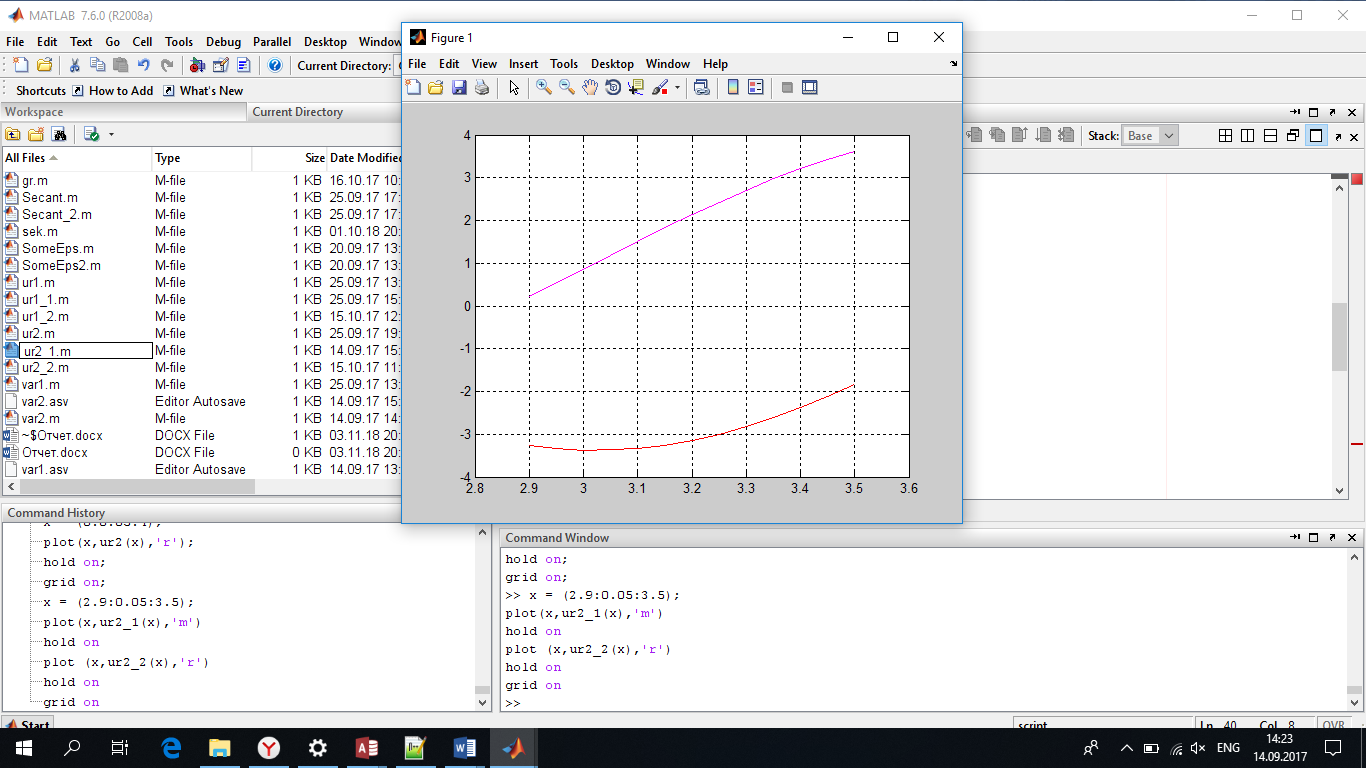


Рисунок 7.График первой и второй производной для третьего корня

Реализация метода половинного деления:

while abs(a-b)>eps

f1= (a + b)/2;

if((ur1(f1) \*ur1(a)) <0)

b=f1;

elseif (ur1(f1) \*ur1(b)<0)

a = f1;

else

a = f1;

b = a;

end

i=i+1;

end

Реализация метода секущих:

fun1 = ur1(xs);

fun2 = ur1(xt);

u = abs(xs-xt);

i=0;

while u>=epsilon

xn = xs - (fun1\*(xs-xt))/(fun1-fun2);

fun2 = fun1;

fun1 = ur1(xn);

u = abs(xn-xs);

xt = xs;

xs = xn;

i=i+1;

end

Часть 2.

Реализация метода квадратных корней решения СЛАУ:

int MainJob(double A[M][M], double b[M], double x[M])

{

double B[M][M];

double d[M], y[M];

int i, j;

printf("Матрица:\n");

for (i = 0; i < M; i++)

{

for (j = 0; j < M; j++)

printf("%lf ", A[i][j]);

printf("\n");

}

if (Det(A) < 0)

{

printf("Sorry,I can not calculate\n");

scanf("%i", &i);

return 0;

}

FindMatrix(A, B);

FindY(A, B, y, b);

FindX(A, B, y, b, x);

for (i = 0; i < M; i++)

printf("x[%i] = %lf ", i, x[i]);

printf("\n");

VectorNevyazki(A, x, b, d);

printf("Вектор невязки: ");

for (i = 0; i < M; i++)

printf("%e ", d[i]);

printf("\n\n");

return 1;

}

Часть 3.

Реализация метода Зейделя решения СЛАУ:

int Zeidel (double alfa[M][M], double betta[M], double x1[M], int param) // 0 - норма < 1, 1 - At

{

int i, j;

int pogr = 0,itr=0;

double X, q,norma;

double x2[M];

for (i = 0; i < M; i++)

x2[i] = x1[i] = betta[i];

norma = Norma(alfa);

if (param = 1)

norma = 0.5;

q = fabs(norma / (1 - norma)) \* 0.000001;

while (pogr != M)

{

pogr = 0;

for (i = 0; i < M; i++)

{

X = 0;

for (j = 0; j < M; j++)

{

X = X + alfa[i][j] \* x1[j];

}

x1[i] = X + betta[i];

if (fabs(x1[i] - x2[i]) < q)

pogr++;

}

for (i = 0; i < M; i++)

x2[i] = x1[i];

itr++;

}

return itr;

}

Часть 4.

Реализация метода Якоби нахождения собственных чисел и собственных векторов.

int Yacobi(double A[M][M],double B[M][M])

{

double U[M][M], Ut[M][M], UtA[M][M],U1[M][M],U2[M][M], u[M], Au[M];

int i=0, j=0, k = 0;

int itr = 0;

int x[2];

while (SumEl(A) > 0.000001)

{

MaxElem(A,x);

CreateU(A, U, x[0], x[1]);

Trans(U, Ut);

if (k == 0)

{

Dub(U, U1);

k++;

}

else

{

Mulmatrix(U1, U, U2);

Dub(U2, U1);

}

Mulmatrix(Ut, A, UtA);

Mulmatrix(UtA, U, A);

itr++;

if (itr > 2000000)

{

printf("ERROR\n");

return 0;

}

}

for (j = 0; j < M; j++)

{

for (i = 0; i < M; i++)

{

u[i] = U1[i][j];

}

MulMatVec(B, u, Au);

for (i = 0; i < M; i++)

{

u[i] = u[i]\*A[j][j];

printf("%e\n", Au[i] - u[i]);

}

printf("\n");

}

printf("Eigenvecors:\n");

for (i = 0; i < M; i++)

{

for (j = 0; j < M; j++)

printf("%lf ", U1[i][j]);

printf("\n");

}

printf("Itteration = %d\n", itr);

return itr;

}