Санкт-Петербургский Политехнический Университет Петра Великого

Институт прикладной математики и механики

Кафедра прикладной математики

Курсовая работа

**по дисциплине "Численные методы"**

Выполнил студент гр.23631/1 Д. Д. Пестряков

Преподаватель: Л.В Павлова

Санкт-Петербург

**2018**

Оглавление

[Лабораторная 1: Решение алгебраических и трансцендентных уравнений. 2](#_Toc533035811)

[Формулировка задачи и ее формализация. 2](#_Toc533035812)

[Предварительный анализ задачи и установка условий применимости. 2](#_Toc533035813)

[Алгоритм метода и условия его применимости. 2](#_Toc533035814)

[Тестовый пример с детальными расчетами для задачи малой размерности (Метод Простых Итераций). 3](#_Toc533035815)

[Перечень контрольных тестов для иллюстрации метода средствами пакета MATLAB. 5](#_Toc533035816)

[Модульные составляющие программы. 6](#_Toc533035817)

[Численный анализ решения задачи. 6](#_Toc533035818)

[Вывод. 10](#_Toc533035819)

[Лабораторная 2: Решение СЛАУ прямыми методами 11](#_Toc533035820)

[Формулировка задачи и ее формализация. 11](#_Toc533035821)

[Алгоритм метода и условия его применимости. 11](#_Toc533035822)

[Предварительный анализ задачи и проверка условий применимости. 12](#_Toc533035823)

[Тестовый пример с детальными расчетами для задачи малой размерности (QR-разложение). 12](#_Toc533035824)

[Модульная структура программы. 13](#_Toc533035825)

[Численный анализ решения задачи. 15](#_Toc533035826)

[Оценка эффективности QR-разложения методом Грамма-Шмидта. 17](#_Toc533035827)

[Вывод. 17](#_Toc533035828)

[Лабораторная 3: Решение СЛАУ итерационными методами 19](#_Toc533035829)

[Формулировка задачи и ее формализация. 19](#_Toc533035830)

[Предварительный анализ задачи и установка условий применимости. 19](#_Toc533035831)

[Алгоритм метода и условия его применимости 19](#_Toc533035832)

[Тестовый пример с детальными расчетами для задачи малой размерности. 20](#_Toc533035833)

[Модульная структура программы. 20](#_Toc533035834)

[Численный анализ решения задачи. 21](#_Toc533035835)

[Вывод. 21](#_Toc533035836)

[Лабораторная 4: Решение Алгебраической Проблемы Собственных Значений 23](#_Toc533035837)

[Формулировка задачи и ее формализация. 23](#_Toc533035838)

[Предварительный анализ задачи и установка условий применимости. 23](#_Toc533035839)

[Алгоритм метода и условия его применимости. 23](#_Toc533035840)

[Тестовый пример с детальными расчетами для задачи малой размерности. 24](#_Toc533035841)

[Модульная структура программы. 24](#_Toc533035842)

[Численный анализ решения задачи. 25](#_Toc533035843)

[Вывод. 26](#_Toc533035844)

[Заключение. 27](#_Toc533035845)

# Лабораторная 1: Решение алгебраических и трансцендентных уравнений.

## Формулировка задачи и ее формализация.

Определение корней алгебраического и трансцендентного уравнений Методом Половинного Деления и Методом Простых Итераций, где под корнем понимается такое значение x, что f(x) = 0.

## Предварительный анализ задачи и установка условий применимости.

Условия f(x) – определена и непрерывна до первой производной на отрезке [a,b] и f(a)\*f(b)<0 гарантируют, что на интервале [*a,* *b*] находится хотя бы 1 корень.

Из того, что ||≤ q < 1 на отрезке [a,b], вытекает сходимость Метода Простых Итераций.

## Алгоритм метода и условия его применимости.

1. Метод Простых Итераций

Условия применимости:

1. f(x) – определена и непрерывна до первой производной на отрезке [a,b].
2. f(a)\*f(b)<0
3. ||≤ q < 1 на отрезке [a,b].

Алгоритм:

1. Заменяем уравнение эквивалентным ему

2. Выбираем первое приближение и строим последовательность:

3. Останавливаемся, когда

Выбор :

,

где на [a,b], знак ставится в зависимости от знака первой производной функции на выбранном промежутке.

При таком выборе ,

где , на [a,b],

1. Метод Половинного Деления

Алгоритм:

1. Вычислим точку c:
2. Если (b – a) < 2\*ɛ, то х = с и остановиться.
3. Вычислить f(c).
4. Если f(c) = 0, то x = c и остановиться.
5. Если f(a)\*f(c)<0, то b = c и f(b) = f(c), вернуться к шагу 1, иначе a = c и f(a) = f(c), вернуться к шагу 1.
6. Условие выхода:

Условия применимости:

1. f(x) – определена и непрерывна до первой производной на отрезке [a,b].
2. f(a)\*f(b)<0

## Тестовый пример с детальными расчетами для задачи малой размерности (Метод Простых Итераций).

*ɛ = 0,0001*

⬄ -5 \* 22 < 0 => существует хотя бы 1 корень. – знакопостоянна на выбранном промежутке => f(x) – монотонна на заданном промежутке.

m1 = min {|f’(0)|, |f’(3)|} = min {|3|, |15|} = 3;

M1 = max {|f’(0)|, |f’(3)|} = max {|3|, |15|} = 15;

3. Продолжаем дальше.
5. Продолжаем дальше.

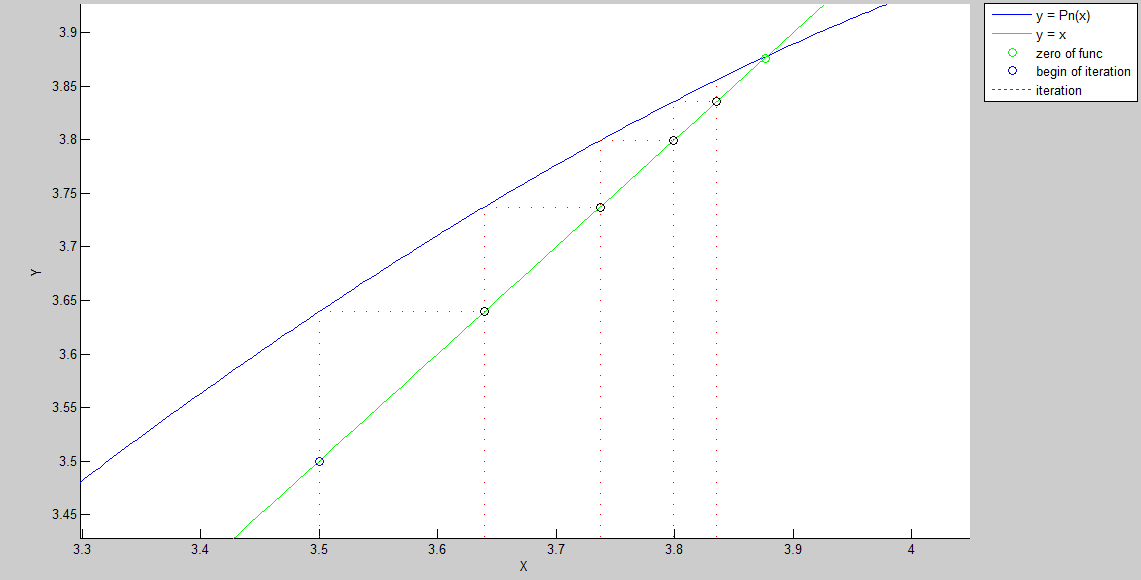
## Перечень контрольных тестов для иллюстрации метода средствами пакета MATLAB.

1. Алгебраическое уравнение

|  |  |  |
| --- | --- | --- |
| С помощью fzero | МПИ | МПД |
| 0.992387 | 0.992387 | 0.992387 |
| 3.876227 | 3.876227 | 3.876227 |

1. Трансцендентное уравнение

|  |  |  |
| --- | --- | --- |
| С помощью fzero | МПИ | МПД |
| 0.311516 | 0.311516 | 0.311516 |



.Иллюстрация МПИ для полинома

## Модульные составляющие программы.

Все составные части программы реализованы на языке матлаб, результаты которых записываются в таблице Excel:

lab.m:

основной исполняемый файл

function Bis = Bisection\_method (func, a, b, t) из файла Bisection\_method.m:

входные данные: func – функция в строковом виде, a – нижний конец промежутка, b – верхний конец, t – корень функции, вычисленный с помощью fzero.

выходные данные: корень.

function Iter = SIM(func, a, b, Fm, FM, t) из файла SIM.m:

входные данные: то же самое, что и предыдущая с добавлением параметров Fm и FM – запись функции типа sym для вычисления минимума и максимум функции.

выходные данные: корень.

function Iter = Draw (a, b, Fm, FM, n) из файла Draw.m:

входящие данные: a – нижний конец промежутка, b – верхний конец, n – корень функции, вычисленный с помощью fzro, Fm и FM – запись функции типа sym для вычисления минимума и максимум функции.

выходные данные: график.

## Численный анализ решения задачи.

1. Алгебраическое уравнение  
   Формула для поиска верхней и нижней границы:

Где ,

Для поиска верхней границы положительных корней

Для нижней границы положительных корней

Как показало исследование функции, у данного полинома нет отрицательных корней.

Для положительных:

Верхняя граница:

Нижняя граница:

a1 = 0,6276

b1 = 2

a2 = 3

b2 = 4,5

f(a1)\*f(b1)<0: 13,127 \* (-45,377) < 0 => существует хотя бы 1 корень.

f(a2)\*f(b2)<0: (-65,938) \* 140,305 < 0 => существует хотя бы 1 корень.

1. Трансцендентное

a3 = 0

b3 = 0,5

f(a3)\*f(b3)<0: (-1) \* 0,5 < 0 => существует хотя бы 1 корень.

Найдем соответствующие корни.

Корни:

1. Алгебраического уравнения

Метод Половинного Деления:

На промежутке [0.6276, 2] x = 0.992387

Метод Простых Итераций:

На промежутке [3, 4.5] x = 3.876227

1. Трансцендентного уравнения

Метод Половинного Деления:

На промежутке [0, 0.5] x = 0.311516

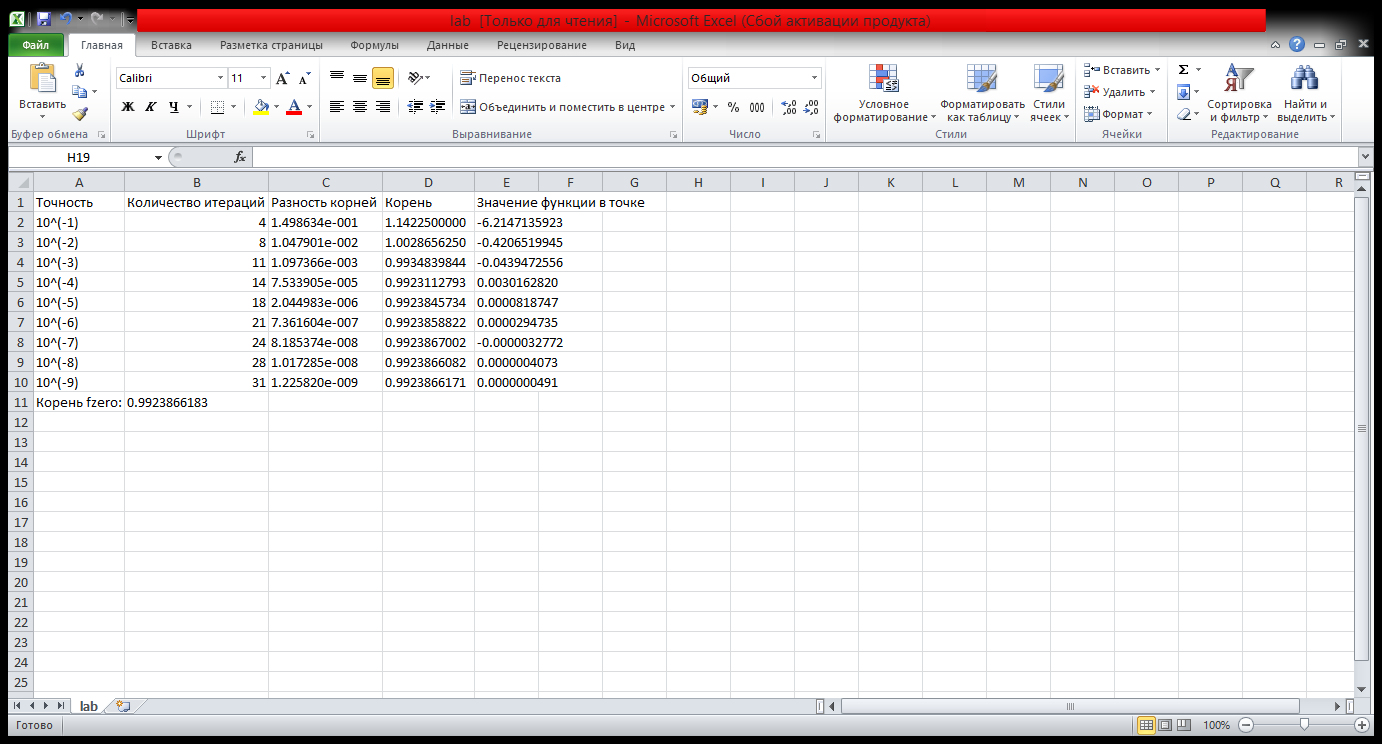
Метод Простых Итераций:

На промежутке [0, 0.5] x = 0.311516

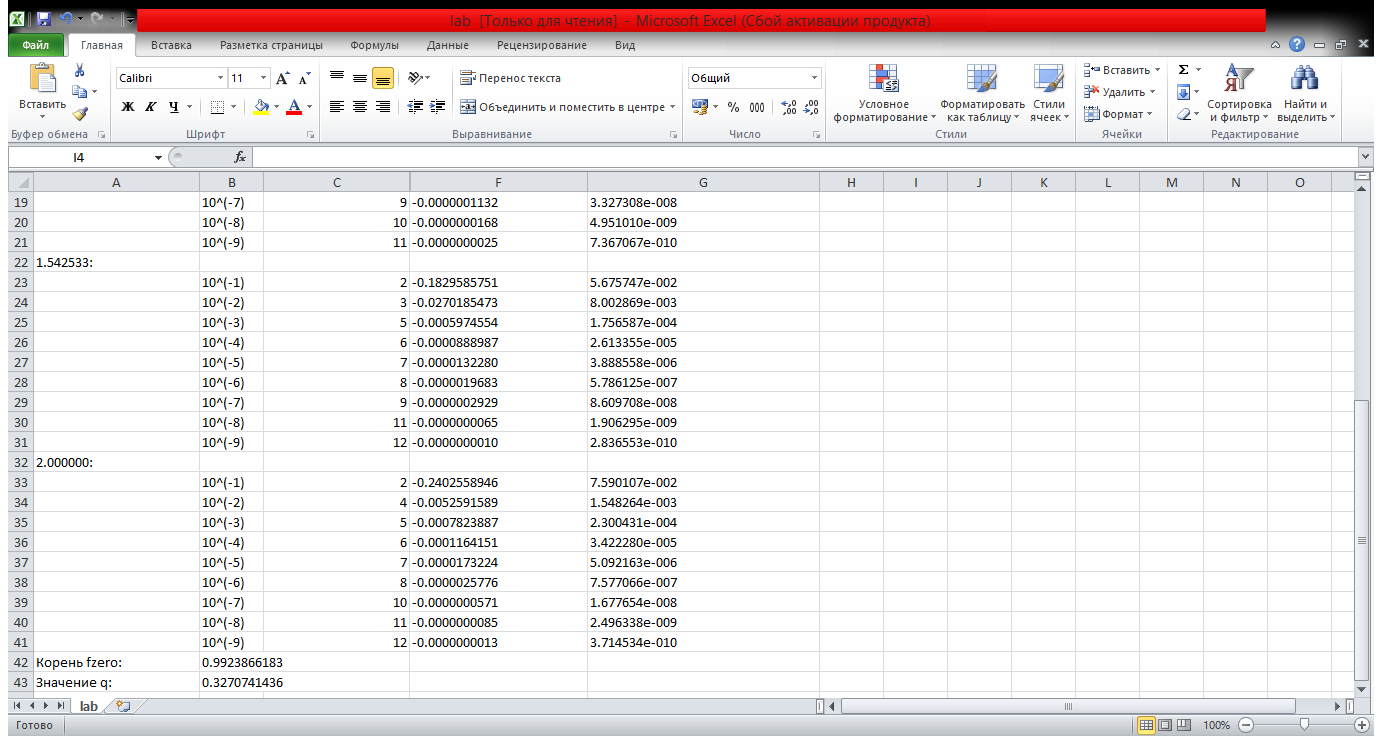
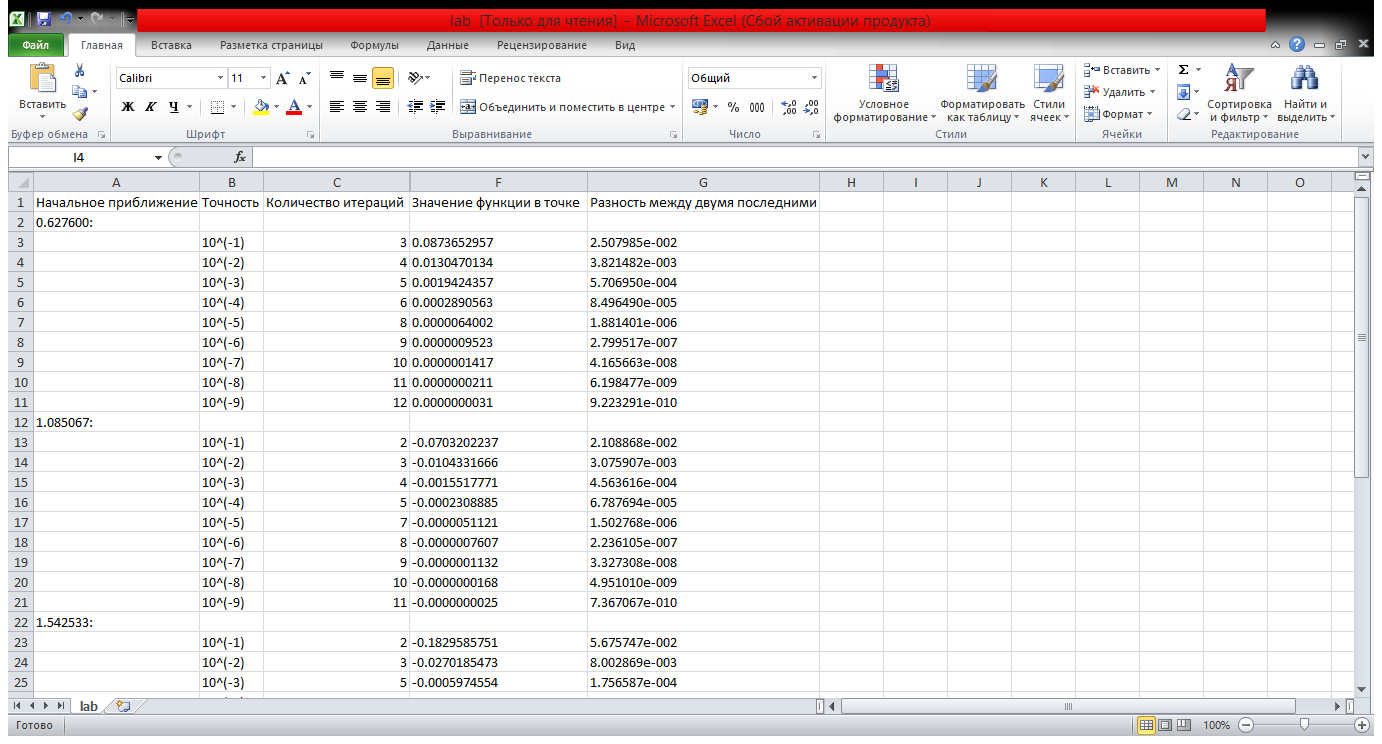
Исследование зависимости числа итераций от заданной точности:

1. Алгебраическое уравнение

Промежуток [0.6276, 2], корень 0.992387, Метод Половинного Деления

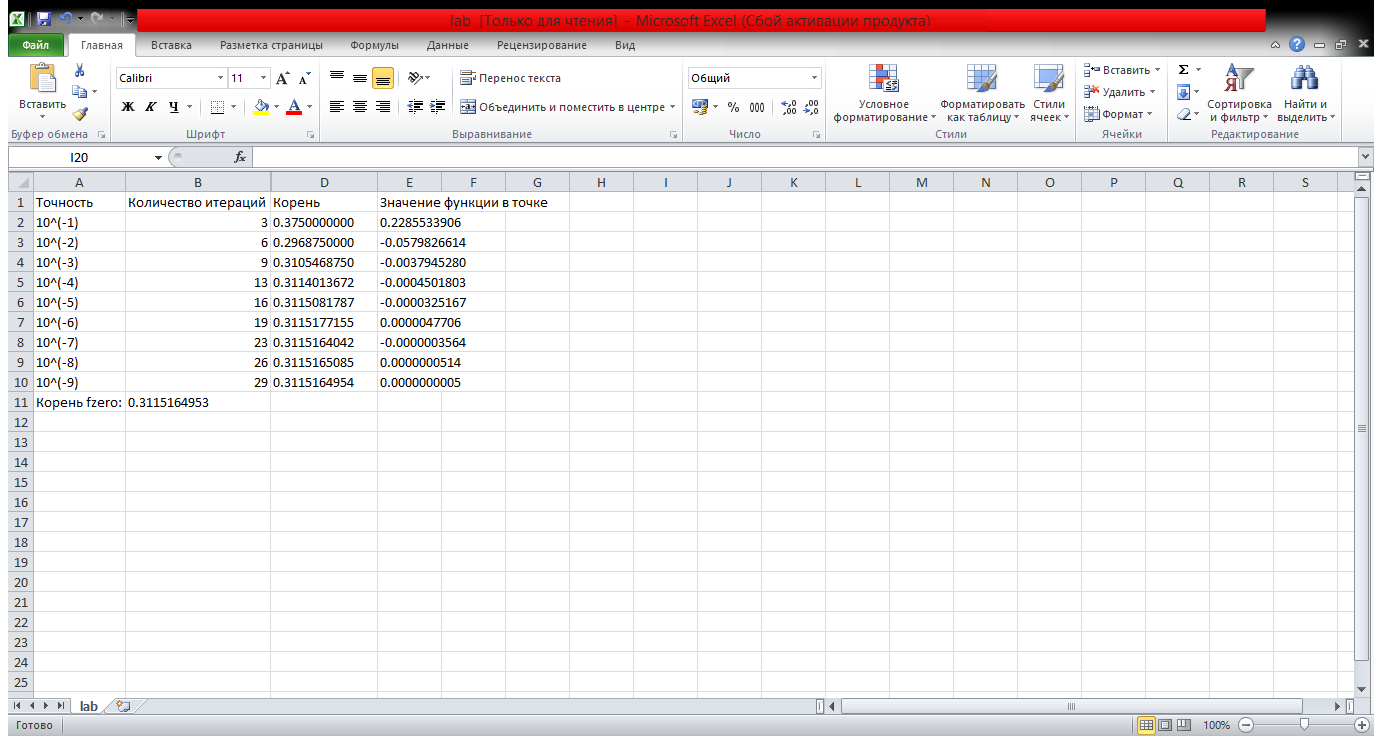


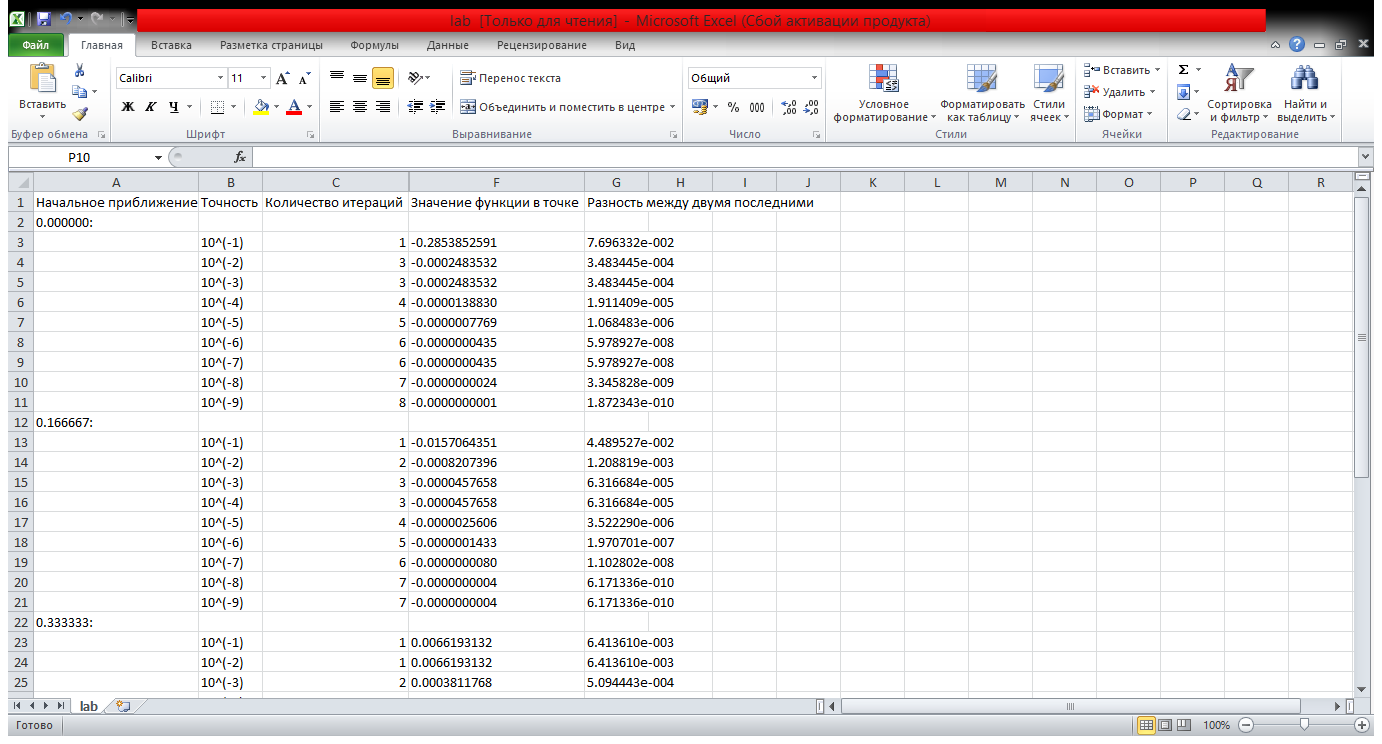
Промежуток [0.6276, 2], корень 0.992387, Метод Простых Итераций

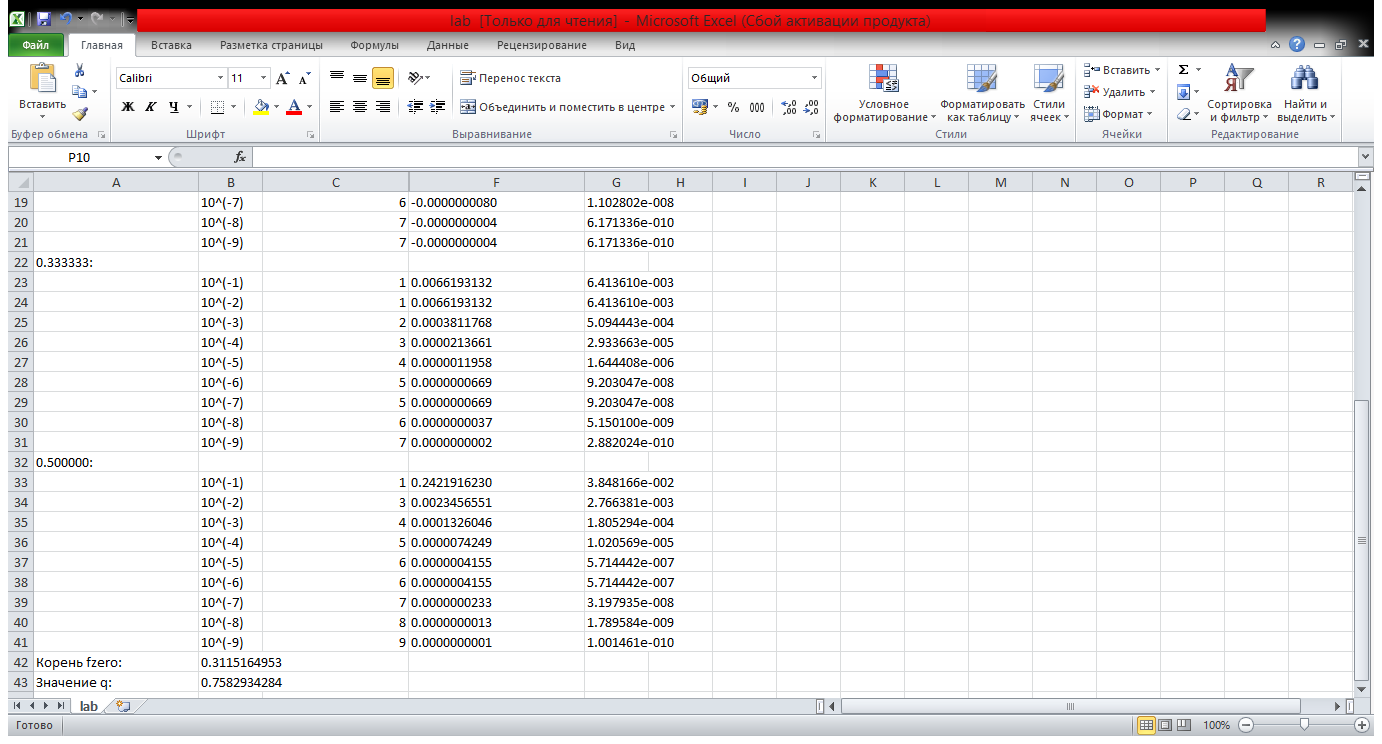


1. Трансцендентное уравнение

Промежуток [0, 0.5], корень 0.311516, Метод Половинного Деления



Промежуток [0, 0.5], корень 0.311516, Метод простых итераций



## Вывод.

В ходе лабораторной работы мы изучили и научились использовать 2 метода приближенного вычисления корней.

В ходе исследований было выявлено, что Методу Простых Итераций (далее МПИ) требуется меньшее количество итераций для того, чтобы достичь нужную точность, чем Методу Половинного Деления (далее МПД). Заметим, что апостериорная оценка для МПИ так же уменьшает количество итераций. Также для МПИ и МПД характерно увеличение числа итераций с увеличением начального приближения в случае МПИ и увеличения длины промежутка в случае МПД.

# Лабораторная 2: Решение СЛАУ прямыми методами

## Формулировка задачи и ее формализация.

Найти решение СЛАУ Ах = b методом QR-разложения, сравнить точность решения для матриц с разным числом обусловленности. Исследовать матрицу 10х10. Здесь R – верхнетреугольная матрица.

Прямыми методами называются такие вычислительные методы, где искомая величина может быть вычислена точно. Идея состоит в различных методов разложения матрицы на различные произведения матриц специального вида (верхнетреугольные или нижнетреугольные, диагональные, ортогональные и другие), которые обладают рядом различных удобных для вычислений свойств.

## Алгоритм метода и условия его применимости.

QR-разложение

А = QR, где

Алгоритм:

Примечания: под Q(i) будем понимать i-ый столбец соответствующей таблицы, под (Q(i), R(j)) будем понимать скалярное произведение вычисляемое как .

1. r(11) =
2. r(12) = (Q(2), Q(1))
3. Q(2) = Q(2) –
4. r(22) =
5. Продолжаем процесс, пока не ортонормируем n-ый столбец

В итоге получаются формулы:

.

Условия применимости:

Все главные миноры матрицы А отличны от нуля

Вычеркиваем (n – j) столбцы и (n – j) строки из матрицы А:

## Предварительный анализ задачи и проверка условий применимости.

Проверяем с помощью MatLab.

Также можно ввести малую величину eps порядка 10 в минус 6 как условие досрочного завершения процесса в силу численной неустойчивости метода.

## Тестовый пример с детальными расчетами для задачи малой размерности (QR-разложение).

В силу численной неустойчивости метода будем проводить все расчеты приближенно.

1. Q(1) =
2. r(11) =
3. r(21) = \* Q(2) = 2 \* 1 + 5 \* 4 + 8 \* 7 = 78
4. Q(2) =
5. r(22) = = 0,82
6. r(31) = \* Q(3) = 3 \* 1 + 6 \* 4 + 10 \* 7 = 97
7. r(32) = \* Q(3) = 0,8 \* 3 + 0,3 \* 6 + (-0,3) \* 10 = 1,2
8. Q(3) =
9. r(33) = = 0,29

,

\*

## Модульная структура программы.

double\*\* InitMatr(int sizeL, int sizeC):

принимает: sizeL - количество строк, sizeC - количество столбцов.  
возвращает: нулевую матрицу с заданными размерами.

void DestroyMatr(double\*\* Matr, int sizeL, int sizeC):

принимает: Matr - матрица, sizeL – количество строк, sizeC – количество строк.

double\*\* MultMatr(double\*\* A, double\*\* B, int sizeLA, int sizeCA, int sizeLB, int sizeCB):

принимает: A, B - две матрицы, sizeLA - количество столбцов A, sizeCA – количество строк A, sizeLB - количество столбцов B, sizeCB – количество строк B.  
возвращает: матрицу – результат перемножения матриц A и B.

double\*\* TranspMatr(double\*\* Matr, int sizeL, int sizeC):  
принимает: Matr - матрица, sizeL – количество строк, sizeC – количество строк.  
возвращает: матрицу – результат транспонирования исходной.

void ColChange(double\*\* A, double\*\* B, int sizeLA, int ColNum):

принимает: A, B - две матрицы, sizeLA - количество столбцов A, ColNum – номер колонки из A, передаваемый в колонку B.

double NormaTwo(double\*\* Matr, int size):

принимает: Matr - матрица, size – количество элементов в столбце.  
возвращает: вторую норму столбца.

int GramShmidtProcess(double\*\* A, double\*\* Q, double\*\* R, int sizeLA, int sizeCA):

принимает: A – исходная матрица, Q – указатель на матрицу Q, R – указатель на матрицу R, sizeLA - количество столбцов A, sizeCA – количество строк A.  
возвращает: 1 в случае успешного выполнения и 0 в случае невозможности разложения.

void PrintMatr(double\*\* Matr, int sizeL, int sizeC):

принимает: Matr - матрица, sizeL – количество строк, sizeC – количество строк.

double MatrNorm(double\*\* Matr, int sizeL, int sizeC):

принимает: Matr - матрица, sizeL – количество строк, sizeC – количество строк.  
возвращает: бесконечную норму матрицы по столбцу.

double\*\* GetNewMatr(double\*\* Matr, double\*\* Delta, int sizeL, int sizeC):

принимает: Matr - матрица, Delta – указатель на матрицу погрешнстей, sizeL – количество строк, sizeC – количество строк.  
возвращает: матрицу с новыми коэффициентами.

double\*\* ReturnX(double\*\* R, double\*\* Y, int size):

принимает: R – матрица R, Y – матрица Y = QT\*b, size – размер столбца.

возвращает: столбец решений x.

## Численный анализ решения задачи.

1. При внесении возмущений в матрицу A выполняется следующее неравенство:
2. При внесении возмущений в вектор b выполняется следующее неравенство:

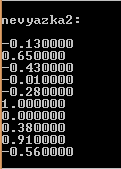
Найдем такие числа и , при которых данные строгие неравенства обращаются в равенства, и сравним их с числом cond(A).  
Для хорошо обусловленной матрицы:

Cond(A) = 100;

Внесение возмущений в b.

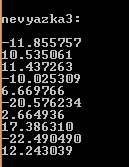
.

Вектор невязки:



Внесение возмущений в A.

Вектор невязки:



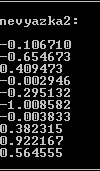
.

Для плохо обусловленной матрицы:

Cond(A) = 2000000;

Внесение возмущений в b.

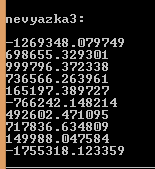
Невязка:



.

Внесение возмущений в A.

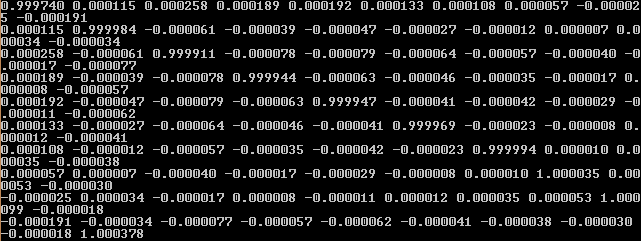
Невязка:



.

## Оценка эффективности QR-разложения методом Грамма-Шмидта.

Данный тест был использован для того, чтобы убедиться в том, что в результате процесса была получена ортогональная матрица. Перемножаются матрицы Q и транспонированная от матрицы Q:



Как видно, диагональные элементы очень близки к 1, а остальные к 0 с погрешностью порядка 10 в минус 4.

## Вывод.

Для любой матрицы из 2-х коэффициентов большим был тот , который был получен при внесении возмущений в матрицу А, а не в столбец b. Также видно, что вектор невязки покомпонентно больше при внесении возмущений в матрицу А, чем в столбец b. При внесении возмущений в матрицу А компоненты векторов невязки имели порядок не выше порядка числа обусловленности.

Коэффициенты всегда меньше числа обусловленности матрицы, так как это наименьшие числа, удовлетворяющие исследуемым нестрогим неравенствам. Именно они представляют практическую ценность, так как число обусловленности может быть велико и не отражать реального поведения СЛАУ при внесении возмущений.

Также в ходе измерений было выявлено, что для плохо обусловленной матрицы метод Грамма-Шмидта не является устойчивым и быстро накапливает погрешности вычислений.

# Лабораторная 3: Решение СЛАУ итерационными методами

## Формулировка задачи и ее формализация.

Найти приближенное решение СЛАУ Ах = b Методом Простых Итераций, исследовать зависимость числа итераций от числа обусловленности матрицы А размерности 10x10 и близости определителя к нулю.   
Итерационными методами называются такие математические методы, при которых неизвестные величины находятся путем постепенного их уточнения. Идея Метода Простых Итераций заключается в том, чтобы привести систему Ax = b к эквивалентной форме x = Bx + d так, чтобы отображение B было сжимающим.  
Теорема о сходимости гласит, что Метод Простых Итераций сходящийся, если исходная матрица системы имеет диагональное преобладание.

## Предварительный анализ задачи и установка условий применимости.

Чаще всего исходная матрица системы не имеет диагональное преобладание. В таких случаях систему можно преобразовать. Удобнее всего оказывается сделать это таким образом: . Чтобы Метод Простых Итераций сходился, достаточно, чтобы любая из ||B|| < 1.

## Алгоритм метода и условия его применимости

Алгоритм:

Будем искать матрицу B в виде B = (E - α\*A), где E – единичная матрица.

1. Заменяем СЛАУ эквивалентной ей :

2. Выбираем первое приближение и строим последовательность:

3. Останавливаемся, когда

Условия применимости:

Метод Простых Итераций сходится при . Это условие является необходимым.

## Тестовый пример с детальными расчетами для задачи малой размерности.

2. => продолжаем
3. => продолжаем
4. Видно, что процесс начинает сходиться, поэтому его можно продолжить далее и получить искомый результат.

## Модульная структура программы.

double\*\* InitMatr(int sizeL, int sizeC):

принимает: sizeL - количество строк, sizeC - количество столбцов.  
возвращает: нулевую матрицу с заданными размерами.

void DestroyMatr(double\*\* Matr, int sizeL, int sizeC):

принимает: Matr - матрица, sizeL – количество строк, sizeC – количество строк.

double\*\* MultMatr(double\*\* A, double\*\* B, int sizeLA, int sizeCA, int sizeLB, int sizeCB):

принимает: A, B - две матрицы, sizeLA - количество столбцов A, sizeCA – количество строк A, sizeLB - количество столбцов B, sizeCB – количество строк B.  
возвращает: матрицу – результат перемножения матриц A и B.

void ColChange(double\*\* A, double\*\* B, int sizeLA, int ColNum):

принимает: A, B - две матрицы, sizeLA - количество столбцов A, ColNum – номер колонки из A, передаваемый в колонку B.

void PrintMatr(double\*\* Matr, int sizeL, int sizeC):

принимает: Matr - матрица, sizeL – количество строк, sizeC – количество строк.

double MatrNorm(double\*\* Matr, int sizeL, int sizeC):

принимает: Matr - матрица, sizeL – количество строк, sizeC – количество строк.  
возвращает: бесконечную норму матрицы по столбцу.

double MatrixNormTwo(double\*\* Matr, int sizeLC):

принимает: Matr – матрица, sizeLC – размерность квадратной матрицы.  
возвращает: вторую норму матрицы.

double\*\* ReturnC(double\*\* A, double\*\* b, int sizeLC):

принимает: A – исходная матрица системы, b – вектор, задающий правый часть СЛАУ, sizeLC – размерность квадратной матрицы.  
возвращает: матрицу и вектор, готовые для итераций.

double\*\* SimpleIterMeth(double\*\* A, double\*\* b, int sizeLC, int\* count):

принимает: A – исходная матрица системы, b – вектор, задающий правый часть СЛАУ, sizeLC – размерность квадратной матрицы, count – счетчик.  
возвращает: вектор - решение.

## Численный анализ решения задачи.

Под max{} будем понимать наибольшую по модулю из компонент вектора невязки.

Для хорошо обусловленной матрицы:

* Cond(A) = 100;
* α = 0.000120;
* max{} = 9.998542e-06;
* Количество итераций: 129804.

Для плохо обусловленной матрицы:

* Cond(A) = 10000000;
* α = 0.000001;
* max{} = 1.701767e+01;
* Количество итераций: 2000000 (итерации завершены досрочно по достижении порога числа итераций).

Стоит отметить, что плохо обусловленная матрица была выбрана с определителем, близким к нулю.

В обоих случаях α = .

## Вывод.

Метод Простых Итераций эффективнее всего работает для матриц с хорошим числом обусловленности. Хотя в целом является затратным по времени. Также мы видим, что метод плохо сходится, если матрица плохо обусловлена и имеет определитель близкий к нулю, так как ранее было установлено, что при внесении очень малых возмущений в правую часть такой СЛАУ решение сильно меняется, а на каждой итерации мы получаем различные свободные вектора справа. Достоинствами данного метода являются простота его использования в случае диагонального преобладания матрицы и в том, что для данного метода достаточно вычислять всего лишь по одной компоненте, что экономит память при проведении огромных вычислительных операций. Исследование данного метода так же показало, что в случае приведения матрицы к виду, удобному для итераций, с помощью коэффициента α, сходимость сильно зависела от этого параметра. Апостериорная оценка в данном методе проявила свою неэффективность, так как в случае очень маленького значения коэффициента α она ухудшала условие выхода из цикла, так как величина , являющаяся оценкой, устремлялась к бесконечности при , которые были относительно больше .

# Лабораторная 4: Решение Алгебраической Проблемы Собственных Значений

## Формулировка задачи и ее формализация.

Найти собственные значения, или собственные числа, и собственные векторы матрицы A методом Якоби. Исследовать зависимость числа итераций для матриц с хорошо разделимыми собственными значениями и плохо разделимыми собственными значениями, а также с хорошим и плохим числом обусловленности.

Собственным значением или числом матрицы A называется такое число λ, что , ( при х=0 этоверно всегда!!)где x – собственный вектор матрицы A, соответствующий собственному числу λ. Все собственные числа вычисляются по формуле:

## Предварительный анализ задачи и установка условий применимости.

Суть метода Якоби в том, что всякая симметричная матрица А порядка n может быть представлена в виде: A = ΛQ, где Q – ортогональная матрица, Λ – диагональная с элементами λ1, … , λn – собственными числами матрицы А. Тогда матрица Q – есть матрица, столбцы которой составлены из собственных векторов матрицы A.

## Алгоритм метода и условия его применимости.

Алгоритм:

Будем использовать алгоритм с оптимизацией (каждый раз уничтожать наибольший вне диагональный элемент) :

1. Ищем такое, что для него выполнено – максимально.
2. В строке ищем максимальный элемент, который не совпадает с диагональным и запоминаем его индекс .
3. Вычисляем величину
4. Строим ортогональную матрицу Q поворота по правилам: вне диагонали ставим нули, по диагонали ставим единицы. На месте Q[][ ] = Q[][ ] = cos(); Q[][] = -1 \* Q[][ ] = sin()
5. до тех пор, пока

Условия применимости:

Метод Якоби имеет применение только в случае, если матрица A является симметричной, то есть

## Тестовый пример с детальными расчетами для задачи малой размерности.

1. Находим 4 + 9 = 13; 4 + 9 = 13; 9 + 9 = 18;
2. = 3;
3. Находим . 3; 3;
4. Q[][ ] = Q[][ ] = 0.76; Q[][] = -1 \* Q[][ ] = -0.64;
5. Q =
6. Находим 0.16 + 12.67 = 12.83; 0.4 + 0= 0.16; 12.67 + 0 = 12.67;
7. = 1;
8. Находим . 0.4; 3.56;

## Модульная структура программы.

double\*\* InitMatr(int sizeL, int sizeC):

принимает: sizeL - количество строк, sizeC - количество столбцов.  
возвращает: нулевую матрицу с заданными размерами.

void DestroyMatr(double\*\* Matr, int sizeL, int sizeC):

принимает: Matr - матрица, sizeL – количество строк, sizeC – количество строк.

double\*\* MultMatr(double\*\* A, double\*\* B, int sizeLA, int sizeCA, int sizeLB, int sizeCB):

принимает: A, B - две матрицы, sizeLA - количество столбцов A, sizeCA – количество строк A, sizeLB - количество столбцов B, sizeCB – количество строк B.  
возвращает: матрицу – результат перемножения матриц A и B.

void PrintMatr(double\*\* Matr, int sizeL, int sizeC):

принимает: Matr - матрица, sizeL – количество строк, sizeC – количество строк.

double\* SearchMax(double\*\* Matr, int size):

принимает: Matr - матрицу, size – размерность матрицы.  
возвращает: массив, содержащий максимальный вне диагональный элемент и его индексы.

double\*\* InitY(double\*\* Matr, int size, int i, int j):

принимает: Matr - матрица, size – размерность матрицы, i, j – координаты максимального элемента.  
возвращает: матрицу поворота.

double\*\* JacobiMethod(double\*\* A, int size, int\* count):  
принимает: Matr - матрица, size – размерность матрицы, count – счетчик итераций.  
возвращает: диагональную матрицу.

## Численный анализ решения задачи.

Все вычисления проводятся для

Для матрицы с хорошим числом обусловленности и хорошо отделимыми собственными значениями:

* Число итераций: 83
* Вектор невязки: 10e-6

Для матрицы с хорошим числом обусловленности и плохо отделимыми собственными значениями:

* Число итераций: 69
* Вектор невязки: 10e-3

Для матрицы с плохим числом обусловленности и хорошо отделимыми собственными значениями (с определителем, близким к нулю):

* Число итераций: 126
* Вектор невязки: 10e-5

Для матрицы с плохим числом обусловленности и плохо отделимыми собственными значениями:

* Число итераций: 69
* Вектор невязки: 10e-2

## Вывод.

Для матриц с плохо отделимыми собственными значениями метод Якоби работает быстрее, чем с хорошо отделимыми собственными значениями. Это происходит в силу того, что внедиагональные элементы для таких матриц будут также близкими числами. Стоит отметить, что в случае близости собственных чисел к нулю метод сходится быстрее, но вырастает вычислительная погрешность, так как существует понятие, как машинный ноль, поэтому для хорошо отделимых значений метод работает на компьютере дольше. О возрастании погрешности можно судить по максимальной компоненте вектора невязки, вычисленного как r = Aλ – λx. Скорость сходимости снижает и близость определителя матрицы к нулю.

## Заключение.

В заключение хочется отметить, что многие используемые в современности численные методы решения тех или иных задач имеют разную эффективность, обусловленную теми или иными факторами. При реализации многих методов на компьютере - это проблема хранения численных данных. К примеру, реализованный мною алгоритм QR – разложения по Граму-Шмидту является прямым методом решения СЛАУ, но при реализации на компьютере оказывается малоэффективным, так как большинство чисел, получаемых при ортогонализации векторов, являются иррациональными и при каждом вычислении грубо округляются. Поэтому он годится только для ручных вычислений на матрицах малых размерностей.  
Большое распространение получили итерационные методы решения задач, таких как нахождение корней уравнения, нахождение вектора решений СЛАУ или собственных значений матриц. Эти методы во многом хорошо реализуются на компьютере в силу того, что полученные результаты постоянно уточняются до тех пор, пока не будет получена приемлемая погрешность. Хотя эти методы имеют ряд ограничений, связанных, прежде всего, с условиями сходимости, если говорить о решении уравнений или их систем, поэтому приходится решать дополнительную задачу приведения уравнений или их систем к виду, удобному для итераций. Есть также методы, которые имеют ограниченное применение, к примеру, метод Якоби поиска собственных значений и собственных векторов, но весьма эффективно работающие для тех матриц, к которым его можно применять.  
Если проследить ход всех лабораторных работ, связанных с матрицами, то можно заметить, что огромную роль для результата вычислений играет число обусловленности. Оно показывает, насколько внесение возмущений в исходную матрицу может изменить результат, в чем не раз можно было убедиться. Поэтому знание числа обусловленности имеет огромное значение, так как при вычислении на компьютере постоянно вносятся погрешности.