Санкт-Петербургский   
Политехнический Университет  
Петра Великого

Курсовой проект

Выполнил: студент второго курса

Дамаскинский К., группа 23631/2

Часть I. Численные методы решения алгебраических и трансцендентных уравнений. Метод половинного деления и метод Ньютона.

Часть II. Прямые методы решения СЛАУ. LDR-разложение.

Часть III. Итерационные методы решения СЛАУ. Метод релаксации.

Часть IV. Итерационные методы решения частичной АПСЗ. Метод обратных итераций со сдвигом.

[1. Численные методы решения алгебраических и трансцендентных уравнений. 4](#_Toc533600213)

[1.1. Постановка задачи. 4](#_Toc533600214)

[1.2. Решение методом половинного деления. 4](#_Toc533600215)

[1.2.1. Алгоритм. 4](#_Toc533600216)

[1.2.2. Условия сходимости. 4](#_Toc533600217)

[1.2.3. Тестовый пример. 4](#_Toc533600218)

[1.2.4. Подготовка контрольных тестов. 4](#_Toc533600219)

[1.2.5. Геометрическая интерпретация. 5](#_Toc533600220)

[1.2.6. Численный анализ решения. 5](#_Toc533600221)

[1.2.7. Выводы о методе 7](#_Toc533600222)

[1.3. Решение методом Ньютона. 7](#_Toc533600223)

[1.3.1. Алгоритм. 7](#_Toc533600224)

[1.3.2. Условия сходимости. 7](#_Toc533600225)

[1.3.3. Тестовый пример. 7](#_Toc533600226)

[1.3.4. Подготовка контрольных тестов. 7](#_Toc533600227)

[1.3.5. Геометрическая интерпретация. 8](#_Toc533600228)

[1.3.6. Численный анализ решения. 8](#_Toc533600229)

[1.3.7. Выводы о методе. 9](#_Toc533600230)

[1.4. Несколько слов о функции fzero. 9](#_Toc533600231)

[1.5. Модульная структура программы. 9](#_Toc533600232)

[2. Прямые методы решения СЛАУ. 10](#_Toc533600233)

[2.1. Постановка задачи. 10](#_Toc533600234)

[2.2. Решение методом LDR-разложения. 10](#_Toc533600235)

[2.2.1. Алгоритм. 10](#_Toc533600236)

[2.2.2. Условия применимости. 11](#_Toc533600237)

[2.2.3. Тестовый пример. 12](#_Toc533600238)

[2.2.4. Подготовка контрольных тестов. 13](#_Toc533600239)

[2.2.5. Численный анализ решения. 13](#_Toc533600240)

[2.2.6. Выводы о методе. 14](#_Toc533600241)

[2.3. Модульная структура программы. 14](#_Toc533600242)

[3. Итерационные численные методы решения СЛАУ. 15](#_Toc533600243)

[3.1. Постановка задачи. 15](#_Toc533600244)

[3.2. Решение методом релаксации. 15](#_Toc533600245)

[3.2.1. Алгоритм. 15](#_Toc533600246)

[3.2.2. Условия сходимости. 16](#_Toc533600247)

[3.2.3. Тестовый пример. 16](#_Toc533600248)

[3.2.4. Подготовка контрольных тестов. 17](#_Toc533600249)

[3.2.5. Замечание о генерации целочисленной положительно определённой матрицы. 18](#_Toc533600250)

[3.2.6. Численный анализ решения. 18](#_Toc533600251)

[3.2.7. Выводы о методе. 21](#_Toc533600252)

[3.3. Модульная структура программы. 21](#_Toc533600253)

[4. Итерационные методы решения частичной АПСЗ. 22](#_Toc533600254)

[4.1. Постановка задачи. 22](#_Toc533600255)

[4.2. Метод обратных итераций. 23](#_Toc533600256)

[4.2.1. Алгоритм. 23](#_Toc533600257)

[4.2.2. Условия сходимости. 23](#_Toc533600258)

[4.2.3. Дополнительное замечание о сходимости. 24](#_Toc533600259)

[4.2.4. Тестовый пример. 24](#_Toc533600260)

[4.2.5. Подготовка контрольных тестов. 25](#_Toc533600261)

[4.2.6. Численный анализ решения. 25](#_Toc533600262)

[4.2.7. Выводы о методе. 27](#_Toc533600263)

[4.3. Модульная структура программы. 28](#_Toc533600264)

# Численные методы решения алгебраических и трансцендентных уравнений.

## Постановка задачи.

Требуется решить уравнение вида с некоторой точностью (), где .

Даны две функции , относительно которых требуется решить описанное выше уравнение.

## Решение методом половинного деления.

### Алгоритм.

Выбирается интервал . Далее интервал делится пополам на два отрезка

Тогда либо:

1. , либо
2. .

В первом случае повторяем алгоритм на интервале (присваиваем ), во втором – (.

Повторяем алгоритм, пока длина интервала больше .

### Условия сходимости.

1);

2).

*Примечание*: для полиномиальной функции границы поиска корней могут быть найдены по теореме о верхней границе положительных корней.

### Тестовый пример.

Для функции на интервале метод будет работать следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| *Шаг* | *Приближённый корень (c)* |  |  |
| 1 | *0* |  |  |
| 2 |  | + | – |
| 3 |  | + | – |
| 4 |  | – | + |
| 5 |  | + | – |

### Подготовка контрольных тестов.

1. Для нахождения границ интервала, на котором будет производиться поиск корней (параметров полиномиальной функции, применим теорему о верхней границе положительных корней:

– верхняя граница положительных корне

нижняя граница положительных корней.

Далее на данном участке путём исследования и построения примерного графика было найдено более точное начальное приближение, на котором имеется ровно один корень:

.

Для трансцендентной функции начальное приближение подбирается путём построения графика в области, на границах которой знаки функции разные.

1. Полиномиальная и показательная функции (и, как следствие, их произведения) являются непрерывными на области определения и, следовательно, на .

Из первого и второго пунктов следует корректность применения метода.

1. Для построения третьего графика были вычислены корни с точностью до . Выше описанные замечания соблюдаются для любого интервала, содержащего приближённый корень и вложенного в и в для соответствующих функций. Это необходимо для построения третьего графика.
2. Описанные выше условия также соблюдаются на интервале [-2;18] для полиномиальной функции и [-0.5; 1.5] трансцендентной – это необходимо для построения четвёртого графика (метод должен быть корректен на этих интервалах).

### Геометрическая интерпретация.

На графике вертикальными линиями отмечены прямые для нескольких итераций (номера итераций подписаны рядом с линией).

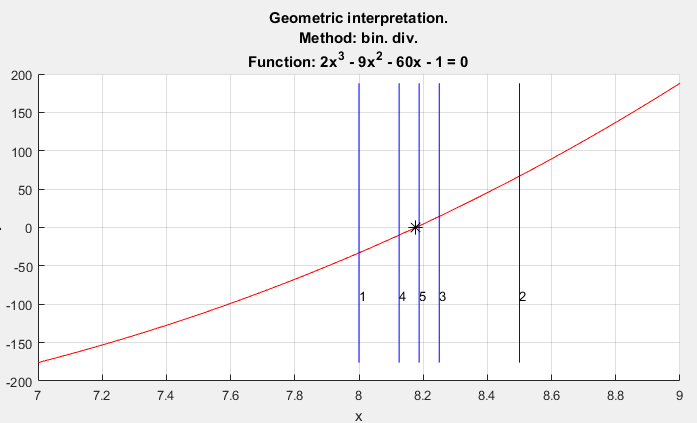


Рисунок 1. Геометрическая интерпретация для функции

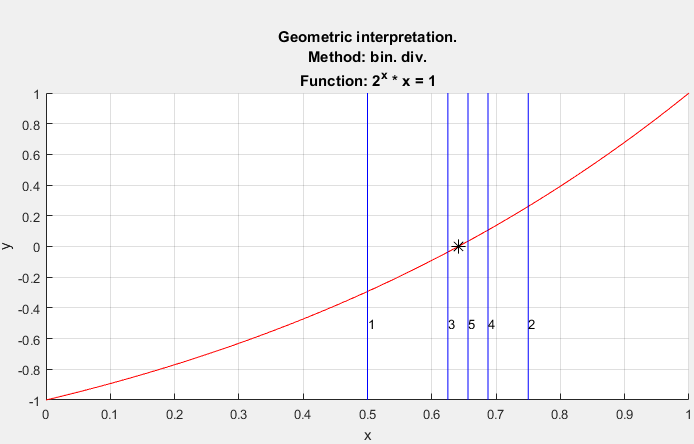


Рисунок 2. Геометрическая интерпретация для функции

### Численный анализ решения.

Из алгоритма понятно, что , где N – число шагов, необходимых для выполнения метода.

На следующем графике показана зависимость количества итераций от заданной точности.

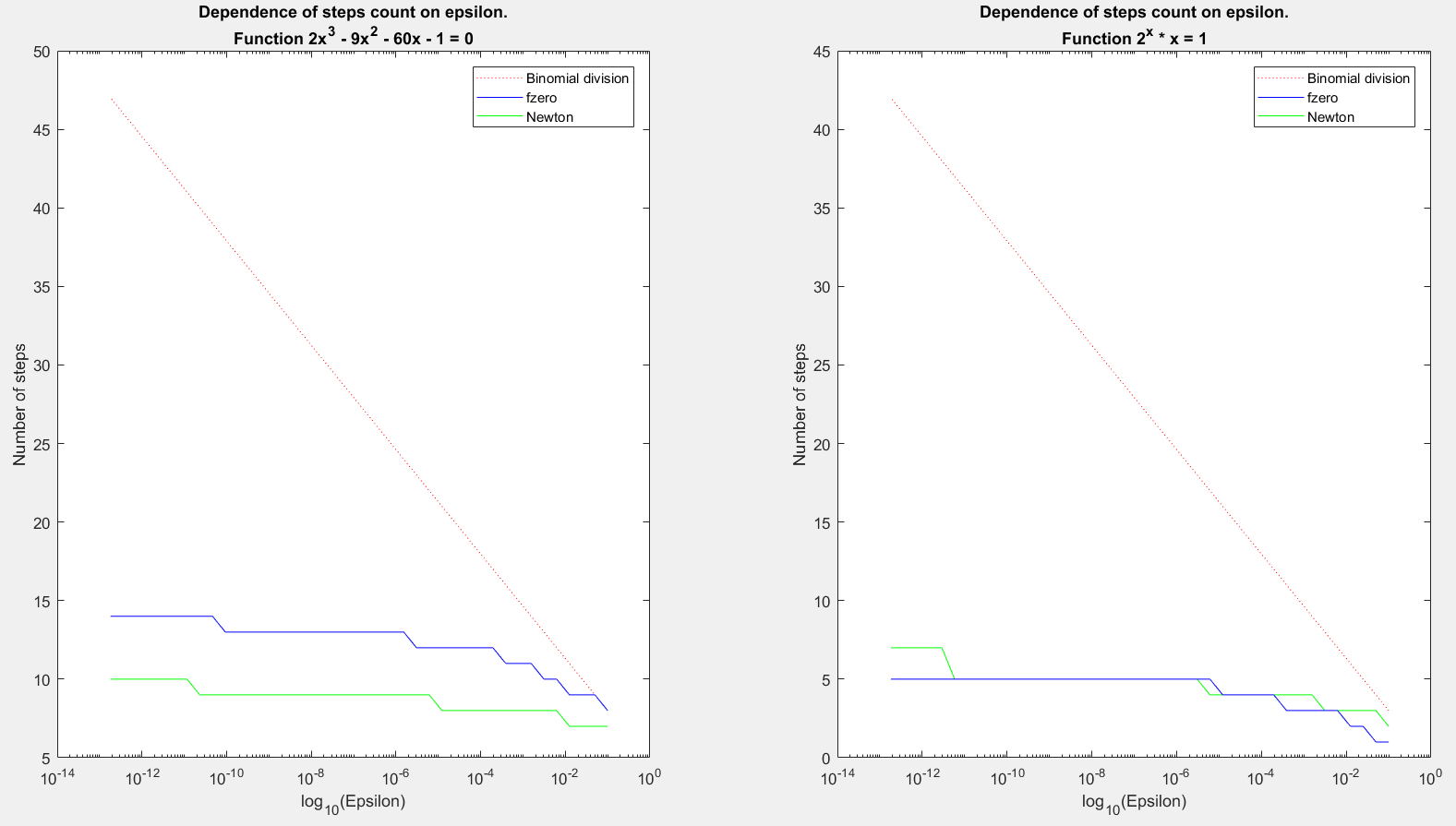


Рисунок 3. Зависимость количества итераций от заданной точности

Красным показана кривая для метода половинного деления. Видно, что зависимость от показателя точности линейная – подтверждается теоретический результат () – это следует из свойств логарифма.

Для поиска корня с точностью у функции был затрачен 21 шаг, у функции – 16.

Зависимость от начального приближения также получается линейной:

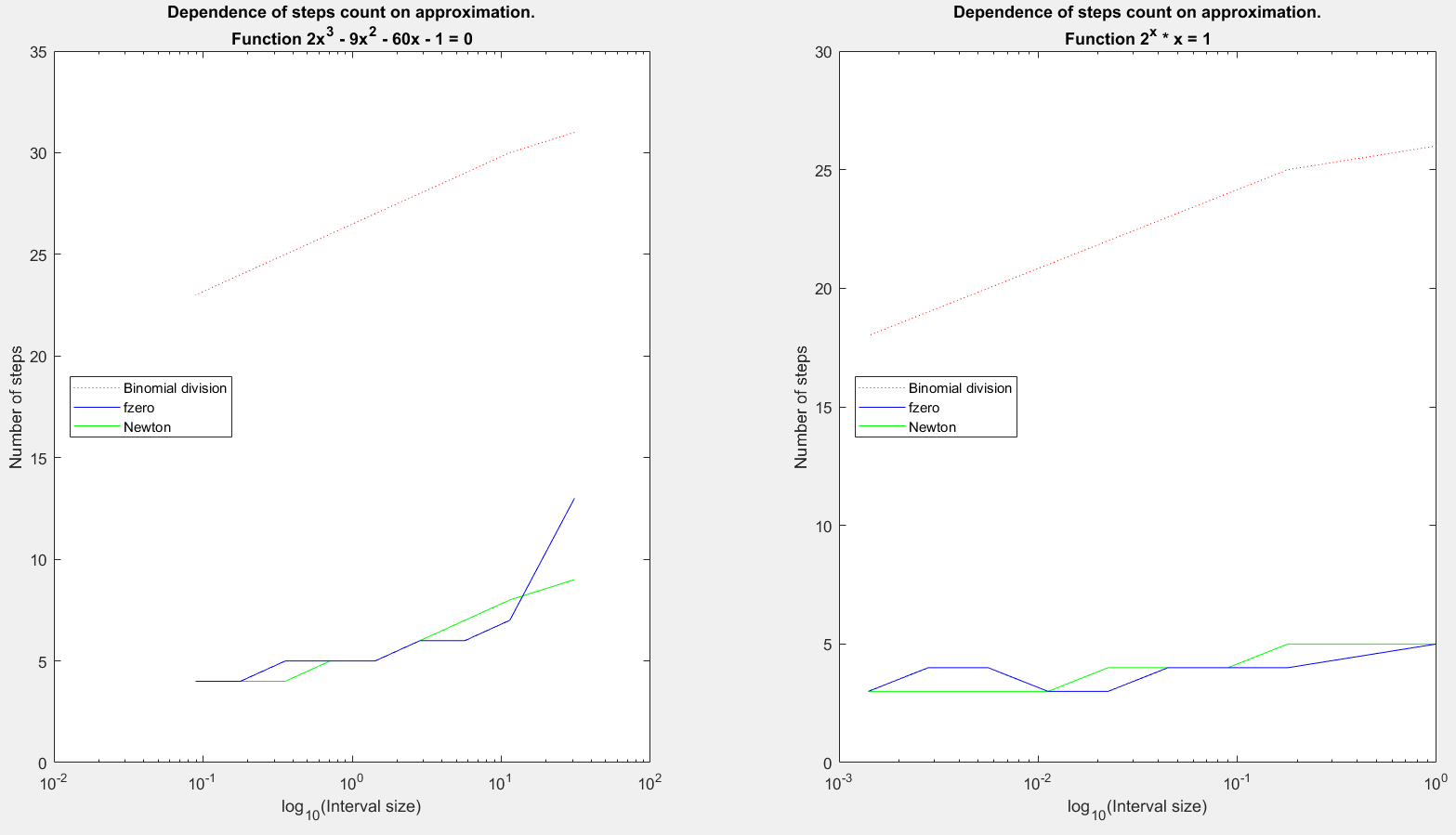


Рисунок 4. Зависимость числа шагов от начального приближения

Это также вытекает из свойств логарифма.

### Выводы о методе

Метод очень неприхотлив с точки зрения условий для применения – требуется лишь только непрерывность функции и наличие интервала, на концах которого значения функции имеют разные знаки. Время выполнения метода линейно зависит от показателя у и от начального размера интервала. По сравнению с методом Ньютона время выполнения оказывается в несколько раз больше.

## Решение методом Ньютона.

### Алгоритм.

Формируем последовательность , которая сходится к корню, следующим образом:

*(см. п.5 условий применения)*

Продолжаем работу, пока , где

### Условия сходимости.

### Тестовый пример.

Для уравнения решение методом Ньютона со стартовой точкой

выглядит следующим образом:

|  |  |  |  |
| --- | --- | --- | --- |
| *k* |  |  |  |
| 1 | 3 | 6 | 3 |
| 2 | 2 | 1 | 4 |
| 3 | 1.75 | 0.0625 | 3.5 |

### Подготовка контрольных тестов.

1. Полиномиальная и показательная функции, а также их произведение являются дважды непрерывно дифференцируемыми на области определения.
2. Знаки на разных концах интервала разные соответственно для каждой функции.
3. На участке полиномиальная функция имеет положительную первую и вторую производные.

Метод выполняем с начальным приближением , так как пятое условие сходимости выполняется на этом конце.

1. На участке трансцендентная функция имеет положительную первую и вторую производные.

Метод выполняем с начальным приближением , так как пятое условие сходимости выполняется на этом конце.

Данные замечания гарантируют корректность применения метода.

1. Для построения третьего графика были вычислены корни с точностью до . Выше описанные замечания соблюдаются для любого интервала, содержащего приближённый корень и вложенного в и в для соответствующих функций. Это необходимо для построения третьего графика.
2. Описанные выше условия также соблюдаются на интервале [-2;18] для полиномиальной функции и [-0.5; 1.5] трансцендентной – это необходимо для построения четвёртого графика.

### Геометрическая интерпретация.

На графике отмечены касательные, проведённые в точках . Видно, что очередная касательная проводится через точку, абсцисса которой совпадает с абсциссой точки пересечения предыдущей касательной с горизонтальной осью. Над точками подписаны их номера.

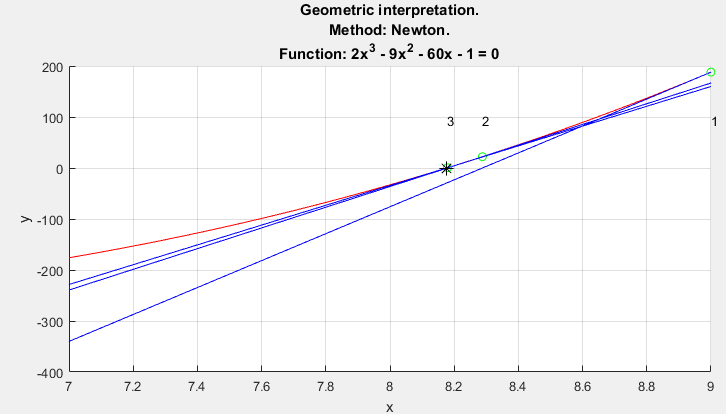


Рисунок 5. Геометрическая интерпретация метода Ньютона для функции

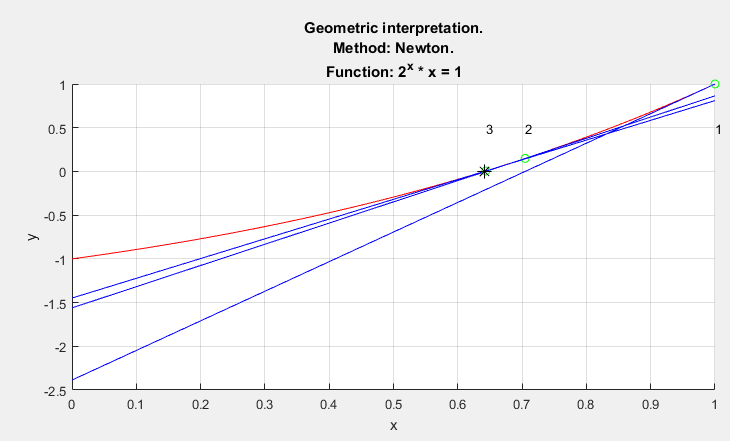


Рисунок 6 Геометрическая интерпретация метода Ньютона для функции

### Численный анализ решения.

Из геометрической интерпретации понятно, что чем меньше производная функции, тем быстрее метод сходится. Как видно из графиков (см. Рисунок 3. Зависимость количества итераций от заданной точности и Рисунок 4. Зависимость числа шагов от начального приближения, зелёным цветом), явной функциональной зависимости количества шагов от требуемой точности и начального приближения не наблюдается. Прослеживается только лишь то, что при повышении точности нужно больше шагов, а при увеличении начального приближения – меньше. То, насколько быстро число шагов меняется, зависит в основном от характера поведения производной функции в окрестности корня.

Для функции при поиске корня с точностью потребовалось 8 шагов, для функции – 4.

### Выводы о методе.

Метод накладывает очень серьёзные ограничения на использование. Для решения уравнений методом Ньютона требуется достаточно объёмный предварительный («на бумажке») анализ исследуемой функции. Сильным преимуществом является скорость сходимости – она очень близка к эталонной функции , а в одном случае оказалась даже больше (число шагов для функции ). Однако учитывая глубину предварительного анализа мне представляется более рациональным озадачиться минимизацией начального приближения для метода половинного деления.

## Несколько слов о функции fzero.

Как можно видеть на Рисунок 3 и Рисунок 4графиках, функция *fzero* своим поведением во многом напоминает метод Ньютона, если начальное приближение невелико (четвёртый график строился с начальным приближением около единицы для полинома и около 0.5 для трансцендентной функции). Однако если посмотреть на крайний правый отрезок третьего графика, то можно заметить неприятную тенденцию: в случае плохого (около 15) начального приближения *fzero* начинает себя вести хуже метода половинного деления, сложность которого является степенной. Из этого можно сделать вывод, что автоматизированный подбор оптимального численного метода, который применяется в *fzero,* нуждается в серьёзной доработке и на текущий момент при использовании данной функции желательно найти максимально хорошее начальное приближение.

## Модульная структура программы.

Программа состоит из файлов:

1. и – в каждом файле – соответствующая функция, относительно которой необходимо решать уравнение. Принимает на вход переменную , возвращает значение функции в точке .

**function y = f1( x )**

**function y = f2( x )**

1. – производные функций соответственно. Принимает на вход переменную , возвращает значение производной в точке .

function y = df1( x )

y = 6 .\* x .^ 2 - 18 .\* x - 60;

end

function y = df2( x )

y = 2 .^ x .\* (log(2) \* x + 1);

end

1. .m – файл содержит функцию численного решения методом половинного деления.

Принимает указатель на функцию f, для которой требуется решить уравнение f(x)=0, указатель на её производную (здесь не используется), интервал поиска и точность.

Возвращает приблизительный корень, затраченное число шагов и последовательность корней, получаемых на каждом шаге.

function [root, steps, appr\_roots] = BinomialDivision( f, df, a, b, eps )

1. – файл содержит функцию численного решения методом Ньютона;

Прототип такой же, однако смысл переменной – первый член последовательности, сходящейся к решению, – неиспользуемая величина. Возвращает то же.

function [x, steps, appr\_roots] = Newton( f, df, a, b, eps )

*Примечание*: прототипы методов решения сделаны одинаковыми для того, чтобы сделать общими для всех уравнений функции численного исследования метода.

Функция построения графика зависимости количества шагов от заданной точности. Принимает указатель на метод решения, функцию и её производную, границы решения, а также строковое представление формулы (для заголовка) и стиль графика.

function PrecisionGraphic( Method, f, df, Formula, a, b, style )

Функция построения графика зависимости количества шагов от начального приближения. Принимает указатель на метод решения, функцию и её производную, границы решения, а так же строковое представление формулы и стиль графика.

function ApproxGraphic( Method, f, df, EqFormula, a, b, style )

1. Построение геометрической интерпретации для каждого метода

Функция принимает указатель на метод решения, функцию и её производную, границы решения, а так же строковое представление формулы и стиль графика.

function GeomInterpretBinDiv( f, df, EqFormula, a, b, style )

function GeomInterpretNewton( f, df, EqFormula, a, b, style )

1. – запуск всех функций, вывод на экран решений и их характеристик (число шагов и приближённые корни).

# Прямые методы решения СЛАУ.

## Постановка задачи.

Пусть задана СЛАУ .

Требуется решить данную СЛАУ методом LDR-разложения, то есть найти такой , что , и исследовать зависимость погрешности корня от числа обусловленности матрицы.

## Решение методом LDR-разложения.

### Алгоритм.

1. Представим исходную матрицу *A* в виде произведения левой нижней унитреугольной (*L*), диагональной (*D*) и правой верхней унитреугольной (*R*) матриц. По теореме о существовании и единственности LDR разложения если все главные миноры матрицы *A* не равны нулю, разложение существует и единственно.

Общая схема такова: сначала выполняется LDR-разложение для матрицы 1х1, единственным элементом который является . Тогда

*.* Далее строится разложение (для матрицы

, построенной на первых k строках и k столбцах матрицы *A*)на основе уже имеющегося разложения. Эта процедура сводится к решению двух СЛАУ с левой нижней матрицей (достраиваются элементы , формулы см. в соответствующем замечании) и вычислению одного скалярного произведения (. Таким образом, увеличивая вплоть до , получаем LDR-разложение.

1. Получив разложение, получаем СЛАУ . Делая замену переменных и , решаем элементарные СЛАУ .

*Замечание о достраивании матриц*. Элементы находятся следующим образом:

*Замечание о решении элементарных СЛАУ.*

1. Решением системы с диагональной матрицей является вектор .
2. Решение системы с правой верхнетреугольной матрицей находится следующим образом: сначала очевидным образом вычисляется последний элемент вектора: Затем из уравнения вычисляется , и в общем виде получаем формулу

В случае унитреугольной матрицы .

По схожему принципу решается система с левосторонней верхнетреугольной матрицей. Отличие заключается в том, что в этом случае вычисления начинаются с первого элемента вектора и последующие вычисляются через предыдущие. Формула для вычисления -й компоненты выглядит следующим образом:

В случае унитреугольной матрицы .

### Условия применимости.

Для выполнения теоремы о существовании и единственности LDR-разложения требуется, чтобы все главные миноры матрицы системы были ненулевыми. Так как (, то , а, значит, и системы, упомянутые в 2.2.1.2, будут гарантированно иметь единственное решение. Поскольку решением последней системы является вектор – решение искомой СЛАУ, и оно единственно, мы гарантированно получаем единственный существующий вектор .

Таким образом для применимости метода необходимо выполнения единственного условия: все главные миноры не равны нулю.

### Тестовый пример.

.

1. .

Итого,

Решаем системы:

Ответ:

### Подготовка контрольных тестов.

Для тестирования программы решения СЛАУ методом LDR-разложения был подготовлен набор матриц размера 15, удовлетворяющих условию применимости метода.

Матрицы задавались следующим образом: сначала генерируется диагональная матрица с нужными собственными числами (имеет вид ). Затем она домножается слева и справа на ортогональную матрицу и обратную к ней. Поскольку такая процедура позволит нам получить матрицу с произвольными элементами и строго заданным числом обусловленности.

Правые столбцы задавались так: вектор без флуктуаций состоит из единиц (), флуктуации задавались путём добавления произвольной добавки из интервала в каждую компоненту вектора ().

После решения выполнялась следующая проверка: норма вектора невязки должна оказаться меньше : . Это делалось для того, чтобы убедиться в стабильности работы кода и невозможности фатальной ошибки. Для построения каждого из нижележащих графиков было решено 200 систем.

### Численный анализ решения.

Для того чтобы прояснить вопрос, насколько погрешность входных данных (правого столбца) может повлиять на погрешность решения, была исследована зависимость относительной погрешности вектора от числа обусловленности: вносилась одна и та же погрешность в правый столбец, а матрицы генерировались разные.

В результате получилась линейная зависимость в двойных логарифмических координатах, то есть степенная зависимость. По графику видно, что оценка является очень грубой: максимум имеет четвёртый порядок, а максимум – двенадцатый. При этом относительная погрешность правого столбца составляла примерно 0.02. Итого получили отличие на 6 порядков.

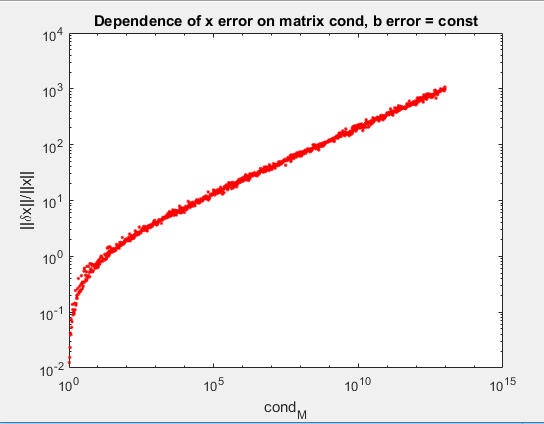


Рисунок . График зависимости погрешности корня от числа обусловленности матрицы при фиксированной погрешности правого столбца

Сложность алгоритма составляет:

1. На шагу разложения – решение трёх простейших СЛАУ размера k, два умножения матриц – , два умножения вектора на матрицу – .
2. Решение трёх простейших систем после разложения.

Сложность решения простейшей системы с матрицей размера составляет диагональной системы, для правой верхней и левой нижней унитреугольной систем.

Итоговая сложность -

### Выводы о методе.

Метод довольно неприхотлив – условия применения являются абсолютно естественными для данной задачи и не доставляют никаких неудобств, так как не накладывают дополнительных ограничений. Теоретически метод является точным. Но из-за высокой сложности метода (то есть большого количества вычислений с вещественными числами) мы получаем очень ощутимый вектор невязки (то есть погрешность решения), особенно у матриц с числом обусловленности порядка больше 10. Кроме того, мы обладаем очень грубым инструментом оценки погрешности решения – это отлично видно на графике.

В результате мы получаем метод, который теоретически претендует на то, чтобы называться точным, но на деле даёт решения с практически неконтролируемой погрешностью и показательной сложностью. Исходя из выше сказанного, я делаю вывод о том, что куда более целесообразным будет использовать приближённый численный метод, например метод релаксаций, для которого мы имеем хорошую оценку погрешности решения и более высокую (в хороших случаях сопоставимую с линейной) скорость сходимости.

## Модульная структура программы.

Подготовка входных данных осуществлялась в *MatLab* с помощью следующих функций:

1. Функция генерации матрицы размера NxN по формуле, описанной [здесь](#_Подготовка_контрольных_тестов.). Возвращает матрицу.

function M = genRandMatrix( N )

1. Функция генерации правого столбца *b* по формуле, описанной [здесь](#_Подготовка_контрольных_тестов.), со случайной флуктуацией

(добавляется, если ideal=1). Возвращает вектора *b* и – первым и вторым параметром соответственно.

function [v, ev] = genVector( N, ideal )

1. Функция подготовки СЛАУ с разными числами обусловленности матрицы и одинаковым правым столбцом. Записывает эти данные в файл. Возвращает полученные числа обусловленности и вектор погрешности правого столбца.

function [conds, ev] = prepareSLAEPlotCond()

1. Функция подготовки СЛАУ с разными порешностями правых столбцов и одной и той же матрицей. Записывает эти данные в файл. Возвращает полученные числа обусловленности и вектора погрешности правого столбца.

function [conds, ev] = prepareSLAEPlotXBerr()

Решение СЛАУ реализовано на языке Си.

1. Модуль операций с матрицами и векторами. Так как названия и аргументы говорящие, позволю себе не делать детального описания функций.

Общие пояснения: вектор – это одномерный массив (float \*), матрица – двумерный массив (float \*\*). Размеры матриц – MxN и NxK. Размеры векторов согласованы с размерами матриц.

float \* MatrMulVec( float \*\*A, float \*b, int M, int N );

float \* VecMulMatr( float \*b, float \*\*A, int M, int N );

float \* VecMulNum( float \*x, int N, float Num );

float \*\* MatrMulMatr( float \*\*A, float \*\*B, int M, int N, int K );

float \*\* Transposing( float \*\*A, int M, int N );

float DotProduct( float \*X, float \*Y, int N );

int IsEqual( float \*X, float \*Y, int N );

1. Модуль решения простейших СЛАУ – с диагональной, левой нижней треугольной и правой верхней треугольной матрицами.

void Diagonal( float \*\*A, float \*b, float \*x, int N );

void Left( float \*\*A, float \*b, float \*x, int N );

void Right( float \*\*A, float \*b, float \*x, int N );

1. Модуль LDR-разложения.

Функция разложения. Принимает раскладываемую матрицу А и указатели на матрицы L, D, R размера NxN.

void LDRDecomposition( float \*\*A, float \*\*\*L, float \*\*\*D, float \*\*\*R, int N )

Функция решения СЛАУ методом LDR-разложения.

void LDRSolve( float \*\*A, float \*b, float \*x, int N )

Обработка данных и построение графиков также осуществлена в *Matlab*.

Построение графика зависимости погрешности правого столбца от произведения числа обусловленности на погрешность решения.

function drawPlot()

# Итерационные численные методы решения СЛАУ.

## Постановка задачи.

Пусть задана СЛАУ . Требуется решить данную систему уравнений методом релаксации с заданной точностью , то есть найти такой вектор , что где

## Решение методом релаксации.

### Алгоритм.

Метод релаксации основан на методе Зейделя, который в свою очередь является модификацией метода Якоби. В связи с этим считаю, что имеет смысл сказать пару слов об упомянутых методах.

* + - 1. *Метод Якоби*.

Матрица системы представляется в виде (обозначения L, D, R оговорены в п. [2.2.1](#_Алгоритм.)). Сделаем следующие преобразования:

Таким образом, обозначив , можно записать СЛАУ в каноническом виде для решения итерационными методами и составить последовательность :

Матрица имеет вид , где

* + - 1. *Метод Зейделя*.

Как уже было сказано, метод Зейделя является развитием идеи метода Якоби. Модификация состоит в том, что -я компонента -го вектора выражается через уже полученные:

* + - 1. *Метод релаксации.*

Будем искать очередной элемент последовательность в виде

, где – приближение к решению заданной СЛАУ, – параметр релаксации.

После выражения вектора как приближённое решение по методу Зейделя и подстановки в (\*) получим:

Это и есть конечная формула для вычисления -й компоненты -го приближения.

Сам алгоритм выглядит следующим образом: последовательность по формуле (\*) строится, пока . По выходе из цикла возвращается последний член последовательности.

### Условия сходимости.

Метод релаксации сходится, если матрица А является нормальной (то есть симметричной и положительно определённой) и параметр релаксации лежит в интервале .

### Тестовый пример.

. Параметр релаксации , начальное приближение Матрица записана с точностью до второго знака. К сожалению, для заданных условий сходимости не получается легко подобрать матрицу, чтобы она была целочисленной. О том, как генерировалась данная матрица, можно прочитать [здесь](#_Замечание_о_генерации).

Вычислим

Вычислим

Вычислим

Видно, что уже на третьей итерации мы получили правильное приближённое решение с точностью до первого знака, и очень близко подобрались к тому, чтобы достичь точности до сотых. Уже на тестовом примере понятно, что при данном параметре релаксации мы будем получать очень высокую скорость сходимости.

### Подготовка контрольных тестов.

Для исследования любого итерационного метода необходимо понять, как меняется его скорость сходимости в зависимости от начального приближения, требуемой точности и определителя. Кроме того, нужно понять, как меняется фактическая точность при снижении определителя до значений, близких к погрешности вещественных вычислений.

Конкретно для метода релаксации также интересно посмотреть, как меняется число шагов при изменении параметра релаксации.

Для того чтобы произвести эти исследования, в первую очередь необходимо сгенерировать матрицу, удовлетворяющую условию сходимости метода.

Это делалось с помощью функции *sprandsym,* которая принимает долю ненулевых коэффициентов в матрице, её размер (в нашем случае размер матрицы равен 15) и набор собственных чисел, которыми должна обладать выходная симметричная матрица.

Для исследования зависимости скорости сходимости от начального приближения генерировались векторы по формуле , то есть все компоненты *k*-го начального вектора равны *k.* При этом матрица была единой для всех тестов, количество которых равнялось 20. Правый столбец построен по одинаковому принципу для всех тестов – так, чтобы точное решение было равно .

Параметр релаксации был выбран 1.2, точность вычислений – .

Для исследования зависимости от точности было выбрано начальное приближение , параметр релаксации, матрица СЛАУ и правый столбец – те же. Измерения проводились в диапазоне , каждый следующий тест имеет точность в 10 раз меньше предыдущего (сначала 0.01, затем 0.001 и так далее).

Исследование зависимости от параметра релаксации проводилось для с шагом 0.1. Матрица и правый столбец – те же, точность .

Исследование зависимости скорости сходимости от определителя проводилось двумя способами. В первом сгенерированная предварительно матрица умножалась на число , где – определитель предварительно сгенерированной матрицы, а – требуемый. Такая операция даёт желаемы результат, так как умножение на число одной строки увеличивает определитель в раз, соответственно умножение всей матрицы – в раз. Во втором способе матрица заново генерировалась на каждом шаге, а всё остальное делалось так же. Таким образом мы исследуем уже не одну и ту же по-разному нормированную матрицу, а уже совершенно разные объекты.

### Замечание о генерации целочисленной положительно определённой матрицы*.*

Почему же нельзя было сгенерировать целочисленную матрицу, удовлетворяющую условиям сходимости? Опишем механизм, по которому создаётся такая матрица.

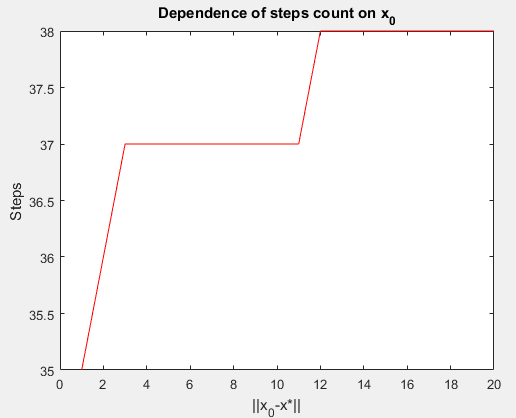
Если матрица *A* является симметричной и обладает собственными числами , то она может быть представлена в виде , где Q – ортогональная матрица, состоящая из соответствующих собственных векторов, а

.

Матрица А ищется именно в таком виде: составляется матрица D, а затем требуется составить ортогональную целочисленную матрицу. Эта задача является трудоёмкой и содержит большую долю подбора. В связи с этим было принято решение сгенерировать ортогональную матрицу, содержащую вещественные числа (например по формуле , а для показа тестового примера пренебречь точностью.

### Численный анализ решения.

Зависимость скорости сходимости от начального приближения представляет собой очень медленно возрастающую функцию:

  
Рисунок . Зависимость числа шагов от начального приближения

Это можно объяснить тем, что метод релаксаций, являясь разновидностью метода простых итераций (МПИ), очень похож на МПИ для решения уравнений в смысле поиска нормы вектора решения: большинство теорем, сформулированных применительно к МПИ решения уравнений для производной функции, формулируются в матричном варианте МПИ для нормы матрицы. Таким образом, можно объяснить высокую, практически не меняющуюся при начальном приближении, скорость получения правильной нормы, ссылаясь на МПИ решения уравнений. А так как общий вид искомого вектора совпадает с начальным, каждая отдельная компонента не будет очень сильно изменяться и «выбиваться» из общего вида.

Зависимость числа шагов от параметра релаксации получилась вполне предсказуемой:

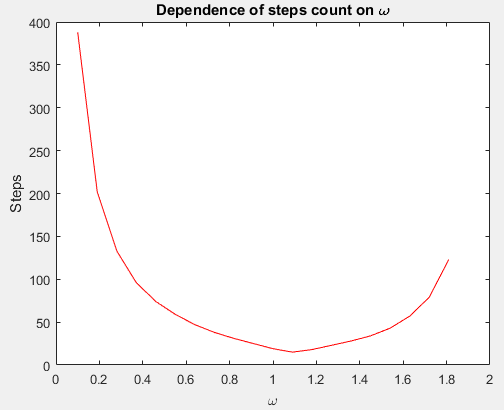


Рисунок . Зависимость числа шагов от параметра релаксации

Ожидаемо получено высокое количество шагов вблизи нуля и двух (при и метод, вообще говоря, будет расходиться). Также вполне предсказуемо получен минимум недалеко от единицы – таким образом подтверждается, что метод релаксации в некоторой степени улучшает метод Зейделя (при получаем метод Зейделя).

Также не вызывает удивлений получаемая зависимость от точности корня:

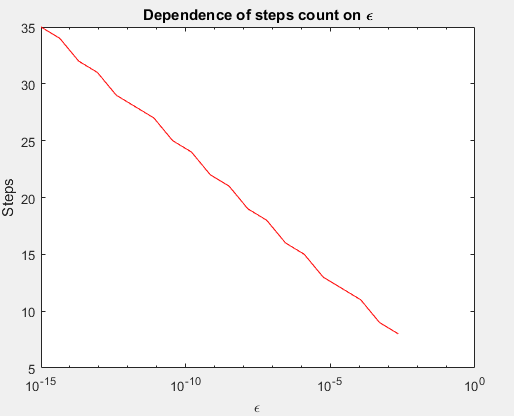


Рисунок . Зависимость числа шагов от точности корня

Общий вид корня при решении данным методом угадывается уже на второй–третьей итерации. Дальше происходит небольшая корректировка значений, которая в некоторой степени напоминает бинпоиск: если при вычислении очередного члена последовательности одно из слагаемых в формуле (\*) оказалось слишком большим, то на следующей итерации за его же счёт произойдёт снижение нескольких компонент, и так до тех пор, пока не будет выполнено условие выхода.

Для бинпоиска зависимость от точности имеет ровно такой вид, как мы получаем на графике 10.

При первом способе исследования метод показал очень приятную зависимость скорости сходимости от определителя:

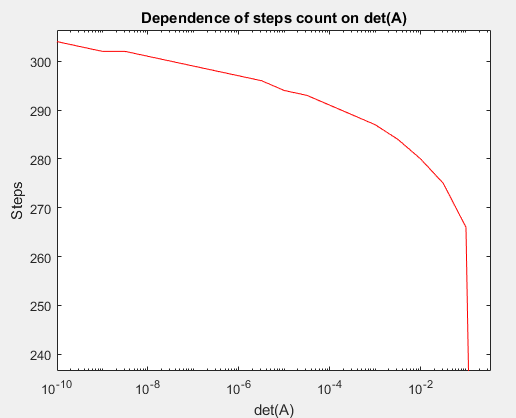


Рисунок . Зависимость числа шагов от определителя матрицы. Метод 1

Видно, что метод очень устойчив: при понижении определителя на 8 порядков число шагов увеличилось всего лишь на 35.

Для того чтобы убедиться, что фактическая точность остаётся удовлетворительной, был также построен соответствующий график:

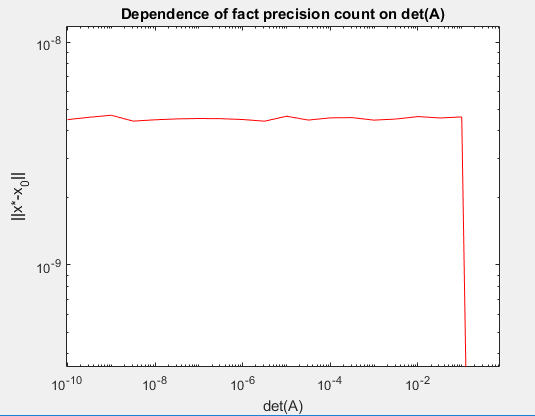


Рисунок . Зависимость фактической точности от определителя. Метод 1

Видно, что фактическая точность является очень стабильной и остаётся практически постоянной при любом определителе.

При втором способе исследования всё не так однозначно:

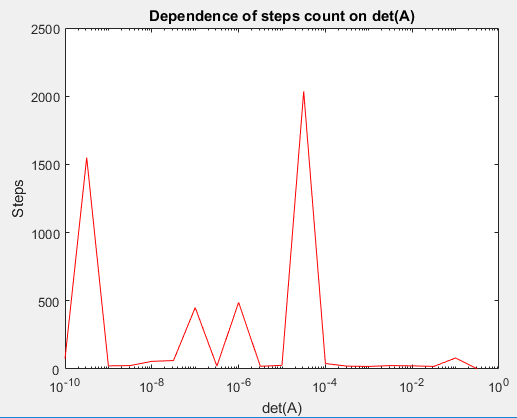


Рисунок . Зависимость числа шагов от определителя. Метод 2

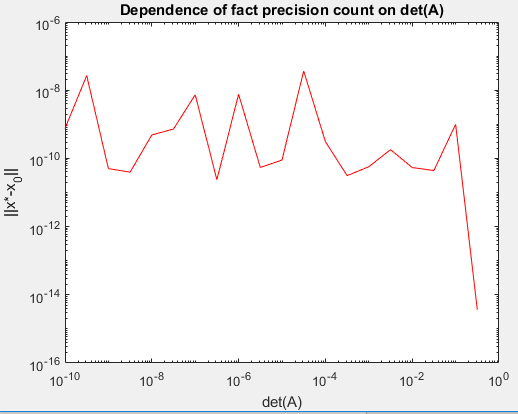


Рисунок . Зависимость фактической точности от определителя. Метод 2

Видно, что появилась, быть может, не совсем однозначная, но весьма яркая тенденция к ухудшению скорости сходимости при некоторых малых определителях матрицы. Стоит отметить, что нет монотонного увеличения числа итераций при уменьшении определителя. Скорее всего, наблюдаемые скачки связаны со значением определителя косвенно: вероятно, у сгенерированных с этими определителями матриц число обусловленности сильно превосходит числа обусловленности остальных матриц. Эту идею подтверждает полученная зависимость фактической точности от определителя – пики на её графике совпадают с пиками на графике зависимости числа шагов от определителя. В [п. 2](#_Численный_анализ_решения._2) мы убедились, что погрешность решения увеличивается с ростом числа обусловленности. Хоть погрешность решения и фактическая точность – несколько разные вещи, они имеют схожую природу, поэтому можно посчитать вышеизложенное предположение вполне разумным.

Стоит заметить при этом, что в областях, где скачков нет, зависимость практически постоянная, то есть ни фактическая точность, ни число шагов особенно не изменяются.

### Выводы о методе.

Метод оставил крайне положительное впечатление: код для него пишется очень легко, не требует длительного процесса отладки. При правильно подобранном параметре релаксации метод обладает высокой скоростью сходимости (десятки итераций вместо тысяч для прямых методов с кубической сложностью – улучшение на два порядка). При этом метод релаксации является очень устойчивым, хорошо работает на определителях вплоть до сопоставимых с вычислительной погрешностью и не имеет тенденции к хаотичному поведению в предельных случаях. Необходимо лишь следить за числом обусловленности матрицы.

Единственным существенным недостатком метода являются жёсткие условия использования – матрица должна быть нормальной. Это очень сильно сужает его сферу применения.

Таким образом, можно смело рекомендовать этот метод для приближённого решения систем с симметричной положительно определённой матрицей.

## Модульная структура программы.

Подготовка входных данных осуществлялась в *MatLab* с помощью следующих функций:

1. Функция генерации матрицы размера NxN по методу, описанному [здесь](#_Подготовка_контрольных_тестов._1). Возвращает матрицу.

function M = genRandMatrix( N )

1. Функция генерации правого столбца *b* по формуле, описанной [здесь](#_Подготовка_контрольных_тестов._1).

Принимает размер вектора и число, которым требуется заполнить все его компоненты.

function [v] = genVector( N, k )

1. Функция подготовки и записи в файл необходимых для первых трех исследований данных: вычислительная точность, параметр релаксации, начальное приближение, матрица, правый столбец. Принимает количество систем, которое требуется сгенерировать. Возвращает записанные в файл начальные данные – массив векторов начальных приближений, массив параметров .

function [x0, omega, epsilon] = prepare( testCnt )

1. Функция, которая записывает в файл те же самые параметры СЛАУ, но подготавливает матрицы для последних двух исследований (с определителем). Также принимает число тестов. Возвращает массив определителей сгенерированных матриц.

function dets = prepare2( testCnt )

Решение СЛАУ реализовано на языке C++.

В библиотеке работы с матрицами произошли незначительные изменения: все матрицы теперь имеют тип std::vector<std::vector<double>>, векторы – std::vector<double>.Все переменные передаются по ссылке. В остальном функции не поменялись (основу можно посмотреть [здесь](#_Модульная_структура_программы.)).

Функция, решающая СЛАУ методом релаксаций. Принимает матрицу, правый столбец, начальное приближение, параметр релаксации, требуемую точность и ссылку на переменную, в которую будет записано число итераций.

std::vector<double> Relax( std::vector<std::vector<double>> const &A,

std::vector<double> const &b, std::vector<double> const &x0, double Omega, double Epsilon, int &Steps )

Обработка данных происходит в *Matlab*.

1. Функция построения первых трёх графиков. Параметры по названию соответствуют описанным в подпункте о подготовке данных. Все параметры относятся к тем системам, решение которых лежит в загружаемом функцией файле.

function drawPlot( testCnt, x0, omega, epsilon )

1. Функция построения последних двух графиков. Принимает число тестов и массив определителей матриц, решение систем с которыми лежит в загружаемом функцией файле.

function drawPlot2( testCnt, dets )

# Итерационные методы решения частичной АПСЗ.

## Постановка задачи.

Пусть – матрица простой структуры. Требуется найти приближённое значение её минимального собственного числа с заданной точностью , то есть такое число что – её минимальное собственное число.

## Метод обратных итераций.

### Алгоритм.

Метод обратных итераций основан на степенном методе, который позволяет найти максимальное собственное число заданной матрицы.

Степенной метод состоит в следующем:

Пока , строятся последовательности и

При этом (иначе и – на него происходит деление).

В таком случае ( – максимальное собственное число), ( – соответствующий собственный вектор).

Метод обратных итераций основан на следующем свойстве обратных матриц: если , то .

То есть если вместо матрицы *А* мы подставим матрицу , то найдём её максимальное собственное число, то есть (.

При этом изменяется условие останова: вышеописанный цикл выполняется, пока , так как к минимальному собственному числу обратной матрицы теперь стремится последовательность .

Кроме того, формула (1) меняется на: , что эквивалентно

. То есть на каждом шаге требуется решить СЛАУ.

В нашем случае для решения СЛАУ был использован точный метод -разложения матрицы: было произведено однократное предварительное разложение, а затем на каждом шаге решалось три элементарных системы – с левой нижней унитреугольной, дигональной и правой верхней унитреугольной матрицами (подробнее см. [п. 2](#_Алгоритм.)).

Для ускорения метода производится сдвиг спектра: вычисление минимального значения собственного вектора происходит для матрицы . В таком случае спектр сдвигается влево на . Так как скорость сходимости зависит от отношения предмаксимального и максимального собственного значений, то для обратной матрицы будет играть ключевую роль отношение . Можно подобрать сдвиг так, чтобы .

### Условия сходимости.

Метод обратных итераций сходится, если матрица – простой структуры, то есть все собственные числа вещественны и различны.

Стоит заметить, что если спектр матрицы содержит отрицательную часть, то будет найдено минимальное *положительное* собственное число (так как при делении единицы на собственное число матрицы знак сохраняется, соответственно, если у исходной матрицы спектр был, например, , то у обратной спектр будет и алгоритм вернёт 4, а не -2).

В этом смысле для сходимости метода требуется положительная определенность матрицы, хотя метод сойдётся и без этого условия (правда, к минимальному положительному собственному числу).

### Дополнительное замечание о сходимости.

В поставленной задаче требуется использовать сдвиги для ускорения сходимости. Однако можно использовать сдвиг для того, чтобы сделать весь спектр положительными и не потерять отрицательные собственные числа. Сдвиг, который будет гарантировать положительность, равен любой норме матрицы.

### Тестовый пример.

Рассмотрим процесс нахождения минимального собственного числа матрицы из [тестового примера п. 2](#_Тестовый_пример.). Её спектр равен c точностью до десятых } – то есть это как раз только что описанный случай.

Для данной матрицы уже получено LDR-разложение:

Мы не будем использовать сдвиг в тестовом примере, так как всё равно на трёх-четырёх итерациях не будет заметно значительное ускорение.

.

Итерация 1.

Решаем систему – система была полностью решена [здесь](#_Тестовый_пример.).

.

*.*

Итерация 2.

Решаем систему . Процесс решения систем уже показывать не будем, поскольку он был продемонстрирован ранее.

Итерация 3.

Решаем систему .

Итерация 4.

Решаем систему .

(здесь обыкновенные дроби становятся слишком сложными, поэтому вычисления были поручены программе).

– уже на четвёртой итерации мы получили собственное число с точностью до десятых.

### Подготовка контрольных тестов.

Для исследования метода были подготовлены тесты, с помощью которых были построены зависимости:

1. числа итераций от требуемой точности собственного числа;
2. числа итераций от сдвига;
3. числа итераций от отделимости;
4. фактической точности от отделимости.

Для выполнения тестов 1, 2 с помощью функции была сгенерирована матрица размера 15х15, в которую передавался набор положительных собственных чисел .

Для третьего и четвёртого теста генерировался набор матриц, в котором вместо собственного числа *2* вводились числа по формуле v0(1) + v0(1) \* 10 ^ (-14.0 \* (i - 1) / (testCnt - 1)), где *testCnt* – число тестов – равно 20, , – минимальное собственное число матрицы – *1*. Таким образом с каждым тестом второе собственное число всё сильнее приближалось к первому.

Второй, третий и четвёртый тесты выполнялись с точностью поиска собственного числа . В первом тесте точность менялась по формуле 10 ^ (-15 \* (i - 1) / (testCnt – 1) + -2 \* (1 – (i – 1) / (testCnt – 1))).

*Примечание*: в показателе числа *10* – формула линейной интерполяции от -15 к 2. Такая формула задана, чтобы вне зависимости от числа тестов равномерно покрыть интервал показателей от -15 до -2.

В первом, третьем и четвёртом тесте использовался нулевой сдвиг. Во втором тесте сдвиг вычислялся по формуле (v0(N) \* 0.9) \* (1 - double(i - 1) / (testCnt - 1), то есть метод был протестирован при значениях сдвига, полученных равномерном разбиением отрезка от 0 до .

### Численный анализ решения.

Зависимость скорости сходимости от требуемого приближения ожидаемо получилась линейной в полулогарифмических координатах. Это связано с тем, что скорость сходимости определяется соотношением , где *Steps* – число шагов, то есть скорость сходимости.

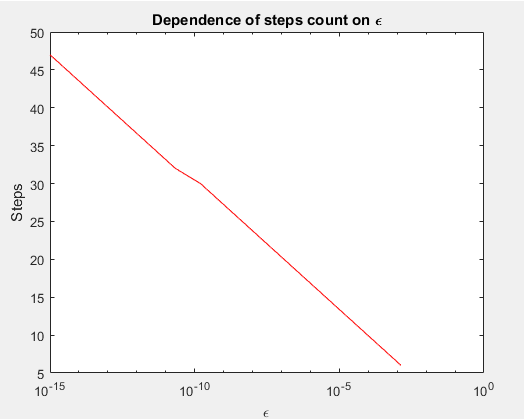


Рисунок . Зависимость числа шагов от заданной точности. Сдвига нет, specA={1,2,...,15},

При построении графика зависимости скорости сходимости от сдвига было ожидаемо выявлено, что оптимальное значение сдвига лежит рядом с минимальным собственным числом (но не равно ему!).



Рисунок . Зависимость скорости сходимости от сдвига. specA={1-shift,2-shift,...,15-shift},

В п. 4.2.5 было указано, что исследовались сдвиги в интервале от 0 до 15, однако этот график целиком не представляет интереса, так как далее начинается очень быстрое понижение скорости сходимости, и на минимум в окрестности *-1* просто перестаёт быть заметным.

График зависимости числа шагов от отделимости в целом получился возрастающим: плохая отделимость собственных чисел матрицы соответствует хорошей отделимости собственных чисел обратной матрицы. Однако имеется странная аномалия при отделимости : число шагов сильно больше, чем при отделимости в . То есть всю тенденцию портит только одна эта точка.

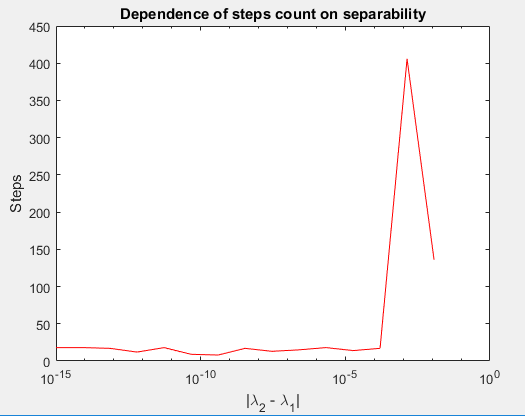


Рисунок . Зависимость числа шагов от отделимости. сдвига нет.

График зависимости фактической точности от отделимости тоже в целом совпал с ожиданием: при плохой отделимости фактическая точность удовлетворительна, при большой – выходит за границы заданного . Есть одна аномалия – при отделимости фактическая точность резко улучшилась, а затем вернулась к общей тенденции.

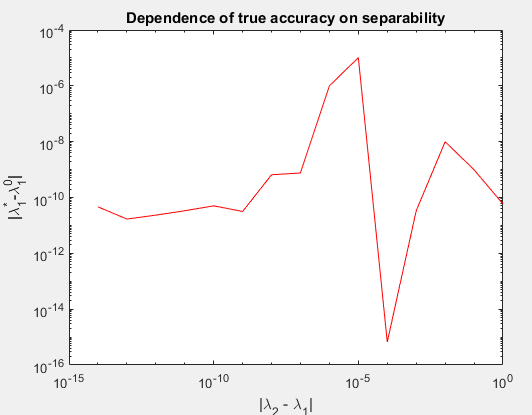


Рисунок . Зависимость фактической точности от определителя. сдвига нет.

### Выводы о методе.

Метод подкупает простотой своей реализации. Однако есть некоторые моменты (например условие останова), которые неочевидны на первый взгляд и из-за которых реализация может растянуться на долгое время. Кроме того, нет ощущения, что метод работает стабильно – то и дело встречаются какие-то труднообъяснимые аномалии.

Несмотря на то, что функциональность метода невелика – он ищет минимальное положительное собственное число – можно легко её расширить. Так, можно с помощью данного метода легко найти минимальное по модулю отрицательное собственное число, взяв для рассмотрения вместо матрицы матрицу .

Очень осторожного обращения требуют сдвиги – надо подбирать их так, чтобы случайно сдвиг не оказался равен собственному числу – иначе метод разойдётся. Учитывая эту особенность, нужно иметь в виду, что ситуация расхождения метода является своего рода сигналом к тому, что мы случайно нашли собственное число. Проверить этот факт нужно, выбрав сдвиг, близкий к тому, при котором метод разошёлся.

На мой взгляд, такая особенность метода не является положительной, поскольку решение АПСЗ не должно требовать человеческого вмешательства.

Резюмируем: метод обладает низким функционалом и требует большой осторожности в использовании.

## Модульная структура программы.

Подготовка входных данных осуществлялась в *MatLab* с помощью следующих функций:

1. Функция генерации матрицы размера NxN по методу, описанному [здесь](#_Подготовка_контрольных_тестов._2). Возвращает матрицу.

function M = genRandMatrix( N )

1. Функция подготовки и записи в файл необходимых для исследований данных: вычислительная точность, сдвиг, матрица. Принимает количество тестов для каждой зависимости. Возвращает данные, необходимые для исследований – вторые во величине собственные числа (для построения третьей и четвертой зависимости), вычислительная точность (для первой зависимости), параметра сдвига (для второй зависимости).

function [v2, epsilon, shifts] = prepare( testCnt )

Решение частичной АПСЗ реализовано на языке C++.

Была сделана обёртка над стандартным классом – появились классы *matr* и *vec*, в которых были перегружены стандартные математические операторы – умножение матрицы на вектор, вектора на матрицу, сложение матриц, сложение векторов и т. д.

1. Функция, решающая частичную АПСЗ методом обратных итераций. Принимает матрицу, требуемую точность и сдвиг. Возвращает структуру eigen, содержащую собственное число, собственный вектор и число затраченных итераций.

struct eigen {

double Value;

vec Vector;

int Steps;

};

eigen invIt( matr const &A, double eps, double shift )

1. Функция решения АПСЗ методом обратных итераций со сдвигом. Принимает и возвращает то же, что и предыдущая функция. Создаёт матрицу со сдвинутым спектром и передаёт её для решения в функцию *invIt*. К возвращённому собственному числу прибавляет сдвиг.

eigen invItWithShift( matr const &A, double eps, double shift )

Обработка данных происходит в *Matlab*.

1. Функция построения графиков всех зависимостей. Параметры по названию соответствуют описанным в подпункте о подготовке данных. Все параметры относятся к тем матрицам, решение АПСЗ которых лежит в загружаемом функцией файле.

function drawPlot( testCnt, v2, epsilon, shift ).