(Место для титульника)

[Часть 1. Решение алгебраических и трансцендентных уравнений 4](#_Toc531681139)

[Постановка задачи 4](#_Toc531681140)

[Использующиеся методы решения 4](#_Toc531681141)

[Метод половинного деления 4](#_Toc531681142)

[Метод секущих 5](#_Toc531681143)

[Структура программы 6](#_Toc531681144)

[Решение уравнений 7](#_Toc531681145)

[Алгебраическое уравнение 7](#_Toc531681146)

[Трансцендентное уравнение 9](#_Toc531681147)

[Анализ результатов 10](#_Toc531681148)

[Иллюстрация метода секущих 11](#_Toc531681149)

[Часть 2. Решение СЛАУ прямыми методами 12](#_Toc531681150)

[Постановка задачи 12](#_Toc531681151)

[Алгоритм применения метода 12](#_Toc531681152)

[Условия применимости метода 13](#_Toc531681153)

[Тестовый пример с ручным расчётом 13](#_Toc531681154)

[Задание матрицы 13](#_Toc531681155)

[Модульная структура программы 14](#_Toc531681156)

[Вычисление коэффициентов для «больших» матриц 15](#_Toc531681157)

[С плохим cond(A) 15](#_Toc531681158)

[С хорошим cond(A) 15](#_Toc531681159)

[Анализ результатов 15](#_Toc531681160)

[Часть 3. Решение СЛАУ итерационными методами. Метод Зейделя 16](#_Toc531681161)

[Постановка задачи 16](#_Toc531681162)

[Условия применимости метода 16](#_Toc531681163)

[Условие остановки 16](#_Toc531681164)

[Тестовый пример 17](#_Toc531681165)

[Модульная структура программы 17](#_Toc531681166)

[Решение СЛАУ с матрицами порядка 10 18](#_Toc531681167)

[Матрица с хорошим числом обусловленности 18](#_Toc531681168)

[Матрица с плохим числом обусловленности 18](#_Toc531681169)

[Для матрицы с определителем, близким к нулю 19](#_Toc531681170)

[Анализ результатов 19](#_Toc531681171)

[Глава 4. Решение алгебраической проблемы собственных значений 20](#_Toc531681172)

[Постановка задачи 20](#_Toc531681173)

[Алгоритм степенного метода 20](#_Toc531681174)

[Условия применимости метода 20](#_Toc531681175)

[Тестовый пример 21](#_Toc531681176)

[Контрольные тесты 21](#_Toc531681177)

[Модульная структура программы 22](#_Toc531681178)

[Выводы 22](#_Toc531681179)

Часть 1. Решение алгебраических и трансцендентных уравнений

Постановка задачи

Требуется найти приближённые решения двух уравнений: алгебраического и трансцендентного, - используя для этого численные методы алгебры. Способы решения: двоичный поиск корня на промежутке и модификация метода Ньютона - метод секущих.

Численные методы алгебры позволяют найти приближенные решения уравнений при заданной точности за некоторое конечное число шагов (итераций). В данной части работы будут описаны выше названные методы решения уравнений, с их помощью будет приведено решение двух уравнений, а также приведён анализ результатов: сравнение числа итераций при заданной точности для каждого из методов.

Использующиеся методы решения

При описании методов считается, что промежутки, содержащие единственный корень уравнения, уже отделены.

Метод половинного деления

; ; Существование и единственность корня проверяется следующими условиями: и знакопостоянна на .

Тогда корень гарантированно попадает в промежуток или , или

Для полученного промежутка проделываем те же действия. Таким образом, область, в которую попадёт корень, сужается со скоростью , где k – число шагов.

Условие остановки: , где ɛ - желаемая точность результата. Тогда является приближённым решением для заданной точности ɛ. Обычно берется .

Пример:

Решим уравнение . . Очевидно, уравнение имеет точные корни и . Найдём методом половинного деления с точностью .

; f(1.7) = -0.21; f(3) = 2.

.

.

Заметим, что на этом этапе длина очередного промежутка составляет . Тогда корнем с точностью является любая точка промежутка, возьмем середину: .

Метод секущих

. знакопостоянные и непрерывные, (условия существования и единственности корня). Найдём методом секущих.

.

.

: .

Геометрическая интерпретация формулы (2) объясняет название метода – через две точки и строится секущая к функции f(x), которая пересекает ось OX в точке , рассматриваемой в качестве следующего приближения. Условия выбора начальных двух точек x0 и x1 обеспечивают сходимость итерационного процесса и «подход» к корню только с одной стороны промежутка. Условие остановки: . - точность, и вычисляются в точках на самих границах – a и b, так как по условию промежуток уже такой, что на нём монотонные.

Пример:

. Вычислим . .

.

.

; . Заметим, что , значит, является приближенным корнем уравнения с точностью .

Структура программы

Программа содержит следующие модули:

double fCount(double x0); //Посчитать значение трансцендентной функции

double pCount(double x0); //Посчитать значение конкретного полинома

double df(double f(double), double x0); //Посчитать первую производную

double d2f(double f(double), double x0); //Посчитать вторую производную

double findRoot(double epsilon, double leftBorder, double rightBorder, double f(double), int\* count, error\_t\* error);

//Найти нуль функции

double findRootHalfDiv(double epsilon, double leftBorder, double rightBorder, double f(double), int\* count, error\_t\* error);

//Найти корень методом половинного деления

void findRootsWithUserInterface(double f(double)); //Найти корень уравнения, используется дружелюбный пользовательский интерфейс

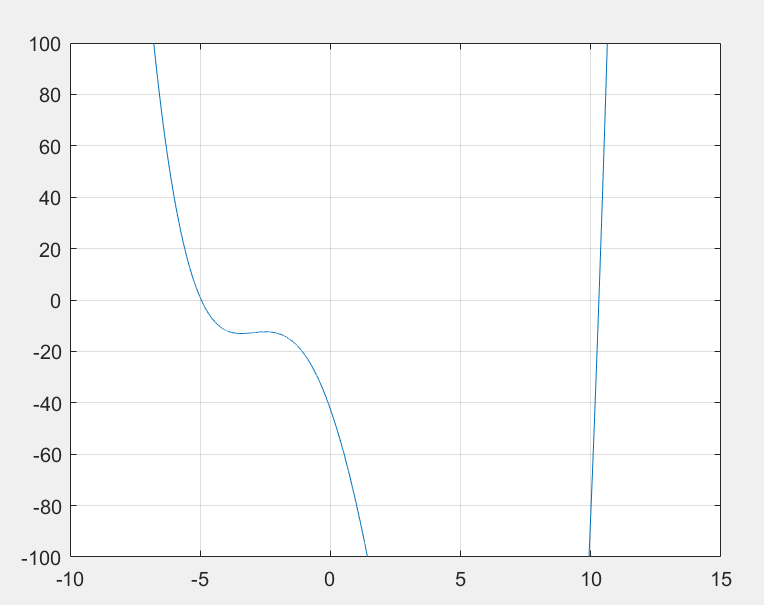
Решение уравнений

Алгебраическое уравнение

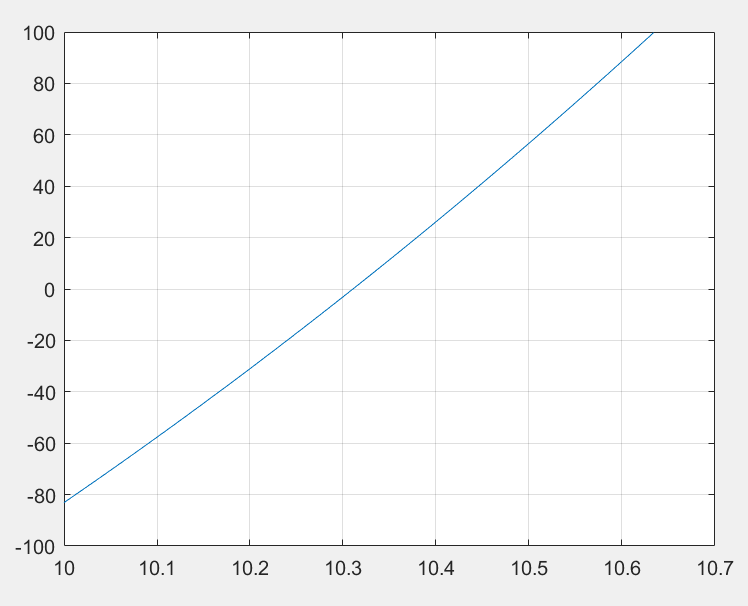
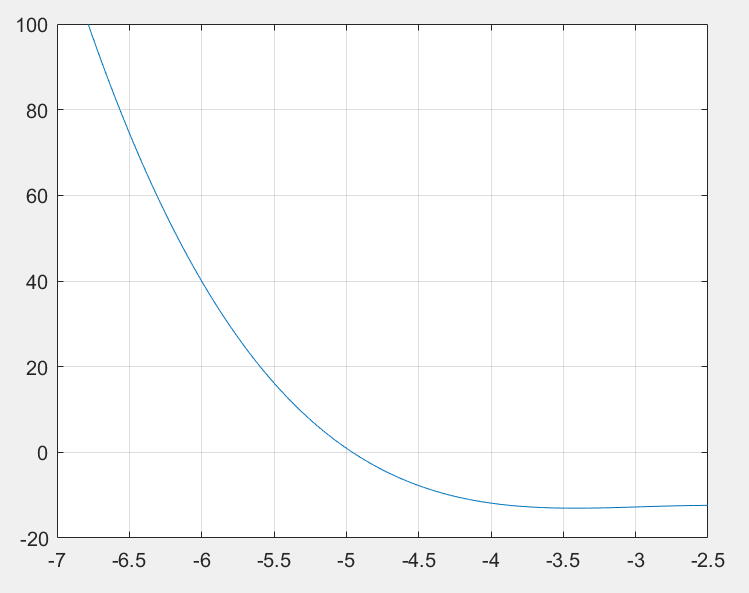
Требуется решить алгебраическое уравнение: . Отделим корни графически, воспользуемся теоремой о границе корней полинома.

Найдем верхнюю границу положительных корней по формуле: , где m – номер первого отрицательного из , . Получаем верхнюю границу 348,17402. Воспользуемся этой же теоремой, для нахождения нижней границы отрицательных корней, выполнив замену переменной: : . Выполнив подстановку и вернувшись к старой переменной, получаем нижнюю границу корней: -19,633.

Построим график функции, используя пакет MatLab:



Функция имеет 2 действительных корня. Отделим корни:



Выберем промежутки [-6; -4] и [10; 10.6], на них выполнены все требуемые начальные условия для обоих методов.

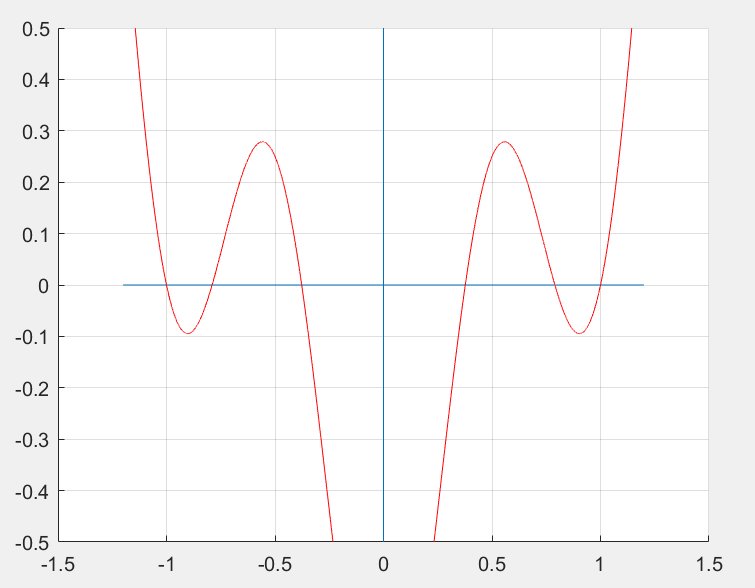
Найдём корни на промежутках с точностью . Для промежутка [-6; -4]: x\* = -4.958715

|  |  |  |
| --- | --- | --- |
|  | Алгебраическое уравнение | |
| Метод секущих | Метод половинного деления |
| 0,001 | 6 | 11 |
| 0,0001 | 7 | 15 |
| 0,000001 | 10 | 21 |

Для промежутка [10; 10.6]: x\* = 10.310747;

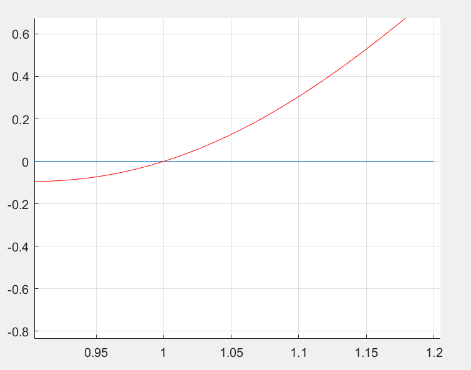
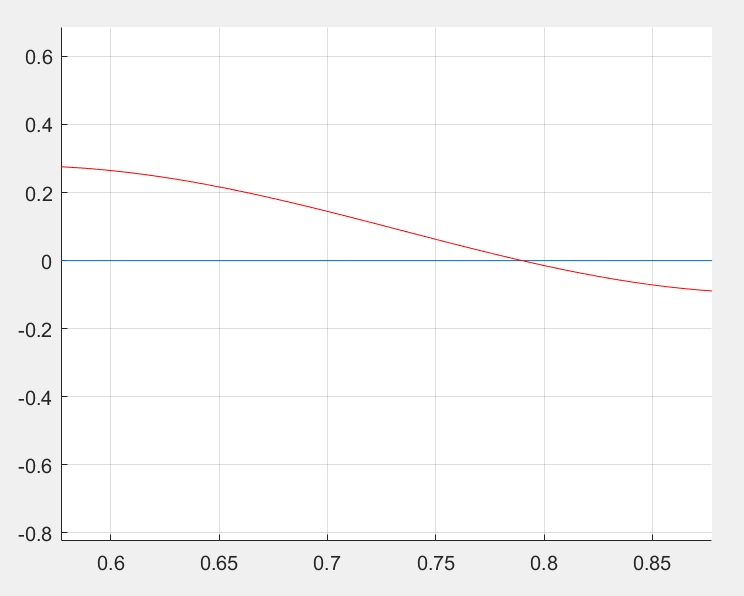
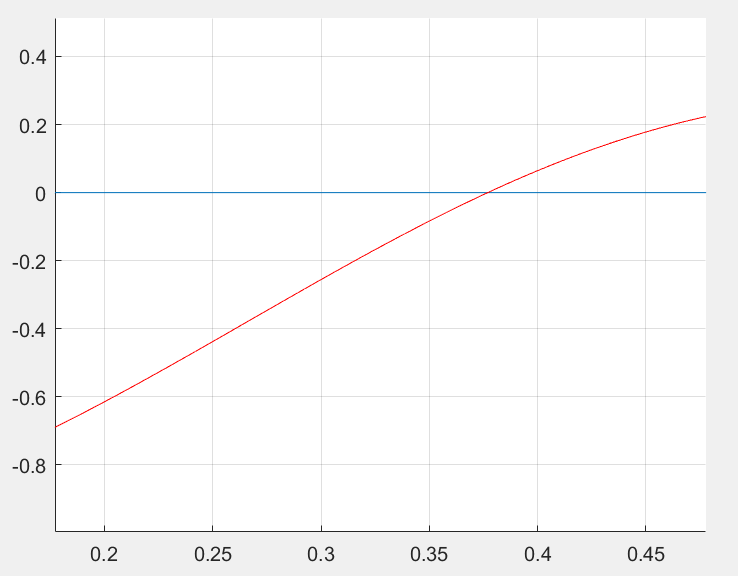
|  |  |  |
| --- | --- | --- |
|  | Алгебраическое уравнение | |
| Метод секущих | Метод половинного деления |
| 0,001 | 1 | 10 |
| 0,0001 | 1 | 13 |
| 0,000001 | 2 | 20 |

Трансцендентное уравнение

Требуется решить трансцендентное уравнение: . Отделим корни. Заметим, что функция чётная, поэтому достаточно рассмотреть корни на положительном направлении оси OX. Построим сразу график в окрестности нуля, содержащей все корни:

Отделим промежутки с положительными корнями:

[0.2; 0.45], [0.65, 0.85], [0.95, 1.1]:



Найдём корни на промежутках с точностью . Для промежутка [0.2; 0.45]:

- Метод секущих: x\* = 0.377112

|  |  |  |
| --- | --- | --- |
|  | Трансцендентное уравнение | |
| Метод секущих | Метод половинного деления |
| 0,001 | 4 | 8 |
| 0,0001 | 5 | 12 |
| 0,000001 | 6 | 18 |

Для промежутка [0.65; 0.85]: x\* = 0.789835

|  |  |  |
| --- | --- | --- |
|  | Трансцендентное уравнение | |
| Метод секущих | Метод половинного деления |
| 0,001 | 5 | 8 |
| 0,0001 | 6 | 11 |
| 0,000001 | 7 | 18 |

Для промежутка [0.95; 1.1]: x\* = 1.000305

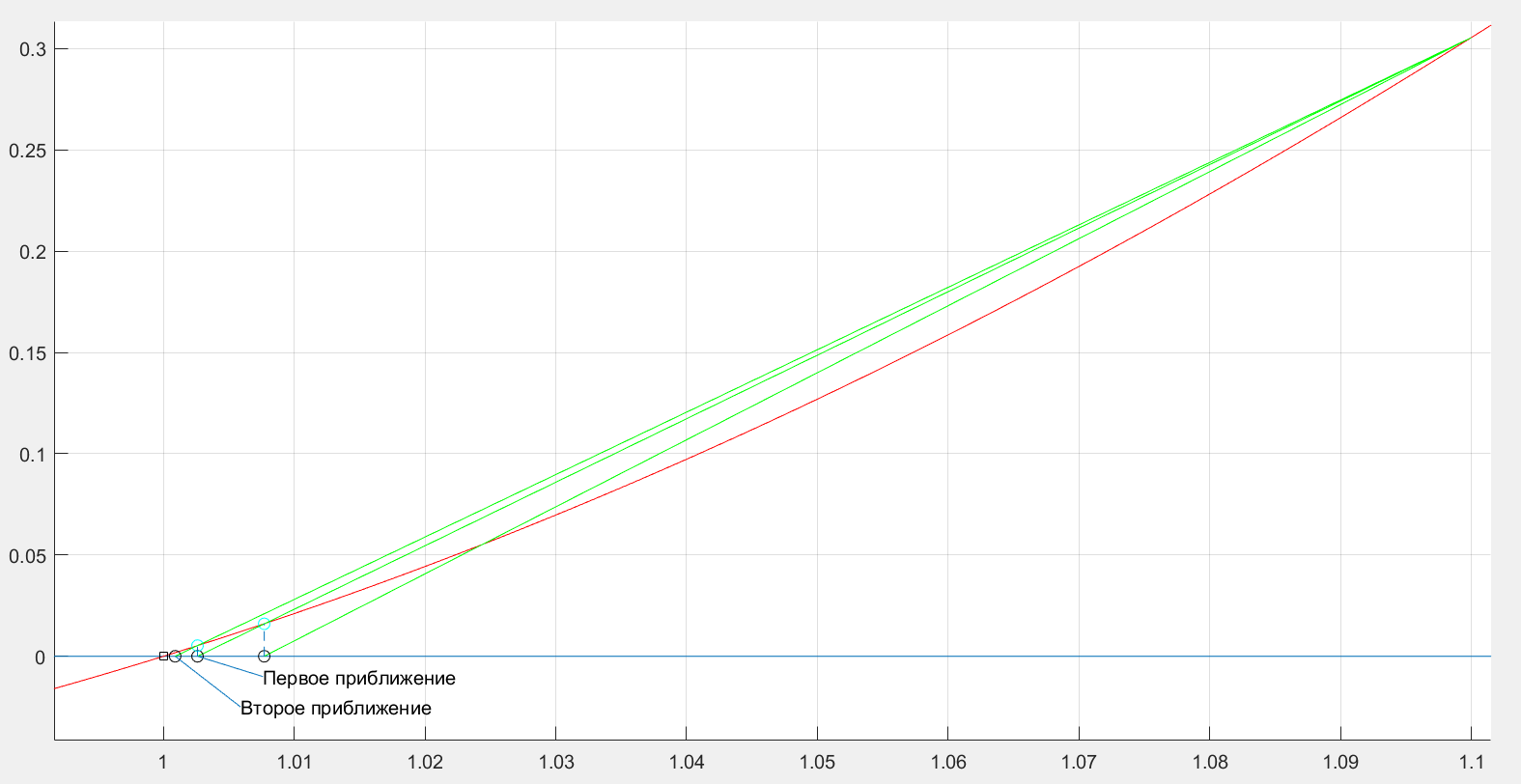
|  |  |  |
| --- | --- | --- |
|  | Трансцендентное уравнение | |
| Метод секущих | Метод половинного деления |
| 0,001 | 2 | 8 |
| 0,0001 | 3 | 11 |
| 0,000001 | 5 | 18 |

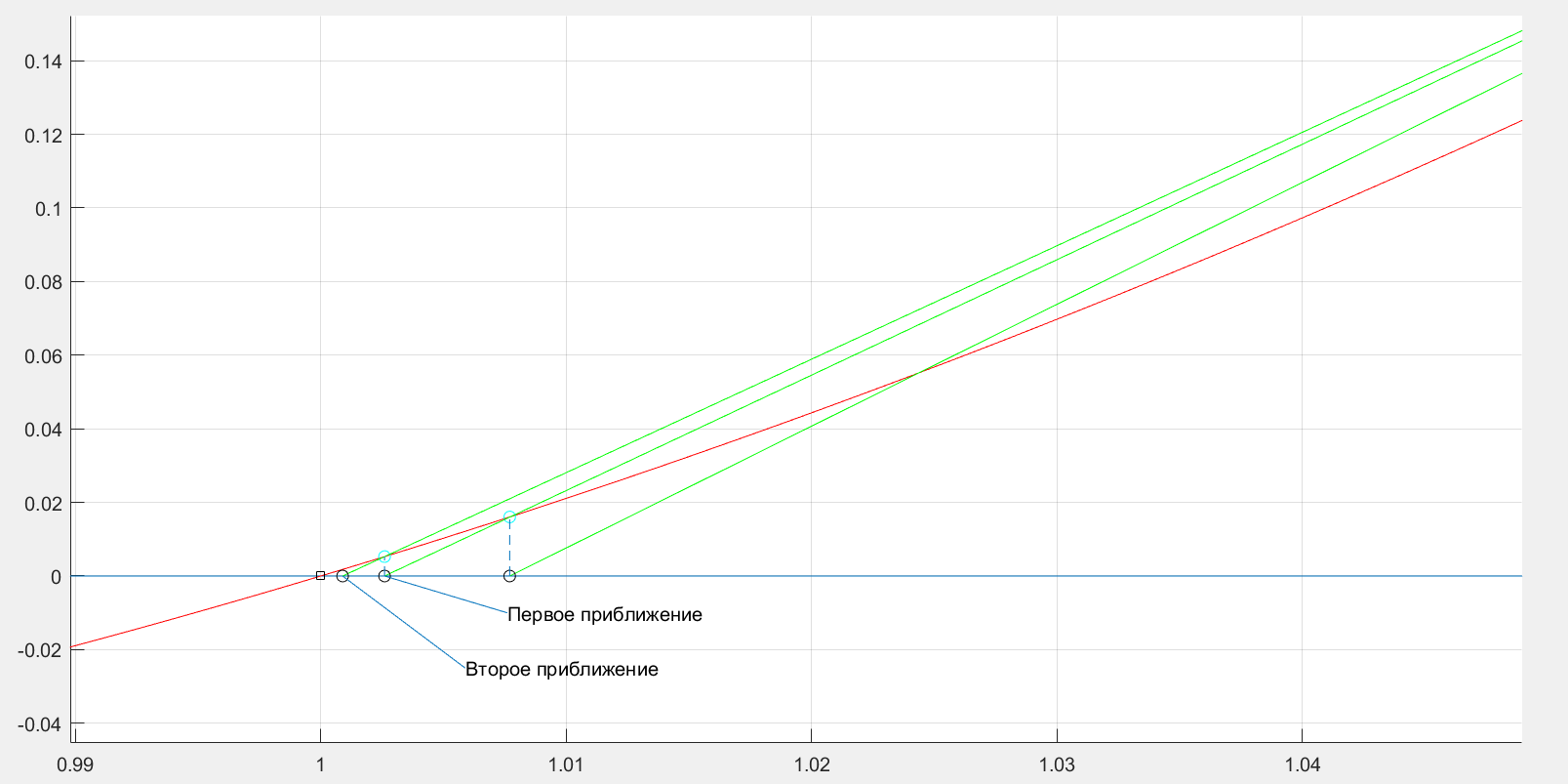
Анализ результатов

По итогам таблиц видно, что метод секущих сходится быстрее, чем метод половинного деления, поэтому считается оптимальным в использовании, где промежутки с корнями уже отделены. При этом заметим, что условия для выбора начальных точек в методе половинного деления гораздо слабее, поэтому этот метод можно признать более уместным в случаях, когда вычисление первой и второй производных затруднено.

Иллюстрация метода секущих

Построим несколько приближений графически для промежутка [0.95; 1.1] трансцендентной функции.





Часть 2. Решение СЛАУ прямыми методами

Постановка задачи

Требуется решить СЛАУ методом вращения, разложив матрицу А системы на ортогональную Q и верхнюю треугольную R. .

Требуется исследовать влияние ошибки в исходных данных на вычисления путём вычисления коэффициентов и нормы вектора невязки d = Ax – b. Известно, что:

Тогда такие, что обращают неравенства в равенства:

Алгоритм применения метода

для i = n – 1, n – 2, … 2, 1.

Преобразуем матрицу А в R применениями операторов вращения. Матрица вращения на каждом шаге имеет вид:

По свойствам матриц вращения на каждом шаге будут меняться только i и j строки матрицы системы.

Условия применимости метода

Для исследования метода построим А следующим образом:

.

Такое построение матрицы А выгодно тем, что мы заранее знаем, что она невырожденная, то есть для неё метод применимый, при этом мы знаем .

Тестовый пример с ручным расчётом

Задание матрицы

Возьмём матрицы порядка 3:

Тогда по приведённым выше формулам получаем разложение:

СЛАУ приобретает вид:

Откуда получаем решение В этом случае вектор невязки получился нулевым.

Внесём случайные ошибки в матрицы:

Тогда получаем искомые коэффициенты:

Модульная структура программы

Norms.h

double CountNormVector1(double\* v, int n); //Посчитать первую норму вектора

double CountNormVector(double\* v, int n); //Посчитать Eвклидову норму вектора

double CountNormMatrix(double\*\* A, int n); //Посчитать 1-ю норму матрицы

MatrixVectorIO.h

void PrintMatrix(double\*\* A, int n); //Вывести матрицу

void PrintVector(double\* v, int n); //Вывести столбец

double\*\* ParseMatrix(char const\* const filename, int n); //Считать матрицу из файла

double\* ParseVector(char const\* const filename, int n); //Считать вектор из файла

QRFactorization.h

typedef struct QR\_t

{

double\*\* q;

double\*\* r;

int n; //Размерность матрицы

} QR\_t;

void DestroyQR(QR\_t\* tmp); //Освободить память

QR\_t\* CallocateQR(int n); //Выделить память под матрицы

QR\_t\* QRFactorizeMatrix(double\*\* A, int n); //Разложить матрицу на матрицу вращения и треугольную

MatrixOps.h

void CopyMatrix(double\*\* src, double\*\* dest, int n); //Скопировать матрицу

void TransposeMatrix(double\*\* A, int n); //Протранспонировать матрицу

double\* MatrixMulVector(double\*\* A, double\* vector, int n); //Перемножить матрицу на вектор

double\*\* MatrixMulMatrix(double\*\* A, double\*\* B, int n); //Перемножить 2 квадратные матрицы

double\*\* MatrixMinus(double\*\* A, double\*\* B, int n); //Вычесть из матрицы A матрицу B

double\* VectorMinus(double\* v1, double\* v2, int n); //Вычесть из вектора v1 вектор v2

SolveSLAE.h

double\* SolveRXSLAE(double\*\* R, double\* b, int n); //Решить СЛАУ с R-матрицей

double\* CountResidualVector(double\*\* A, double\* x, double\* b, int n); //Посчитать вектор невязки

MyRandom.h

double Random(double a, double b); //Получить случайное число в интервале [a; b)

double\* SpoilVector(double\* v, int n); //Испортить вектор

double\*\* SpoilMatrix(double\*\* A, int n); //Испортить матрицу

void RandomInit(int key); //Для генерации псевдослучайных чисел

Вычисление коэффициентов для «больших» матриц

Возьмём матрицы порядка 10.

С плохим cond(A)

Cond(A) = 10^7

С хорошим cond(A)

Cond(A) = 10

Анализ результатов

Метод вращения достаточно удобный в реализации, обладает большой устойчивостью – как видно из результатов, при малых возмущениях отклонения незначительные.

Часть 3. Решение СЛАУ итерационными методами. Метод Зейделя

Постановка задачи

Требуется решить СЛАУ итерационным методом Зейделя, представив систему в удобном для итераций виде:

В методе Зейделя при подсчёте каждой следующей компоненты текущего приближения используются уже полученные компоненты этого же приближения:

Где матрицу С получаем следующим образом:

И столбец g получаем следующим образом:

Условия применимости метода

Для сходимости метода достаточно и A – положительно-определённая. Также необходимо . Если матрица не является симметричной, всегда можно свести систему к подходящей:

При этом, метод является итерационным, поэтому точность зависит от cond(A). Если СЛАУ несимметричная, при домножении cond новой матрицы возрастает квадратично.

Условие остановки

Считается, что решение получено с заданной точностью ɛ, если

Тестовый пример

Возьмём матрицу 3х3:

Очевидно,

Нормализуем СЛАУ:

Решим СЛАУ методом Зейделя с точностью :

Тогда получаем решение и для него вектор невязки:

Ответ получен за 23 итерации.

Модульная структура программы

MatrixOps.h

void CopyMatrix(double\*\* src, double\*\* dest, int n); //Скопировать матрицу

void TransposeMatrix(double\*\* A, int n); //Протранспонировать матрицу

double\* MatrixMulVector(double\*\* A, double\* vector, int n); //Перемножить матрицу на вектор

double\*\* MatrixMulMatrix(double\*\* A, double\*\* B, int n); //Перемножить 2 квадратные матрицы

double\*\* MatrixMinus(double\*\* A, double\*\* B, int n); //Вычесть из матрицы A матрицу B

double\* VectorMinus(double\* v1, double\* v2, int n); //Вычесть из вектора v1 вектор v2

Norms.h

double CountNormVector1(double\* v, int n); //Посчитать первую норму вектора

double CountNormVector(double\* v, int n); //Посчитать Eвклидову норму вектора

double CountNormMatrix(double\*\* A, int n); //Посчитать 1-ю норму матрицы

double CountNormMatrixInf(double\*\* A, int n); //Посчитать бесконечную норму матрицы

double CountNormVectorInf(double\* v, int n); //Посчитать бесконечную норму вектора

MatrixVectorIO.h

void PrintMatrix(double\*\* A, int n); //Вывести матрицу

void PrintVector(double\* v, int n); //Вывести столбец

double\*\* ParseMatrix(char const\* const filename, int n); //Считать матрицу из файла

double\* ParseVector(char const\* const filename, int n); //Считать вектор из файла

Zeydel.h

int isSymmetrial(double\*\* A, int n); //Узнать, является ли матрица симметричной

int isDiagPredominant(double\*\* A, int n); //Узнать, является ли матрица диагонально-преобладающей

double\*\* CreateZeydelCMatrix(double\*\* A, double\* b, int n, double \*\* g); //Посчитать матрицу C и g

double\* ZeydelIteration(double\*\* C, double\* xPrev, int n, double\* g); //Осуществить итерацию метода Зейделя

double\* SolveSLAEZeydel(double\*\* C, int n, double\* g, double epsilon, int\* counter); //Решить СЛАУ методом Зейделя

double\* CountResidualVector(double\*\* A, double\* x, double\* b, int n); //Посчитать вектор невязки

double\*\* GetNormalMatrix(double\*\* A, double\* b, double\*\* newB, int n); //Привести систему к решаемой методом Зейделя

Решение СЛАУ с матрицами порядка 10

Матрица с плохим числом обусловленности

|  |  |  |
| --- | --- | --- |
| ɛ | Число итераций | Норма невязки |
| 0,01 | 34 | 3.41571e-09 |
| 0,001 | 1947 | 5.64351e-10 |
| 0,00001 | 8688 | 5.64426e-12 |

Матрица с хорошим числом обусловленности

Cond(A) = 100

|  |  |  |
| --- | --- | --- |
| ɛ | Число итераций | Норма невязки |
| 0,01 | 55 | 5.12696e-05 |
| 0,001 | 80 | 6.73195e-06 |
| 0,00001 | 4125 | 1.34972e-08 |

Для матрицы с определителем, близким к нулю

|  |  |  |
| --- | --- | --- |
| ɛ | Число итераций | Норма невязки |
| 0,01 | 50 | 4.7768e-05 |
| 0,001 | 69 | 4.75828e-06 |
| 0,00001 | 107 | 4.72146e-08 |

|  |  |  |
| --- | --- | --- |
| ɛ | Число итераций | Норма невязки |
| 0,01 | 169 | 7.1732e-15 |
| 0,001 | 553 | 6.66376e-16 |
| 0,00001 | 84415 | 5.55241e-18 |

Анализ результатов

Метод Зейделя является модификацией метода простых итераций, которая существенно увеличивает сходимость. При этом, условия применимости метода позволяют работать с симметричными матрицами, для обычных матриц приведение СЛАУ к подходящему виду увеличивает число обусловленности. В том случае, когда число обусловленное «плохое» - порядок больше 10^7, - метод сходится, но на бесконечности, поэтому для таких матриц его использовать неэффективно. При этом, если определитель матрицы системы близок к нулю, метод показывает хорошие результаты.

Глава 4. Решение алгебраической проблемы собственных значений

Постановка задачи

Для заданной матрицы А требуется найти хотя бы 2 собственных числа и соответствующих им собственных вектора итерационными методами. , где X – собственный вектор, λ – соответствующее ему собственное число. Для решения задачи требуется использовать степенной итерационный метод. Для нахождения второй пары собственного значения и вектора необходимо использовать модификацию со сдвигом.

Алгоритм степенного метода

Для обеспечения устойчивости степенного метода будем использовать нормировку каждого приближения. За начальное приближение будем брать единичный вектор. Для матрицы А степенной метод позволяет найти максимальное собственное число и соответствующий ему собственный вектор, поэтому для нахождения ещё одной пары воспользуемся сдвигом: , где

Тогда для новой матрицы B собственные векторы не поменяются, а собственные значения изменятся на константу:

Таким образом, максимальное собственное число, полученное для матрицы B степенным методом, будет соответствовать минимальному собственному числу исходной матрицы.

Для сдвига будем брать за µ след матрицы, так как он точно больше максимального собственного числа.

Условия применимости метода

Степенной метод позволяет найти собственные значения и соответствующие им векторы в том случае, если исходная матрица А – простой структуры с вещественными коэффициентами.

Тестовый пример

Проверим работу метода для матрицы размерности 3 с известными собственными числами с точностью ε = 0,01.

Найдём пару, соответствующую

10 итераций:

Проверка

Найдём пару, соответствующую

Сдвиг = 2, новая матрица

2 итерации:

Проверка

Контрольные тесты

Проведём ряд тестов над матрицами, для которых зафиксируем

Для хорошо отделённых собственных чисел

Для собственных чисел,

|  |  |  |
| --- | --- | --- |
| ɛ | Число итераций | |
|  |  |
| 0.01 | 29 | 100 |
| 0.001 | 41 | 128 |
| 0.0001 | 52 | 156 |

Для собственных чисел,

|  |  |  |
| --- | --- | --- |
| ɛ | Число итераций | |
|  |  |
| 0.01 | 35 | 118 |
| 0.001 | 97 | 195 |
| 0.0001 | 159 | 228 |

Для плохо отделённых собственных чисел

Для собственных чисел,

|  |  |  |
| --- | --- | --- |
| ɛ | Число итераций | |
|  |  |
| 0.01 | 14 | 73 |
| 0.001 | 93 | 86 |
| 0.0001 | 145 | 88 |

Для собственных чисел,

|  |  |  |
| --- | --- | --- |
| ɛ | Число итераций | |
|  |  |
| 0.01 | 18 | 2 |
| 0.001 | 116 | 2 |
| 0.0001 | 222 | 863 |

Для собственных чисел,

|  |  |  |
| --- | --- | --- |
| ɛ | Число итераций | |
|  |  |
| 0.01 | 2 | 2 |
| 0.001 | 2 | 2 |
| 0.0001 | 2 | 2 |

Для собственных чисел,

|  |  |  |
| --- | --- | --- |
| ɛ | Число итераций | |
|  |  |
| 0.01 | 2 | 2 |
| 0.001 | 2 | 2 |
| 0.0001 | 2 | 2 |

Модульная структура программы

typedef struct OwnPair\_t //Структура для хранения собственного числа и вектора

{

double number;

double\* vector;

} OwnPair\_t;

void DestroyPair(OwnPair\_t\* pair); //Освободить память

void PrintPair(OwnPair\_t\* pair, int n); //Вывод на экран

void CheckPair(OwnPair\_t\* pair, double\*\* A, int n); //Проверить

double\*\* MoveToFindMin(double\*\* A, int n, double\* move); //Сдвиг для поиска минимального соб. числа

OwnPair\_t\* CountMinOwn(double\*\* A, int n, double epsilon, int\* iter); //Найти минимальное соб. число

OwnPair\_t\* CountMaxOwn(double\*\* A, int n, double epsilon, int\* iter); //Сосчитать максимальное соб. значение и его вектор

Выводы

Степенной метод очень прост в реализации и использовании. Данный метод показал хорошие результаты для матриц с плохой отделимостью собственных чисел. Однако, минус степенного метода состоит в том, что находится только одно собственное число и соответствующий ему вектор, поэтому приходится использовать модификации этого метода.