Министерство образования и науки Российской Федерации  
Санкт-Петербургский политехнический университет Петра Великого  
Институт прикладной математики и механики  
Кафедра “Прикладная математика”

Курсовой проект

По дисциплине: “Численные методы”

Тема: “Исследование численных методов алгебры: итерационные методы”

Студент группы 23631/2   
Самутичев Евгений Романович  
Выполнил(подпись)\_\_\_\_\_(дата)\_\_\_\_\_  
  
Научный руководитель:  
Павлова Людмила Владимировна

# Оглавление

Оглавление

[Оглавление 2](#_Toc533507268)

[Цель работы 4](#_Toc533507269)

[Часть 1. Решение алгебраических и трансцендентных уравнений: метод Ньютона 4](#_Toc533507270)

[Формулировка задачи и её формализация 4](#_Toc533507271)

[Алгоритмы методов и условия их применимости 4](#_Toc533507272)

[Предварительный анализ задачи 5](#_Toc533507273)

[Тестовый расчет. 5](#_Toc533507274)

[Модульная структура программы 5](#_Toc533507275)

[Анализ решения задачи, сравнение методов 6](#_Toc533507276)

[Анализ результатов исследования 8](#_Toc533507277)

[Часть 2. Решение СЛАУ прямыми методами: метод LDR-разложения 10](#_Toc533507278)

[Формулировка задачи и её формализация 10](#_Toc533507279)

[Алгоритм метода LDR-разложения и условия его применимости 10](#_Toc533507280)

[Предварительный анализ задачи 11](#_Toc533507281)

[Тестовый расчет 11](#_Toc533507282)

[Модульная структура программы 11](#_Toc533507283)

[Составление и решение систем 12](#_Toc533507284)

[Анализ результатов исследования 12](#_Toc533507285)

[Часть 3. Решение СЛАУ итерационными методами: метод релаксации 13](#_Toc533507286)

[Формулировка задачи и её формализация 13](#_Toc533507287)

[Алгоритм метода релаксации и условия его применимости 13](#_Toc533507288)

[Модульная структура программы 14](#_Toc533507289)

[Исследование метода релаксации 15](#_Toc533507290)

[Анализ результатов исследования 16](#_Toc533507291)

[Часть 4. Решение алгебраической проблемы собственных чисел: метод Якоби 17](#_Toc533507292)

[Формулировка задачи и её формализация 17](#_Toc533507293)

[Алгоритм метода Якоби и условия его применимости 17](#_Toc533507294)

[Тестовый расчет 18](#_Toc533507295)

[Модульная структура программы 19](#_Toc533507296)

[Исследование метода Якоби 20](#_Toc533507297)

[Анализ результатов исследования 20](#_Toc533507298)

[Выводы 21](#_Toc533507299)

[Приложения 22](#_Toc533507300)

[Приложение к части 1 22](#_Toc533507301)

[Графическая интерпретация 22](#_Toc533507302)

[Программный код 22](#_Toc533507303)

[Приложение к части 2 25](#_Toc533507304)

[Программный код 25](#_Toc533507305)

[Генератор матриц 29](#_Toc533507306)

[Приложение к части 3 32](#_Toc533507307)

[Библиотека для работы с матрицами 32](#_Toc533507308)

[Программный код 33](#_Toc533507309)

[Приложение к части 4 36](#_Toc533507310)

[Программный код 36](#_Toc533507311)

# Цель работы

Целью данной работы является раскрытие преимуществ и недостатков итерационных методов численного решения задач Алгебры. Для этого была построена реализация ряда алгоритмов, исследование которых позволяет собрать информацию по данной теме. Следует добавить, что хотя акцент и делается на численных методах Алгебры, но в части 1 проводится некоторое отступление с целью продемонстрировать присутствие этого класса методов в Математическом Анализе на примере решения давно известной проблемы поиска корня уравнения.

# Часть 1. Решение алгебраических и трансцендентных уравнений: метод Ньютона

## Формулировка задачи и её формализация

Найти корни заданного уравнения с заданной точностью. Т.е. , найти . Требуется сравнить метод Ньютона с методом половинного деления для решения алгебраического уравнения - и трансцендентного .

## Алгоритмы методов и условия их применимости

1. Метод половинного деления.

1.1. Условия применимости: непрерывна на промежутке , тогда для заданного уравнения существует корень, если .

1.2. Алгоритм метода:  
   
 if then   
 if then else   
 if then goto 1 else

2. Метод Ньютона.

2.1. Условия применимости: и , сохраняют знак на и .

2.2. Алгоритм метода:  
if then else   
 ,   
 if then goto 2, , else

## Предварительный анализ задачи

Если существует такой что , то уравнение имеет хотя бы один корень.

## Тестовый расчет.

Рассмотрим двумя методами. В качестве промежутка выберем , поскольку , поэтому корень существует.

1. Метод половинного деления, точность 0.1, промежуток

* , ,
* , ,
* , ,
* , 3 итерации

2. Метод Ньютона, точность 0.1.

* Модифицируем промежуток
* сохраняет знак на промежутке, как и .
* , поэтому
* , ,
* , , , , 2 итерации

## Модульная структура программы

1. Модули функций и их производных. **Fun1**, **dFun1**, **d2Fun1** принимают , возвращают , , соответственно для алгебраической функции. **Fun2**, **dFun2**, **d2Fun2** для трансцендентной.

2. Модуль **Plot** для построения графика функции на промежутке . Принимает **a**, **b** и ссылку на функцию **@fun** (например **@Fun1**, **@dFun2** …). Возвращает график соответствующей функции, с построенной сеткой.

3. Модуль **Slash** для нахождения корня функции на промежутке с заданной точностью , методом половинного деления. Принимает **a**, **b**, **eps** и ссылку на функцию **@fun**. Возвращает корень **root** и число пройденных итераций **k**.

4. Модуль **Newton** для нахождения корня функции на промежутке с заданной точностью , методом Ньютона. Принимает **a**, **b**, **eps** и ссылки на функцию и её производные: **@fun**, **@dfun**, **@d2fun**. Возвращает корень root и число пройденных итераций **k**.

5. Модуль **NewtonPlot** для графической интерпретации метода Ньютона. Является модификацией модуля **Newton**, поэтому принимает те же параметры. Возвращает график.

6. Модули **TestSlash** и **TestNewton** для тестирования методов, принимают те же параметры, что и модули соответствующих методов, кроме точности. Возвращают таблицу 3x4, в которой каждый столбец соответствует определенному значению точности: 0.1, 0.01, 0.001, 0.0001. И содержит приближенное значение **root**, количество итераций **k** и **root** – значение функции в найденной точке.

## Анализ решения задачи, сравнение методов

Строим графики функций на достаточно большом промежутке, для обнаружения корней. Для трансцендентной функции также построим график в другом масштабе для того, чтобы уточнить промежутки. Были запущены модули: Plot(-5,5,@Fun1),   
Plot(-3,3,@Fun2) и Plot(0.5,1.5,@Fun2).

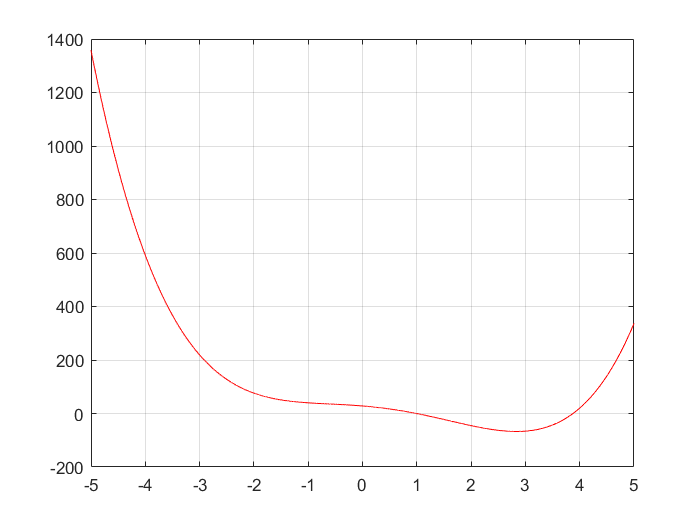


Рисунок 1. График алгебраической функции.

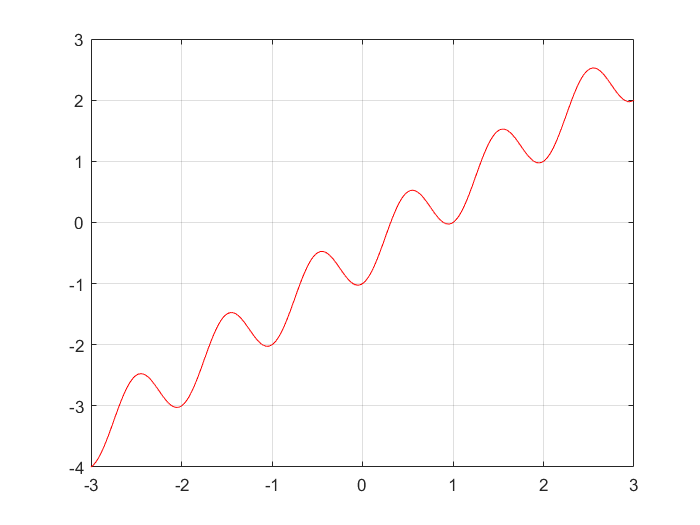


Рисунок 2. График трансцендентной функции.

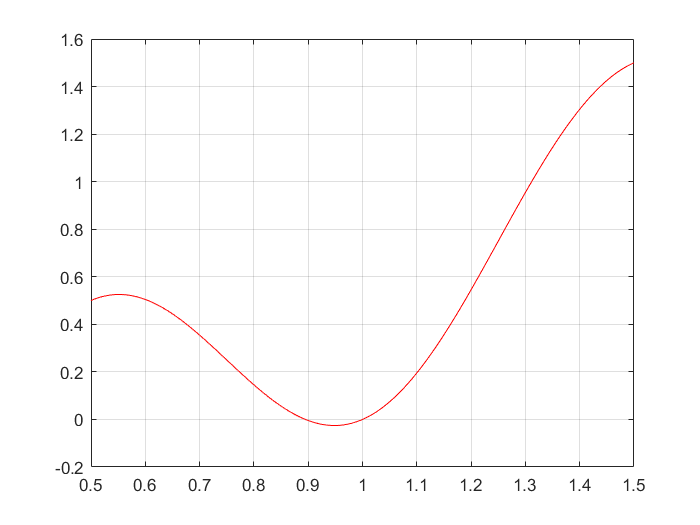


Рисунок 3. График трансцендентной функции в другом масштабе.

Можно сделать вывод, что заданное алгебраическое уравнение имеет корни на промежутках и , а трансцендентное на , и . Теперь, для каждого корня каждого уравнения построим таблицу, в которую при заданной точности войдут найденный корень , число итераций , и значение функции в соответствующей точке при каждом из двух методов (см. Таблицы 1-10 в приложении). Также, построим графическую интерпретацию метода Ньютона для алгебраического уравнения на , с точностью 0.1 (см. Рисунок 4 в приложении).

Таблица 1. Алгебраическое уравнение, [0, 2].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | 0.9688 | 0.9961 | 0.9927 | 0.9924 |
|  | 0.9688 | 0.9926 | 0.9926 | 0.9924 |
|  | 6 | 9 | 12 | 16 |
|  | 3 | 3 | 3 | 4 |
|  | 0.9406 | -0.1486 | -0.0116 | -0.0006 |
|  | -0.0089 | -0.0089 | -0.0089 | -3.0032e-006 |

Таблица 2. Алгебраическое уравнение, [3, 5].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | 3.9063 | 3.8789 | 3.8765 | 3.8763 |
|  | 3.8814 | 3.8762 | 3.8762 | 3.8762 |
|  | 6 | 9 | 12 | 16 |
|  | 3 | 4 | 4 | 5 |
|  | 4.6068 | 0.4035 | 0.0358 | 0.0037 |
|  | 0.7870 | 0.0028 | 0.0028 | 37.1727e-009 |

Таблица 3. Транцендентное уравнение, [0.2, 0.4].

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | 0.2813 | 0.3086 | 0.3110 | 0.3115 |
|  | 0.2930 | 0.3113 | 0.3113 | 0.3115 |
|  | 4 | 7 | 10 | 14 |
|  | 1 | 2 | 2 | 3 |
|  | -0.1212 | -0.0115 | -0.0019 | -0.0010 |
|  | -0.0734 | -0.0010 | -0.0010 | -240.6736e-009 |

Таблица 4. Трансцендентное уравнение, [0.8, 0.95], метод половинного деления.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | 0.9125 | 0.8984 | 0.8946 | 0.8949 |
|  | 0.8949 | 0.8949 | 0.8949 | 0.8949 |
|  | 2 | 5 | 9 | 12 |
|  | 3 | 3 | 4 | 4 |
|  | -0.0138 | -0.0032 | 0.0003 | 28.7945e-006 |
|  | 0.0001 | 0.0001 | 27.4725e-009 | 27.4725e-009 |

Таблица 5. Трансцендентное уравнение, [0.96, 1.05], метод половинного деления.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | 1.0050 | 1.0022 | 0.9997 | 1.0000 |
|  | 1.0122 | 1.0012 | 1.0000 | 1.0000 |
|  | 1 | 5 | 8 | 11 |
|  | 1 | 2 | 3 | 3 |
|  | 0.0052 | 0.0022 | -0.0003 | 34.1912e-006 |
|  | 0.0137 | 0.0012 | 13.5342e-006 | 13.5342e-006 |

## Анализ результатов исследования

Результаты тестирования показали, что среднее количество итераций по методу Ньютона , меньше чем по методу половинного деления. При этом, значения функций в соотв. точках (точность 0.001) в среднем ближе к нулю для метода Ньютона, чем для метода половинного деления. Этим подтверждается что в сравнении итерационных методов: метод Ньютона лучше, чем метод половинного деления. Также заметим, что он сходится даже с меньшей ошибкой, чем требуется. Таким образом проявляется квадратичная сходимость метода.

# Часть 2. Решение СЛАУ прямыми методами: метод LDR-разложения

## Формулировка задачи и её формализация

Пусть . Найти такой , что . Требуется сравнить вычислительную ошибку для матриц при различных порядках числа обусловленности . Также, оценить погрешность вычислений при малых возмущениях и проверить согласованность полученного результата с теорией.

## Алгоритм метода LDR-разложения и условия его применимости

Метод LDR-разложения основан на представлении матрицы системы в виде произведения трех матриц – нижней треугольной (), диагональной () и верхней треугольной ().

. После чего, решение данной системы сводится к решению трех элементарных .

Алгоритм разложения:

 for to do   
   
  for to do  
   
 end-do;  
   
 if then   
 for to do   
   
 for to do  
   
 end-do;  
   
   
 for to do   
   
 end-do;  
   
 end-do;   
end-do;

Метод применим, если все угловые миноры матрицы системы отличны от нуля. На практике это реализуется проверкой неравенства нулю на каждой итерации.

## Предварительный анализ задачи

Если , то система имеет единственное решение. Алгоритм, приведенный выше завершает работу если . Поскольку при разложении , мы можем утверждать, что если решение получено, то оно единственное.

## Тестовый расчет

**1.**   
Вычислим , далее , .  
Вычислим .  
Имеем , , .

**2.** Далее решим систему . , .  
Решим систему . , .  
Наконец, решим систему . , .  
**3.** Действительно, - решение системы, таким образом работа алгоритма проверена на примере.

## Модульная структура программы

**float\*\* CreateMatrix (int N)** – создает матрицу порядка N.

**void FreeMatrix (float\*\* matrix, int N)** – удаляет матрицу matrix порядка N.

**void DSolve (float\*\* D, float\* b, float\* x, int N)** – решает систему, матрица D которой диагональная. b это столбец правой части, x это столбец корней.

**void RSolve (float\*\* R, float\* b, float\* x, int N)** – решает систему, матрица R которой верхне-треугольная.

**void LSolve (float\*\* L, float\* b, float\* x, int N)** – решает систему, матрица L которой нижне-треугольная.

**void LDR (float\*\* A, float\*\* L, float\*\* D, float\*\* R, int N)** – выполняет LDR разложение матрицы A.

**void EpsB (float\*\* A, float\* b, float\* x, int N)** – печатает относительную погрешность, столбцы x, A\*x и b.

**float VecNorma (float\* vec, int N)** – считает бесконечную норму вектора vec.

**float MatNorma (float\*\* mat, int N)** – считает бесконечную норму матрицы mat.

**int Sign ()** – случайным образом возвращает -1 или 1.

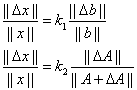
**void Solve (float\*\* A, float\* b, float\* x, int N)** – решает систему методом LDR-разложения.

**void K1 (float\*\* A, float\* b, float\* x, int N)** – вычисляет k1.

**void K2 (float\*\* A, float\* b, float\* x, int N)** – вычисляет k2.

## Составление и решение систем

Для составления системы, с заданным числом обусловленности , выберем ненулевые элементыдиагональной матрицы таким образом, чтобы . Далее, сгенерируем случайным образом столбец , так чтобы . Тогда матрица , где будет иметь число обусловленности, соответствующее . Программный код см. в [Приложении](#_Генератор_матриц). Построим 3 матрицы порядка 10 с обусловленностью . После генерации, матрица записывается в файл In.txt, вместе со сгенерированным вектор-столбцом правой части. Откуда затем считывается программой, после чего программа решает систему, вычисляет коэффициенты , определяемые формулами:



Для каждого числа обусловленности был проведен эксперимент с матрицей порядка 10. Средние значения и промежуточные результаты занесены в таблицу:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Анализ результатов исследования

Таким образом установлено, что с ростом числа обусловленности относительная погрешность вычислений при использовании метода LDR-разложения – возрастает. поскольку возмущение матрицы, в отличие от возмущения вектора, производит дополнительные погрешности в процессе LDR-разложения.

## 

# Часть 3. Решение СЛАУ итерационными методами: метод релаксации

## Формулировка задачи и её формализация

Пусть , . Требуется решить СЛАУ вида . Все итерационные методы основываются на приведении системы к каноническому виду. Итерационный процесс задает последовательность , исходя из некоторого начального приближения в виде первых членов последовательности. Требуется найти такой член последовательности , что . Обычно полагают, что в таком случае получено решение СЛАУ, с точностью .

## Алгоритм метода релаксации и условия его применимости

Каноническая форма уравнения итерационного процесса метода релаксации – .

Пусть - начальное приближение к корню. Тогда воспользуемся следующим алгоритмом:

;   
;   
   
   
 ;  
 ;   
   
 ;  
   
   
 ;  
   
 ;   
   
 ;   
 )  
 ;   
 ;

По *теореме Островского-Рейча*: если матрица удовлетворяет условиям и при , метод релаксации сходится при любом . Поэтому, в качестве начального приближения был использован нулевой вектор. Заметим, что при , метод релаксации обращается в метод Зейделя.

## Модульная структура программы

*Модули для работы с матрицами:*

Тип **mat** эквивалентен **float\*\***, тип **vec** эквивалентен **float\***

**mat matrixInit (int size)** – инициализирует матрицу порядка **size**

**void matrixDone (mat matrix, int size)** – удаляет матрицу **matrix**

**vec vectorInit (int size)** – инициализирует вектор порядка **size**

**void vectorDone (vec vector)** – удаляет вектор **vector**

**void matXvec (mat matrix, vec vector, int size, vec result)** – умножает матрицу **matrix** на вектор **vector**, записывает результат в **result**.

**float vecInfNorm (vec vector, int size)** – вычисляет бесконечную норму вектора **vector**.

**float matInfNorm (mat matrix, int size)** – вычисляет бесконечную норму матрицы **matrix**.

**void vecDvec (vec vec1, vec vec2, vec result, int size)** – вычисляет разность векторов **vec1** и **vec2**, записывает результат в вектор **result**.

**void matDevideDetOnS (mat matrix, int size, float S)** – уменьшает определитель матрицы **matrix** в **S** раз, сохраняя положительную определенность и симметричность.

*Модуль метода релаксации и модули его тестирования:*

**vec RelaxationMethod (float omega, float epsilon, mat A, vec b, int size, int\* k)** – решает СЛАУ методом релаксации, возвращая вектор решений. Зависит от коэффициента **,** заданной точности **.** Возвращает число итераций по ссылке на **k**.

**void Test1 (void)** – запускает решение системы методом релаксации, считывая информацию о матрице **А**, столбце **b**, коэффициенте и точности из файла *Input.txt*. Выводит на экран корни и число итераций.

**void Test2 (void)** – запускает решение системы, записанной в файле *Input.txt*, при различных значениях коэффициента**.**

**void Test3 (void)** – запускает решение системы, записанной в файле *Input.txt*, при различных порядках определителя матрицы **А.**

## Исследование метода релаксации

Для исследования метода релаксации, решим СЛАУ с тремя матрицами (обусловленность , , ) **А** порядка 12, 19 раз, при различных значениях коэффициента от 0.1 до 1.9. После чего вынесем число итераций в *таблицу 1*.

Сгенерируем 5 матриц (обусловленность ) таким образом, чтобы сразу после генерации собственных чисел, они делились некоторое количество раз на . После чего вычислялся определитель. Результаты вычислений при (для метода верхней релаксации) приведены в *таблице 2*. Дополнительным экспериментом, будем изменять уже сгенерированную ранее матрицу **А** (обусловленность ) таким образом, чтобы её определитель уменьшался в раз, посредством умножения всех строк на . Запишем число итераций в *таблицу 3*. 

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 0.1 | 2350 | 10718 | 35679 |
| 0.2 | 1551 | 6502 | 25112 |
| 0.3 | 1160 | 5527 | 19249 |
| 0.4 | 939 | 4739 | 15483 |
| 0.5 | 777 | 4059 | 12729 |
| 0.6 | 649 | 3489 | 10705 |
| 0.7 | 539 | 3021 | 9047 |
| 0.8 | 439 | 2649 | 7603 |
| 0.9 | 472 | 2366 | 6283 |
| 1.0 | 496 | 2169 | 4986 |
| 1.1 | 590 | 2396 | 3668 |
| 1.2 | 628 | 2807 | 4650 |
| 1.3 | 778 | 3476 | 5798 |
| 1.4 | 942 | 3852 | 5856 |
| 1.5 | 1258 | 5046 | 8591 |
| 1.6 | 1611 | 7512 | 10130 |
| 1.7 | 2140 | 10161 | 13294 |
| 1.8 | 3355 | 16543 | 20999 |
| 1.9 | 7009 | 34536 | 44427 |

**Таблица 1.** Зависимость сходимости от коэффициента релаксации для трех различных матриц.

|  |  |  |  |
| --- | --- | --- | --- |
|  | , k | , k | , k |
| 0.00127 | 799 | 367 | 223 |
| 0.000002 | 1812 | 1000 | 2923 |

**Таблица 2.** Зависимость сходимости от величины определителя, случай 1.

|  |  |
| --- | --- |
|  |  |
| 1 | 496 |
|  | 505 |
|  | 513 |
|  | 595 |
|  | 612 |
|  | 626 |
|  | 638 |
|  | 648 |
|  | 658 |
|  | 670 |
|  | 693 |

**Таблица 3.** Зависимость сходимости от величины определителя, случай 2.

## Анализ результатов исследования

В двух экспериментах из трех при сходимость лучше. Таким образом продемонстрировано, что можно добиться лучших результатов, чем при применении метода Зейделя, при оптимальном коэффициента , подбор которого происходит экспериментально.

При уменьшении определителя, заметно ухудшение сходимости при любом коэффициенте “омега” (см. Таблицу 2). Число итераций меняется на целый порядок. Это связано с тем, что при стремлении определителя к нулю, система перестает быть детерминированной и сложнее выделить точное решение из множества близких к нему. Если две матрицы отличаются постоянным множителем, то данное явление незаметно (см. Таблицу 3), поскольку вычисления также будут отличаться лишь постоянным множителем и ухудшение сходимости будет связано только с точностью задания элементов.

# Часть 4. Решение алгебраической проблемы собственных чисел: метод Якоби

## Формулировка задачи и её формализация

Для симметричной матрицы требуется найти все числа , каждое из которых характеризуется существованием вектора . Мы рассматриваем симметричные матрицы () поскольку только в этом случае метод Якоби сходится. Более того, в таком случае гарантируется вещественность как собственных чисел , так и соответствующих им собственных векторов .

## Алгоритм метода Якоби и условия его применимости

Ключевая идея алгоритма метода заключается в выборе матрицы вращения на каждой итерации таким образом, чтобы её можно было использовать для приближения к нулю элементов вне главной диагонали матрицы. Формулы для косинуса и синуса преобразованы с целью избавиться от использования тригонометрических функций в вычислениях, поскольку это нецелесообразно с вычислительной точки зрения. Изначальные формулы закомментированы в тексте алгоритма.

;  
   
   
   
 ;   
   
   
   
   
   
 ; //Замена ;  
 ; //---||---

; //Замена ;  
 ; //---||---

;

;   
 ;  
   
   
   
 ;   
   
   
   
   
;   
;

Метод применим для симметричных матриц поскольку в этом случае все СЧ вещественные. В случае матрицы порядка 2, метод сходится за одну итерацию, что продемонстрировано далее.

## Тестовый расчет

## Модульная структура программы

*Модули для работы с матрицами:*

Тип **mat** эквивалентен **float\*\***, тип **vec** эквивалентен **float\***

**mat matrixInit (int size)** – инициализирует матрицу порядка **size**

**void matrixDone (mat matrix, int size)** – удаляет матрицу **matrix**

**vec vectorInit (int size)** – инициализирует вектор порядка **size**

**void vectorDone (vec vector)** – удаляет вектор **vector**

**void matXvec (mat matrix, vec vector, int size, vec result)** – умножает матрицу **matrix** на вектор **vector**, записывает результат в **result**.

**void matXmat (mat mat1, mat mat2, mat result, int size)** – перемножает матрицы **mat1** и **mat2**, результат записывает в **result**.

**void trponMat (mat matrix, int size)** – транспонирует матрицу **matrix.**

**float vecInfNorm (vec vector, int size)** – вычисляет бесконечную норму вектора **vector**.

**void vecDvec (vec vec1, vec vec2, vec result, int size)** – вычисляет разность векторов **vec1** и **vec2**, записывает результат в вектор **result**.

*Модули метода Якоби:*

**mat createRotMatrix (int I, int J, float theta, int size)** – создает матрицу поворота с заданным углом **theta**.

**vec methodYakobi (mat matrix, int size, float epsilon, int\* k, mat eigVec)** – решает АПСЧ для матицы **matrix**, преобразуя матрицу в диагональную некоторой точностью **epsilon**, возвращает столбец собственных чисел и число итераций **k**.

## Исследование метода Якоби

*Поставим следующие эксперименты по нахождению СЧ (для матрицы порядка 10):*  
**1.** Для хорошо обусловленной матрицы ()  
**2.** Для плохо обусловленной матрицы ()  
**3.** Для хорошо отделимых собственных чисел (1, 2, 3, …, 10)  
**4.** Для плохо отделимых собственных чисел (1.00000, 1.00001, 1.00002, …., 1.00008, 10.0)

Матрицы генерировались в программе, алгоритм работы которой описан в части 2. Выбор собственных чисел для генерации матрицы в 3 и 4 экспериментах был выполнен таким образом, чтобы обусловленность была одна и та же.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 117 | 132 |
|  | 0.000011 | 9.107666 |

Таблица 1.

Таблица 1 отражает результаты экспериментов 1, 2. Небольшие различия в числе итераций послужили поводом провести два дополнительных теста с целью удостовериться в том, что оно по большей части зависит от размерности матрицы, но не от обусловленности. Результаты занесены в таблицу 2.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 108 | 266 |

Таблица 2.

Наконец, в таблицу 3 занесем результаты экспериментов 3 и 4.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 0.999999, 1.999999, 8.000000, 3.999999, 5.000000, 6.000001, 6.999999, 3.000000, 8.999998, 9.999998 | 1.000000, 1.000063, 1.000021, 1.000025, 1.000035, 1.000050, 1.000026, 1.000066, 1.000077, 10.000000 |

Было замечено, что если уменьшить , то результаты во втором тесте намного лучше сопоставляются исходным данным: 1.000000, 1.000061, 1.000020, 1.000010, 1.000040, 1.000050, 1.000031, 1.000070, 1.000080, 10.000000.

## Анализ результатов исследования

**1.** Число итераций зависит от размерности матрицы и не зависит от обусловленности. Это объясняется тем, что условие выхода метода зависит не от качества результата как такового (точности вычисления собственных чисел), а от близости матрицы к диагональной. Чем матрица больше, тем больше нужно выполнить операций.  
**2.** С ростом числа обусловленности увеличивается погрешность, вычисляемая как

. Вот эта величина уже зависит от точности результата и очевидно она будет выше в случаях, когда погрешность вычислений больше из-за плохой обусловленности.

**3.** Если собственные числа отличаются в разряде, вес которого меньше , то отделение друг от друга собственных чисел затруднительно. Можно даже интерпретировать результаты так, согласно которым собственное число имеет кратность (в тесте выше 1.000026 и 1.000025), что не соответствует действительности. Таким образом, важно применять метод с различными значениями , чтобы быть уверенными в корректности результатов.

# Выводы

В ходе курсовой работы был исследован ряд методов численного решения алгебраических задач, большинство из которых являются итерационными. В частях 2 и 3, раскрыты методы решения одной и той же проблемы поиска корней СЛАУ, но с разных сторон. Итерационный метод – метод релаксации позволяет путем более продолжительной работы, снизить влияние погрешности вычислений. В этом заключается его главное преимущество перед прямым методом LDR-разложения.

Части 1 и 4 подчеркивают широту применения итерационных методов и раскрывают иные их особенности. Так, в методе Якоби можно заметить необходимость в уточнении значений, поскольку первые полученные результаты позволяют сделать вывод о наличии кратности собственных чисел, который после оказывается неверным. Часть 1 предлагает сравнение двух итерационных методом друг с другом, в котором выигрывает метод Ньютона, опирающийся на более четкие предпосылки наличия корня в некоторой точке на следующей итерации, нежели метод половинного деления. Эта идея также прослеживается в части 3, когда грамотный выбор коэффициента “омега” позволяет значительно повысить производительность, в сравнение с методом Зейделя, при котором он всегда равен 1.

Итерационный метод тем лучше, чем выше уверенность в том направлении вычислений, в котором он идет. Поэтому довольно часто они ограничены в применении. Это справедливо как для метода Ньютона из части 1, так и для остальных методов. Например, если определитель системы мал, матрица не является симметричной и прочее, итерационные методы алгебры могут быть крайне неэффективны из-за неопределенности результата. В этом, в свою очередь, их главный недостаток.

# Приложения

## Приложение к части 1

### Графическая интерпретация

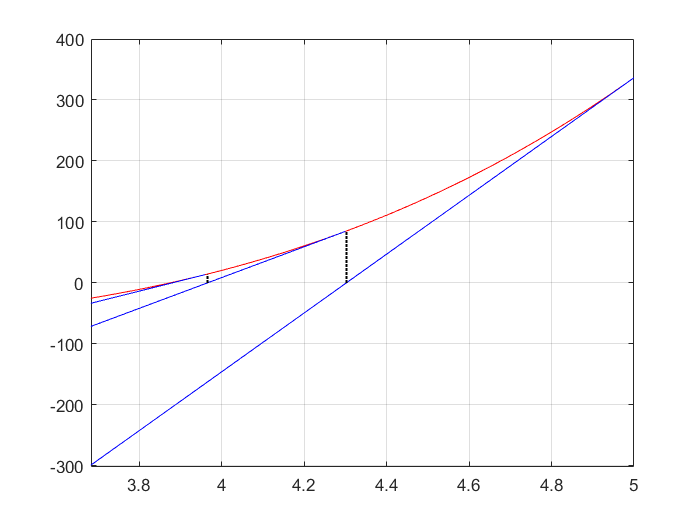


Рисунок 4. Несколько итераций метода Ньютона - графически.

### Программный код

**d2Fun1.m**

function [y] = d2Fun1(x)

a = 1.7125;

b = -3.4192;

c = -10.0420;

d = -16.6985;

e = 28.1418;

y = 12\*a\*x.^2 + 6\*b\*x + 2\*c;

end

**d2Fun2.m**

function [y] = d2Fun2(x)

y = 2\*(pi^2)\*cos(2\*pi\*x);

end

**dFun1.m**

function [y] = dFun1 (x)

a = 1.7125;

b = -3.4192;

c = -10.0420;

d = -16.6985;

e = 28.1418;

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

end

**dFun2.m**

function [y] = dFun2(x)

y = 1 + pi\*sin(2\*pi\*x);

end

**Fun1.m**

function [y] = Fun1(x)

a = 1.7125;

b = -3.4192;

c = -10.0420;

d = -16.6985;

e = 28.1418;

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

end

**Fun2.m**

function [y] = Fun2 (x)

y = x - (cos(pi\*x)).^2;

end

**Newton.m**

function [root, k] = Newton(a,b,eps,fun,dfun,d2fun)

L = a:0.001:b;

m1 = min(abs(dfun(L)));

M2 = max(abs(d2fun(L)));

if fun(a)\*d2fun(a) > 0

x = a;

else

x = b;

end

t = 0;

X = x;

while (M2\*((X-x)^2)/(2\*m1) > eps || t == 0)

x = X;

X = x - (fun(x)/dfun(x));

t = t+1;

end

root = X;

k = t;

end

**NewtonPlot.m**

function [] = NewtonPlot(a,b,eps,fun,dfun,d2fun)

L = a:0.001:b;

m1 = min(abs(dfun(L)));

M2 = max(abs(d2fun(L)));

Plot(a,b,fun);

hold on;

if fun(a)\*d2fun(a) > 0

x = a;

else

x = b;

end

X = x;

t = 0;

while (M2\*((X-x)^2)/(2\*m1) > eps || t == 0)

x = X;

T = fun(x) + dfun(x) \* (L - x);

plot(L,T,'b');

X = x - (fun(x)/dfun(x));

L = a:0.01:X;

K = X\*ones(1, length(0:0.1:fun(X)));

plot(K,0:0.1:fun(X),'k:');

hold on;

t = 1;

end

end

**Plot.m**

function [] = Plot(a,b,fun)

x = a:0.001:b;

plot (x,fun(x),'r');

grid on;

end

**Slash.m**

function [root, k] = Slash(a,b,eps,fun)

t = 1;

while (b-a) > eps

if fun(a)\*fun((a+b)/2) < 0

b = (a+b)/2;

else

a = (a+b)/2;

end

t = t + 1;

end

root = (a+b)/2;

k = t;

end

**TestNewton.m**

function [] = TestNewton(a, b, fun, dfun, d2fun)

eps = [0.1 0.01 0.001 0.0001];

Test = [];

k = 0;

for i=1:1:4

[x k] = Newton(a, b, eps(i), fun, dfun, d2fun);

Test(1, i) = x;

Test(2, i) = k;

Test(3, i) = fun(x);

end

disp(Test);

end

**TestSlash.m**

function [] = TestSlash(a, b, fun)

eps = [0.1 0.01 0.001 0.0001];

Test = [];

k = 0;

for i=1:1:4

[x k] = Slash(a, b, eps(i), fun);

Test(1, i) = x;

Test(2, i) = k;

Test(3, i) = fun(x);

end

disp(Test);

end

## Приложение к части 2

### Программный код

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>

#define PTH\_MAX 80

#define EPS 0.000001f

int flag = 0;

float\*\* **CreateMatrix** (int N)

{

int i;

float\*\* q;

q = (float\*\*)malloc(N\*sizeof(float\*));

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

q[i] = (float\*)malloc(N\*sizeof(float));

return q;

}

void **FreeMatrix** (float\*\* matrix, int N)

{

int i;

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

free(matrix[i]);

free(matrix);

}

void **DSolve** (float\*\* D, float\* b, float\* x, int N)

{

int i;

for (i = 1; i <= N; i++)

x[i-1] = b[i-1]/D[i-1][i-1];

}

void **RSolve** (float\*\* R, float\* b, float\* x, int N)

{

int k, j;

float temp = 0;

for (k = N; k >= 1; k--)

{

temp = b[k-1];

for (j = k+1; j <= N; j++)

temp = temp - (R[k-1][j-1]\*x[j-1]);

x[k-1] = temp/R[k-1][k-1];

}

}

void **LSolve** (float\*\* L, float\* b, float\* x, int N)

{

int k, j;

float temp = 0;

for (k = 1; k <= N; k++)

{

temp = b[k-1];

for (j = 1; j <= k; j++)

temp = temp - (L[k-1][j-1]\*x[j-1]);

x[k-1] = temp/L[k-1][k-1];

}

}

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

{

int i, j, k, m;

float temp;

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

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

{

D[i][j] = 0;

if (i == j)

{

L[i][j] = 1;

R[i][j] = 1;

}

else

{

L[i][j] = 0;

R[i][j] = 0;

}

}

for (m = 1; m <= N; m++)

{

temp = 0;

for (k = 1; k <= m-1; k++)

temp = temp + L[m-1][k-1]\*D[k-1][k-1]\*R[k-1][m-1];

D[m-1][m-1] = A[m-1][m-1] - temp;

if (fabsf(D[m-1][m-1]) < EPS)

{

flag = 1;

return;

}

for (j = m+1; j <= N; j++)

{

temp = 0;

for (k = 1; k <= m-1; k++)

temp = temp + L[m-1][k-1]\*D[k-1][k-1]\*R[k-1][j-1];

R[m-1][j-1] = (A[m-1][j-1] - temp)/D[m-1][m-1];

temp = 0;

for (k = 1; k <= m-1; k++)

temp = temp + L[j-1][k-1]\*D[k-1][k-1]\*R[k-1][m-1];

L[j-1][m-1] = (A[j-1][m-1] - temp)/D[m-1][m-1];

}

}

}

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

{

int i, j;

float temp;

float max1 = 0, max2 = 0;

printf("x: ");

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

printf("%f ", x[i]);

printf("\n");

printf("A\*x: ");

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

{

temp = 0;

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

temp = temp + A[i][j]\*x[j];

printf("%f ", temp);

if (max1 < fabsf(b[i] - temp))

max1 = fabsf(b[i] - temp);

if (max2 < fabsf(b[i]))

max2 = fabsf(b[i]);

}

printf("\nb: ");

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

printf("%f ", b[i]);

printf("\neps = %f\n", max1);

}

float **VecNorma** (float\* vec, int N)

{

int i;

float max = 0;

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

if (max < fabsf(vec[i]))

max = fabsf(vec[i]);

return max;

}

float **MatNorma** (float\*\* mat, int N)

{

int i, j;

float max = 0;

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

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

if (max < fabsf(mat[i][j]))

max = fabsf(mat[i][j]);

return max;

}

int **Sign** ()

{

int p = rand() % 2;

if (p == 1)

return -1;

else

return 1;

}

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

{

float\* y = (float\*)malloc(N\*sizeof(float));

float\* z = (float\*)malloc(N\*sizeof(float));

float\*\* L = CreateMatrix(N);

float\*\* D = CreateMatrix(N);

float\*\* R = CreateMatrix(N);

LDR(A, L, D, R, N);

LSolve(L, b, z, N);

DSolve(D, z, y, N);

RSolve(R, y, x, N);

free(y);

free(z);

FreeMatrix(L, N);

FreeMatrix(D, N);

FreeMatrix(R, N);

}

void **K1** (float\*\* A, float\* b, float\* x, int N)

{

int i;

float norma = VecNorma(b, N);

srand(time(0));

float\* db = (float\*)malloc(N\*sizeof(float));

float\* B = (float\*)malloc(N\*sizeof(float));

float\* X = (float\*)malloc(N\*sizeof(float));

float\* dx = (float\*)malloc(N\*sizeof(float));

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

db[i] = Sign() \* (1.0f\*rand())/RAND\_MAX;

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

{

db[i] = db[i] \* 0.01f \* b[i];

B[i] = b[i] + db[i];

}

Solve(A, B, X, N);

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

dx[i] = X[i] - x[i];

printf("k1: %f\n", norma \* VecNorma(dx, N) / (VecNorma(db, N) \* VecNorma(x, N)));

free(dx);

free(X);

free(db);

free(B);

}

void **K2** (float\*\* A, float\* b, float\* x, int N)

{

int i, j;

float norma = MatNorma(A, N);

srand(time(0));

float\*\* dA = CreateMatrix(N);

float\*\* AA = CreateMatrix(N);

float\* X = (float\*)malloc(N\*sizeof(float));

float\* dx = (float\*)malloc(N\*sizeof(float));

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

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

dA[i][j] = Sign() \* (1.0f\*rand())/RAND\_MAX;

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

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

{

dA[i][j] = dA[i][j] \* 0.01 \* A[i][j];

AA[i][j] = A[i][j] + dA[i][j];

}

Solve(AA, b, X, N);

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

dx[i] = X[i] - x[i];

printf("k2: %f", VecNorma(dx, N) \* MatNorma(AA, N) / (VecNorma(x, N) \* MatNorma(dA, N)));

free(dA);

free(AA);

free(X);

free(dx);

}

int **main**()

{

int N;

int i, j;

FILE\* f = fopen("D:\\Input.txt", "r");

if (f == NULL)

{

printf("ERROR");

return -1;

}

fscanf(f, "%d", &N);

float\*\* A = CreateMatrix(N);

float\* b = (float\*)malloc(N\*sizeof(float));

float\* x = (float\*)malloc(N\*sizeof(float));

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

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

fscanf(f, "%f", &A[i][j]);

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

fscanf(f, "%f", &b[i]);

Solve(A, b, x, N);

if (flag)

{

printf("Can't apply method!");

return -1;

}

flag = 0;

EpsB(A, b, x, N);

K1(A, b, x, N);

K2(A, b, x, N);

if (flag)

{

printf("ERROR");

return -1;

}

FreeMatrix(A, N);

free(b);

free(x);

fclose(f);

return 0;

}

### Генератор матриц

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

//#define POS //Параметр запуска, положительно опред. матрица

//#define MAN //Параметр запуска, введение СЧ с клавиатуры

//#define DETCHANGE //Параметр запуска, деление СЧ на некоторое число

//#define TIMES 30 //Параметр запуска, количество делений

float Random()

{

float sign;

if ((rand() % 2) == 0)

sign = 1.0;

else

sign = -1.0;

return sign \* ((float)rand())/RAND\_MAX;

}

float\*\* CreateMatrix (int N)

{

int i;

float\*\* q = (float\*\*)malloc(N\*sizeof(float\*));

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

q[i] = (float\*)malloc(N\*sizeof(float));

return q;

}

void FreeMatrix (float\*\* matrix, int N)

{

int i;

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

free(matrix[i]);

free(matrix);

}

int Compare (int i, int j)

{

return (i == j);

}

float\* W (int N)

{

float\* q = (float\*)malloc(N\*sizeof(float));

int i;

float norma = 0;

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

{

q[i] = Random();

norma = norma + q[i] \* q[i];

}

norma = sqrtf(norma);

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

q[i] = q[i] / norma;

return q;

}

float\*\* H (int N)

{

int i, j;

float\* V = W (N);

float\*\* q = CreateMatrix(N);

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

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

q[i][j] = Compare(i, j) - 2 \* V[i] \* V[j];

free(V);

return q;

}

float\*\* MultMat (float\*\* A, float\*\* B, int N)

{

int i, j, k;

float\*\* AB = CreateMatrix(N);

float temp;

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

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

{

temp = 0;

for (k = 0; k < N; k++)

temp = temp + A[i][k]\*B[k][j];

AB[i][j] = temp;

}

return AB;

}

float\*\* D (float cond, int N)

{

int i, j;

float\*\* q = CreateMatrix(N);

float min = cond;

#ifndef MAN

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

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

q[i][j] = Compare(i, j) \* cond \*

#ifdef POS

fabsf(Random()) / 10;

#else

Random() / 10;

#endif

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

if (q[i][i] < min)

min = q[i][i];

q[0][0] = cond \* min;

#ifdef DETCHANGE

float times = 1;

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

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

q[j][j] = q[j][j] / pow(10, 1/(float)N);

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

times = times\*q[i][i];

printf("Det: %f\n", times);

#endif

#else

printf("\nEnter eigenvalues: ");

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

scanf("%f", &q[i][i]);

#endif

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

printf("%f ", q[i][i]);

return q;

}

int main()

{

float temp;

int i, j;

int N;

float cond;

srand(time(0));

FILE \*f = fopen("D:\\Input.txt", "w");

printf("Enter cond and N: ");

scanf("%f%d", &cond, &N);

float\*\* d = D(cond, N);

float\*\* h = H(N);

float\*\* dh = MultMat(d, h, N);

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

for (j = i+1; j < N; j++)

{

temp = h[i][j];

h[i][j] = h[j][i];

h[j][i] = temp;

}

float\*\* hdh = MultMat(h, dh, N);

fprintf(f, "%d\n",N);

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

{

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

fprintf(f, "%f ", hdh[i][j]);

fprintf(f, "\n");

}

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

fprintf(f, "%f ", cond \* Random());

FreeMatrix(d, N);

FreeMatrix(h, N);

FreeMatrix(dh, N);

FreeMatrix(hdh, N);

fclose(f);

return 0;

}

## Приложение к части 3

### Библиотека для работы с матрицами

#include "matrix.h"

#include <math.h>

mat matrixInit(int size)

{

int i, j;

float\*\* q = (float\*\*)malloc(size \* sizeof(float\*));

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

{

q[i] = (float\*)malloc(size \* sizeof(float));

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

q[i][j] = 0;

}

return q;

}

void matrixDone(mat matrix, int size)

{

int i;

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

free(matrix[i]);

free(matrix);

}

vec vectorInit(int size)

{

int i;

float\* q = (float\*)malloc(size \* sizeof(float));

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

q[i] = 0;

return q;

}

void vectorDone(vec vector)

{

free(vector);

}

void matXvec(mat matrix, vec vector, int size, vec result)

{

int i, j;

float temp;

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

{

temp = 0;

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

temp = temp + matrix[i][j] \* vector[j];

result[i] = temp;

}

}

float vecInfNorm(vec vector, int size)

{

float max = 0;

int i;

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

if (fabsf(vector[i]) > max)

max = fabsf(vector[i]);

return max;

}

float matInfNorm(mat matrix, int size)

{

float max = 0;

int i, j;

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

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

if (fabsf(matrix[i][j]) > max)

max = fabsf(matrix[i][j]);

}

void vecDvec(vec vec1, vec vec2, vec result, int size)

{

int i;

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

result[i] = vec1[i] - vec2[i];

}

void matDevideDetOnS(mat matrix, int size, float S)

{

int i, j;

float temp = powf(S, 1.0f / (size));

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

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

matrix[i][j] = matrix[i][j] / temp;

}

### Программный код

#define \_CRT\_SECURE\_NO\_WARNINGS  
#include <stdio.h>  
#include "matrix.h"  
#define FILE\_INPUT  
#define FILE\_OUTPUT  
#define EPS 0.0001f

vec RelaxationMethod(float omega, float epsilon, mat A, vec b, int size, int\* k)

{

int run;

int i, j;

\*k = 0;

float discrep = 0;

vec x1 = vectorInit(size);

vec x2 = vectorInit(size);

vec temp1 = vectorInit(size);

vec temp2 = vectorInit(size);

float temp3, temp4;

run = 1;

while (run)

{

\*k = \*k + 1;

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

{

temp3 = 0;

temp4 = 0;

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

temp3 = temp3 + A[i][j] \* x2[j];

for (j = i + 1; j < size; j++)

temp4 = temp4 + A[i][j] \* x1[j];

x2[i] = (1 - omega)\*x1[i] + (omega / A[i][i])\*(b[i] - temp3 - temp4);

}

vecDvec(x1, x2, temp2, size);

discrep = vecInfNorm(temp2, size);

if (discrep < epsilon)

run = 0;

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

x1[i] = x2[i];

}

vectorDone(x2);

vectorDone(temp1);

vectorDone(temp2);

return x1;

}

void Test1()

{

int i, j, k;

int size;

float omega;

float epsilon;

mat A;

vec b;

vec x;

FILE\* input = stdin;

#ifdef FILE\_INPUT

input = fopen("D:\\Input.txt", "r");

#endif // FILE\_INPUT

fscanf(input, "%d%f%f", &size, &omega, &epsilon);

A = matrixInit(size);

b = vectorInit(size);

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

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

fscanf(input, "%f", &A[i][j]);

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

fscanf(input, "%f", &b[i]);

x = RelaxationMethod(omega, epsilon, A, b, size, &k);

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

printf("%f ", x[i]);

printf("\n");

printf("%d", k);

matrixDone(A, size);

vectorDone(b);

vectorDone(x);

fclose(input);

}

void Test2()

{

int i, j, k;

float test;

int size;

float omega;

float epsilon;

mat A;

vec b;

vec x = NULL;

FILE\* input = stdin;

#ifdef FILE\_INPUT

input = fopen("D:\\Input.txt", "r");

#endif // FILE\_INPUT

fscanf(input, "%d%f%f", &size, &omega, &epsilon);

A = matrixInit(size);

b = vectorInit(size);

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

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

fscanf(input, "%f", &A[i][j]);

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

fscanf(input, "%f", &b[i]);

for (i = 1; i <= 19; i++)

{

test = i / 10.0;

x = RelaxationMethod(test, epsilon, A, b, size, &k);

printf("omega: %f - ", test);

printf("k: %d", k);

printf("\n");

vectorDone(x);

}

matrixDone(A, size);

vectorDone(b);

fclose(input);

}

void Test3()

{

int i, j, k;

int size;

float omega;

float epsilon;

mat A;

vec b;

vec x = NULL;

FILE\* input = stdin;

#ifdef FILE\_INPUT

input = fopen("D:\\Input.txt", "r");

#endif // FILE\_INPUT

fscanf(input, "%d%f%f", &size, &omega, &epsilon);

A = matrixInit(size);

b = vectorInit(size);

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

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

fscanf(input, "%f", &A[i][j]);

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

fscanf(input, "%f", &b[i]);

x = RelaxationMethod(omega, epsilon, A, b, size, &k);

printf("1: %d\n", k);

vectorDone(x);

for (i = 1; i <= 10; i++)

{

matDevideDetOnS(A, size, 10);

x = RelaxationMethod(omega, epsilon, A, b, size, &k);

printf("10^-%d: %d\n", i, k);

vectorDone(x);

}

matrixDone(A, size);

vectorDone(b);

fclose(input);

}

## Приложение к части 4

В этой части используется библиотека для работы с матрицами, которая была представлена [ранее](#_Библиотека_для_работы).

### Программный код

**yakobi.c**

e "leaktest.h"

#include <math.h>

#include "yakobi.h"

#include "matrix.h"

#define EPS 0.000001f

#define PI 3.141592f

mat createRotMatrix(int I, int J, float P, int size, int cas)

{

int i, j;

mat q = matrixInit(size);

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

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

q[i][j] = (i == j);

float s, c;

if (cas == 1)

{

s = (float)sqrt(0.5\*(1 - (1 / sqrt(1 + ((double)P\*P)))));

if (P < 0)

s = -s;

c = (float)sqrt(0.5\*(1 + (1 / sqrt(1 + ((double)P\*P)))));

}

else

{

s = sqrtf(0.5);

c = sqrtf(0.5);

}

q[I][I] = c;

q[I][J] = -s;

q[J][I] = s;

q[J][J] = c;

return q;

}

vec methodYakobi(mat matrix, int size, float epsilon, int\* k, mat eigVec)

{

mat rotateMat;

mat temp1 = matrixInit(size);

mat temp2 = matrixInit(size);

mat temp3 = matrixInit(size);

vec q = vectorInit(size);

int i, j;

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

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

temp3[i][j] = (i == j);

float theta;

\*k = 0;

int I = 0, J = 1;

float max = fabsf(matrix[I][J]);

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

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

temp2[i][j] = matrix[i][j];

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

for (j = i + 1; j < size; j++)

{

if (fabsf(temp2[i][j]) > max)

{

I = i;

J = j;

max = fabsf(temp2[i][j]);

}

}

float P;

while (max >= epsilon)

{

if (fabsf(temp2[I][I] - temp2[J][J]) >= EPS)

{

P = 2 \* temp2[I][J] / (temp2[I][I] - temp2[J][J]);

rotateMat = createRotMatrix(I, J, P, size, 1);

}

else

{

P = (temp2[I][I] - temp2[J][J])\*temp2[I][J];

rotateMat = createRotMatrix(I, J, P, size, 0);

}

matXmat(temp3, rotateMat, eigVec, size);

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

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

temp3[i][j] = eigVec[i][j];

matXmat(temp2, rotateMat, temp1, size);

trponMat(rotateMat, size);

matXmat(rotateMat, temp1, temp2, size);

I = 0;

J = 1;

max = fabsf(temp2[I][J]);

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

for (j = i + 1; j < size; j++)

{

if (fabsf(temp2[i][j]) > max)

{

I = i;

J = j;

max = fabsf(temp2[i][j]);

}

}

\*k = \*k + 1;

matrixDone(rotateMat, size);

}

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

q[i] = temp2[i][i];

trponMat(eigVec, size);

matrixDone(temp1, size);

matrixDone(temp2, size);

matrixDone(temp3, size);

return q;

}

**main.c**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "leaktest.h"

#include "matrix.h"

#include "yakobi.h"

#include <stdio.h>

int main()

{

SetDbgMemHooks();

float test, max;

int i, j, k;

int size;

float epsilon;

FILE\* f = fopen("D:\\InputEV.txt", "r");

fscanf(f, "%d%f", &size, &epsilon);

mat A = matrixInit(size);

mat eigVec = matrixInit(size);

vec result;

vec temp = vectorInit(size);

vec temp1 = vectorInit(size);

vec temp2 = vectorInit(size);

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

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

fscanf(f, "%f", &A[i][j]);

result = methodYakobi(A, size, epsilon, &k, eigVec);

printf("Eigenvalues: ");

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

printf("%f ", result[i]);

printf("\n\nEigenvectors: \n");

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

{

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

printf("%f ", eigVec[j][i]);

printf("\n");

}

printf("\nIterations: %d\n", k);

max = 0;

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

{

matXvec(A, eigVec[i], size, temp1);

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

temp2[j] = result[i] \* eigVec[i][j];

vecDvec(temp1, temp2, temp, size);

test = vecInfNorm(temp, size);

if (test > max)

max = test;

}

printf("\nmax(Au-ku): %f", test);

matrixDone(eigVec, size);

vectorDone(result);

matrixDone(A, size);

vectorDone(temp);

vectorDone(temp1);

vectorDone(temp2);

return 0;

}