Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

Институт прикладной математики и механики

Кафедра “Прикладная математика”

Курсовой проект по дисциплине

“Численные методы”

Выполнил студент группы 23631/1 Васильева А.В.

Руководитель Павлова Л.В.

Работа принята (подпись) \_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербург, 2018

Оглавление

[Часть 1. Решение алгебраических и трансцендентных уравнений 4](#_Toc533599207)

[Формулировка задачи и ее формализация 4](#_Toc533599208)

[Алгоритмы методов и условия их применимости 4](#_Toc533599209)

[Предварительный анализ задачи, проверка условий применимости метода 5](#_Toc533599210)

[Тестовый пример с детальными расчетами для задачи малой размерности 6](#_Toc533599211)

[Перечень контрольных тестов для иллюстрации метода средствами пакета MATLAB 7](#_Toc533599212)

[Часть 2. Решение СЛАУ прямыми методами 10](#_Toc533599213)

[Формулировка задачи и ее формализация 10](#_Toc533599214)

[Алгоритм метода и условие его применимости 10](#_Toc533599215)

[Предварительный анализ задачи, проверка условий применимости метода 11](#_Toc533599216)

[Тестовый пример с детальными расчетами для задачи малой размерности 12](#_Toc533599217)

[Модульная структура программы 13](#_Toc533599218)

[Численный анализ решения задачи 14](#_Toc533599219)

[Краткие выводы 15](#_Toc533599220)

[Часть 3. Решение СЛАУ итерационными методами 15](#_Toc533599221)

[Формулировка задачи и ее формализация 15](#_Toc533599222)

[Алгоритм метода и условие его применимости 15](#_Toc533599223)

[Предварительный анализ задачи и условий применимости метода 16](#_Toc533599224)

[Тестовый пример с детальными расчетами для задачи малой размерности 16](#_Toc533599225)

[Модульная структура программы 17](#_Toc533599226)

[Численный анализ решения задачи 17](#_Toc533599227)

[Краткие выводы 18](#_Toc533599228)

[Часть 4. Решение алгебраической проблемы собственных значений. 18](#_Toc533599229)

[Формулировка задачи и ее формализация 18](#_Toc533599230)

[Алгоритм метода и условия его применимости 18](#_Toc533599231)

[Предварительный анализ задачи и условий применимости метода 19](#_Toc533599232)

[Тестовый пример с детальными расчетами для задачи малой размерности 19](#_Toc533599233)

[Модульная структура программы 20](#_Toc533599234)

[Численный анализ решения задачи 20](#_Toc533599235)

[Краткие выводы 21](#_Toc533599236)

[Приложение 21](#_Toc533599237)

# Часть 1. Решение алгебраических и трансцендентных уравнений

## Формулировка задачи и ее формализация

Необходимо определить корни алгебраического и трансцендентного уравнений методом половинного деления и методом простых итераций, где корень - такое значение x, что f(x) = 0.

## Алгоритмы методов и условия их применимости

1. Метод половинного деления

Алгоритм:

1. Вычисляем середину отрезка по формуле
2. Вычисляем f(c). Если f(c) = 0, то корень x = c , завершаем цикл.

Если f(a) \* f(c) < 0, то b = c, иначе a = c

1. Если выходим из цикла, иначе переходим к пункту 1.

Условия применимости:

f(x) – определена и непрерывна на промежутке [a,b].

f(a) \* f(b) < 0

1. Метод простых итераций

Алгоритм:

1. Заменяем уравнение эквивалентным ему
2. Выбираем и строим последовательность:
3. Условие остановки:

Функцию :

, при этом .

на [a,b],

В функции знак ставится в зависимости от знака первой производной функции на выбранном промежутке.

Условия применимости:

f(x) – определена и непрерывна до первой производной на промежутке [a,b].

f(a) \* f(b) < 0

## Предварительный анализ задачи, проверка условий применимости метода

Рассмотрим полином . С помощью теоремы о верхней границе положительных корней полинома, найдем:

Верхняя граница положительных корней:

Нижняя граница положительных корней:

Заменим :

Верхняя граница отрицательных корней:

Заменим :

Нижняя граница отрицательных корней:

Заменим :

Для метода половинного деления функция должна быть определена и непрерывна при всех x на отрезке [a, b], что выполняется на отрезках [] и [-, -]   
Также должно выполняться условие :   
Для положительного корня: Для отрицательного корня:   
  
Рассмотрим трансцендентную функцию = x – cos По графику функции определим, что корень находится на промежутке [0, 1]

= 1+sin(x)

На промежутке [0, 1] и определены и непрерывны.

Также должно выполняться условие :

: -0.4597 < 0

Выполнены условия для метода половинного деления и метода простых итераций.

## Тестовый пример с детальными расчетами для задачи малой размерности

У = х^2+3x

Рассмотрим на промежутке [-1, 1]

*Возьмем ɛ = 0,1*

– знакопостоянная функция на промежутке [-1, 1] => f(x) – монотонна на промежутке [-1, 1]

m1 = 1 M1 = 5

= 0.8

= 0.025



При заданной точности ɛ = 0,1 получаем корень Х =

## Модульная структура программы

1. «main.m» – основной файл из которого запускаются функции для реализации разных методов
2. «func.m» – трансцендентное уравнение (функция);
3. «halfer\_divisions\_cyclel.m» – функция, реализующая метод половинного деления. Принимает на вход уравнение и границы промежутка, на котором ищем корень. Результаты работы записываются в файл result\_MPD.csv
4. «simple\_iterations.m» – функция, реализующая метод простых итераций. Принимает на вход уравнение, производную и границы промежутка, на котором ищем корень.

Результаты работы записываются в файл result\_MPI.csv

1. «graph.m» – функция, которая графически изображает первые 3 итерации метода половинного деления.

## Численный анализ решения задачи

Исследуем влияние заданной точности:

Метод половинного деления для алгебраического уравнения:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *ɛ* | Положительный корень | | Отрицательный корень | |
| итерации | значение корня | итерации | значение корня |
| 10-2 | 5 | 0.9913929688 | 6 | -0.2628398438 |
| 10-4 | 7 | 0.9925913940 | 8 | -0.2627663696 |
| 10-6 | 14 | 0.9925937545 | 15 | -0.2626354882 |
| 10-8 | 20 | 0.9925929499 | 21 | -0.2626351176 |

Корень с помощью функции fzero пакета Mathlab:

Положительный: 0.9925

Отрицательный: -0.2626

Метод половинного деления для трансцендентного уравнения:

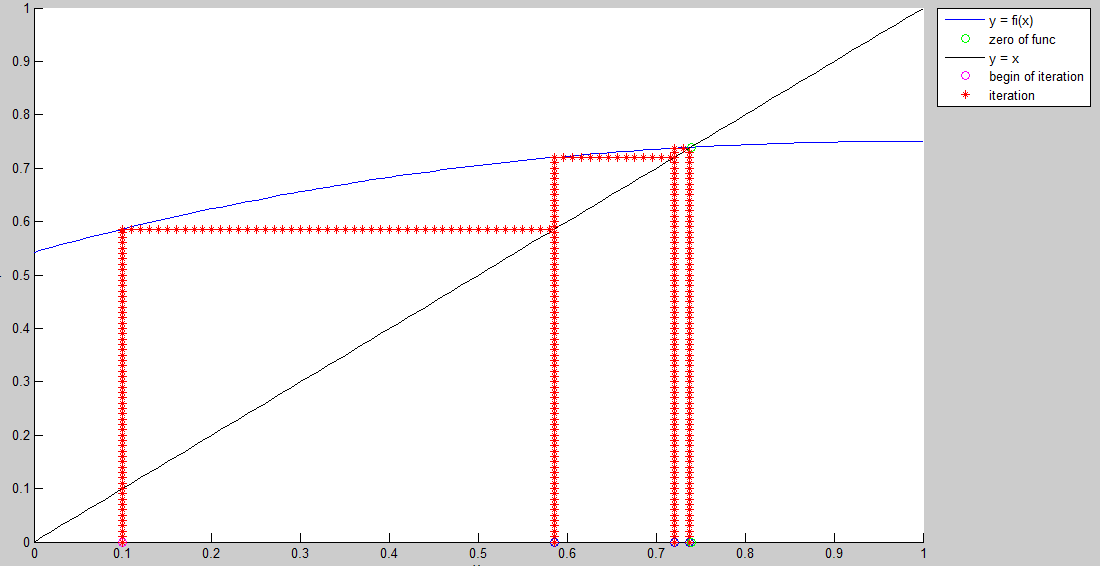
|  |  |  |
| --- | --- | --- |
| *ɛ* | Положительный корень | |
| итерации | значение корня |
| 10-2 | 5 | 0.7343708369 |
| 10-4 | 9 | 0.7400452871 |
| 10-6 | 18 | 0.7390851838 |
| 10-8 | 25 | 0.7390851329 |

Корень с помощью функции fzero пакета Mathlab: 0.739085

Метод простых итераций для трансцендентного уравнения:

|  |  |  |  |
| --- | --- | --- | --- |
| Начальное приближение | *ɛ* | Метод простых итераций | |
| итерации | значение корня |
| 0 | 10-2 | 3 | 0.7388553505 |
| 10-4 | 5 | 0.7390832236 |
| 10-6 | 7 | 0.7390851174 |
| 10-8 | 9 | 0.7390851331 |
| 0.333333 | 10-2 | 2 | 0.7383601944 |
| 10-4 | 4 | 0.7390791014 |
| 10-6 | 6 | 0.7390850831 |
| 10-8 | 8 | 0.7390851328 |
| 0.66666 | 10-2 | 1 | 0.7383737907 |
| 10-4 | 3 | 0.7390792147 |
| 10-6 | 5 | 0.7390850841 |
| 10-8 | 7 | 0.7390851328 |
| 1 | 10-2 | 1 | 0.7400871849 |
| 10-4 | 3 | 0.7390934362 |
| 10-6 | 5 | 0.7390852022 |
| 10-8 | 7 | 0.7390851338 |

Иллюстрация метода простых итераций для трансцендентного уравнения(3 итерации)



## Краткие выводы

С помощью данной работы мы изучили два метода для нахождения корней уравнения, реализовали графическую интерпретацию метода простых итераций. Исходя из полученных данных, можно сказать, что метод простых итераций работает быстрее и точнее метода половинного деления. Начальное приближение влияет на скорость метода простых итераций.

# Часть 2. Решение СЛАУ прямыми методами

## Формулировка задачи и ее формализация

Необходимо найти решение СЛАУ вида используя метод LU-разложения. Проверить вычислительную ошибку для матриц с разными числами обусловленности, сравнив полученное решение с точным решением. Вычисления производим на вещественной квадратной матрице коэффициентов СЛАУ размерности при

## Алгоритм метода и условие его применимости

Мы используем метод LU-разложение. Пусть A - данная матрица, L нижняя (левая) треугольная матрица, U - верхняя (правая) треугольная матрица. Существует теорема:

Квадратная матрица А с ненулевыми минорами однозначно представляется в виде произведения LU нижнетреугольной матрицы L, главная диагональ которой состоит из ненулевых элементов и верхнетреугольной матрицы U с единицами на главной диагонали.

Таким образом задача сводится к решению уравнения вида LUx =b. Для решения введем вектор вспомогательных переменных , тогда эквивалентное уравнение можно переписать в виде системы . Сначала решается система Ly=b , далее система Ux=y

Алгоритм:

Матрицы L и U находим по формулам:

1. Ly=b :

1. Ux=y :

;

.

## Предварительный анализ задачи, проверка условий применимости метода

Условия применимости:

Все главные миноры матрицы А отличны от нуля.

Возьмем матрицу , у которой первый главный минор равен нулю. При разложении получили следующие две матрицы:

,

.

Получившаяся матрица L не соответствует нижнетреугольной матрице, которая должна была бы получиться, при этом U является верхнетреугольной матрицей. Теорема не выполняется.

Т.к. необходимо исследовать матрицы с разными числами обусловленности, рассмотрим метод на двух матрицах – хорошо и плохо обусловленной.

Хорошо обусловленная матрица:

*,* где:

, где W – вектор размера , состоящий из случайных чисел;

*–* треугольная или диагональная матрица размера , состоящая из случайных чисел.

Плохо обусловленная матрица:

Для плохо обусловленной матрицы после вычисления повторяем действие несколько раз.

## Тестовый пример с детальными расчетами для задачи малой размерности

Проверим главные миноры квадратной матрицы A:

1. ;
2. ;

Т.к. все главные миноры матрицы А отличны от нуля, можем применять метод.

,

## Модульная структура программы

* + - 1. “creature.m“ – создает две матрицы: одну плохо обусловленную матрицу Гильберта, вторую хорошо обусловленную матрицу. Создает два вектора свободных членов. Записывает данные в файл.

1. double\*\* СreatingArray(int sizeLine, int sizeColumn) – выделение памяти под матрицу
2. void FreeDoubleArray(double \*\*Array, int sizeLine, int sizeColumn) – освобождение памяти, выделенной под матрицу
3. void PrintDoubleArray(double \*\*Array, int siseLine, int sizeColumn) – печать матрицы
4. void Nevyazka(double \*\*A, double \*B, double\* X) – вычисление вектора невязки
5. double Norma(double\*\* A) – вычисление нормы матрицы
6. void Transposition(double \*\*A, double \*\*B)- транспонирование матрицы
7. void MartixVectorMultiplication(double \*\*A, double \*B, double \*C) – перемножение матрицы и вектора
8. void MartixMultiplication(double \*\*A, double \*\*B, double \*\*C) – перемножение матриц
9. void LU(double \*\*A, double \*\*L, double \*\*U, int size) – создание матриц L и U
10. double\* Decision(double \*\*L, double \*\*U, double \*B, int size) – нахождение решения с помощью полученных L и U
11. double\*\* VozmushcheniyeA(double\*\* A, int size)

double\* VozmushcheniyeB(double\* B, int size) – внесение возмущения в матрицу А и вектор В

1. double\* Sum(double\* X, double\* X1, int size) – сложение векторов
2. double\* Diff(double\* X, double\* X1, int size)

double\*\* DiffMatr(double\*\* X, double\*\* X1, int size) – разница между изначальным результатом и результатом, полученным после внесения возмущения

1. double SearchK1(double\* B, double\* B1, double\* X, double\* X1, int size)

double SearchK2(double\*\* A, double\*\* A1, double\* X, double\* X1, int size) – вычисление коэффициентов

## Численный анализ решения задачи

Справедливы неравенства:

Нас интересуют коэффициенты, при которых получается равенство. Поэтому получаются такие формулы:

Для хорошо обусловленной матрицы:

Cond(A) = 657.3369;

Внесение возмущений в b.

Внесение возмущений в A.

Для плохо обусловленной матрицы:

Cond(A) = 2.6791e+13;

Внесение возмущений в b.

Внесение возмущений в A.

## Краткие выводы

С помощью данной работы мы изучили метод LU-разложения для нахождения вектора-решения уравнения вида , а также проверили вычислительную ошибку для матриц с разными числами обусловленности. Коэффициент при внесении возмущений в вектор b был меньше, чем коэффициент при внесении возмущений в матрицу А. В независимости от числа обусловленности матрицы, коэффициент был небольшим. Коэффициент при внесении возмущений в матрицу А был маленьким для матрицы с маленьким числом обусловленности, и большим для матрицы с большим числом обусловленности. Коэффициент не превосходит числа обусловленности матрицы.

# Часть 3. Решение СЛАУ итерационными методами

## Формулировка задачи и ее формализация

Необходимо найти решение СЛАУ вида используя метод Зейделя, представив систему в удобном для итераций виде. Исследовать точность решения, когда определитель матрицы близок к 0.

## Алгоритм метода и условие его применимости

Суть итерационного метода состоит в нахождении по приближённому значению величины следующего приближения. Метод Зейделя является модификацией метода простых итераций. При нахождении i-й компоненты (k+1)-го приближения сразу используются уже найденные компоненты (к+1) -го приближения с меньшими номерами 1,2,3,…,i−1.

Можем записать в матричной форме:

= L\* + U\* + β, где L – нижнетреугольная матрица, U – верхнетреугольная матрица, являющиеся LU-разложением матрицы А.

*Условия сходимости:*

Можем представить систему в виде x = αx + β.

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

Т.к. , то является достаточным условием сходимости.

Т.к. метод Зейделя работает для СЛАУ, в которых матрица А - симметричная и положительно определенная, то необходимо предусмотреть действия в случае, когда исходная матрица А не удовлетворяет требованиям. В этом случае систему можно свести к подходящей с помощью домножения обеих частей равенства слева на , получив таким образом систему \*А\*х = \*b, которая является нормальной, при этом число обусловленности матрицы возрастает квадратично.

*Условие остановки:*

Решение получено с заданной точностью ɛ, если: || - || < \*ε.

*Алгоритм метода Зейделя:*

1. \*А\*х = \*b, если система изначально не является нормальной, иначе сразу перейти к пункту 2.
2. Преобразовать систему к виду x = αx + β: , =
3. Задать начальное приближение решение и *к* = 0
4. Вычислить вектор по формуле:
5. Если || - || ≥ ε, увеличить *к* на 1 и повторить пункт 4.

## Предварительный анализ задачи и условий применимости метода

А - симметричная и положительно определенная матрица.

## Тестовый пример с детальными расчетами для задачи малой размерности

α =

= β =

При решении методом Зейделя с точностью :

Решение -

Вектор невязки -

Количество итераций - 7

## Модульная структура программы

1. “Zeidel.m” - создает матрицу А и вектор В, вычисляет число обусловленности.
2. double\*\* СreatingArray(int sizeLine, int sizeColumn) – выделение памяти под матрицу
3. void FreeDoubleArray(double \*\*Array, int sizeLine, int sizeColumn) – освобождение памяти, выделенной под матрицу
4. void PrintDoubleArray(double \*\*Array, int siseLine, int sizeColumn) – печать матрицы
5. void Nevyazka(double \*\*A, double \*B, double\* X) – вычисление вектора невязки
6. double Norma(double\*\* A) – вычисление нормы матрицы
7. void Transposition(double \*\*A, double \*\*B)- транспонирование матрицы
8. void MartixVectorMultiplication(double \*\*A, double \*B, double \*C) – перемножение матрицы и вектора
9. void MartixMultiplication(double \*\*A, double \*\*B, double \*\*C) – перемножение матриц
10. void CreateAlfBet(double \*\*A, double \*B, double \*\*alfa, double \*betta) – создание матрицы α и β
11. int MmethodZeidel(double\*\* alfa, double \*betta, double \*x1, int param)- решение СЛАУ методом Зейделя, подсчет количества итераций.

## Численный анализ решения задачи

.

|  |  |  |  |
| --- | --- | --- | --- |
| det(A) | cond(A) | Кол-во итераций | Невязка |
| 59 | 30.0004 | 34 | 6.908284e-007 |
| 3.628800e-009 | 144,576071 | 146 | 3.040201e-007 |
| 1.04018e+003 | 595.2294 | 2032 | 4.652414e-006 |
| -2.0438e-004 | 2.4682e+003 | 2357 | 8.398626e-005 |
| 1.7391e-008 | 3.4278e+003 | 3642 | 2.364117е-006 |
| 6.5002e-015 | 1.7236e+004 | 43880 | 1.367014e-005 |
| 5,02301e-019 | 1,125418+007 | 213224 | 2.205160e-004 |

## Краткие выводы

В ходе исследования было выяснено, что метод Зейделя хорошо сходится для матриц с небольшим числом обусловленности. При росте числа обусловленности, резко увеличивается количество итераций и погрешность вычислений, метод сходится, но на бесконечности, поэтому его применение неэффективно. Когда определитель матрицы близок к 0, метод работает так же, как и в случае с определителем существенно отличающимся от 0.

# 

# Часть 4. Решение алгебраической проблемы собственных значений.

## Формулировка задачи и ее формализация

Решить алгебраическую проблему собственных значений , исследовать сходимость при хорошей и плохой отделимости искомого собственного числа. Итерационным методом Якоби найти собственные числа λ и соответствующие им собственные векторы X.

λ – собственное число матрицы А, если существует ненулевой вектор Х такой, что A\*Х = λ\*Х.

Х - собственный вектором матрицы А, соответствующий собственному числу λ.

## Алгоритм метода и условия его применимости

Условие применимости:

А – симметричная матрица, т.е. АТ = А.

Алгоритм метода:

1. Выбираем из наддиагональной матрицы максимальный по модулю элемент

|| =

1. Если || ≥ , то находим угол поворота:

, иначе выходим из цикла, переходим к пункту 5.

1. Составляем матрицу поворота:

:

;

1. =
2. , , … ,

## Предварительный анализ задачи и условий применимости метода

Задаем матрицу А в MatLab таким образом, чтобы она была симметричной.

## Тестовый пример с детальными расчетами для задачи малой размерности

1) А =

Максимальный элемент наддиагональной матрицы: 5

=

=

Максимальный элемент наддиагональной матрицы:

=

=

Продолжаем алгоритм пока не достигнем требуемой точности, в данном примере В конечном счете получим следующие результаты:

## Модульная структура программы

1. “Jacobi.m” - создает симметричную матрицу А вычисляет число обусловленности.
2. double\*\* СreatingArray(int sizeLine, int sizeColumn) – выделение памяти под матрицу
3. void FreeDoubleArray(double \*\*Array, int sizeLine, int sizeColumn) – освобождение памяти, выделенной под матрицу
4. void PrintDoubleArray(double \*\*Array, int siseLine, int sizeColumn) – печать матрицы
5. void Diff (double \*\*A, double \*\*B) – вычисление
6. double Norma(double\*\* A) – вычисление нормы матрицы
7. void Transposition(double \*\*A, double \*\*B)- транспонирование матрицы
8. void MartixMultiplication(double \*\*A, double \*\*B, double \*\*C) – перемножение матриц
9. double MaxFind(double \*\*A)- поиск максимального элемента не лежащего на диагонали
10. void MatrixRotation(double \*\*G, double \*\*GT, double \*\*A)- вычисляет матрицу поворота
11. double \*\*Jacobi(double \*\*A, double \*\*G, double \*\*GT, double \*\*B) – функция решает задачу нахождения собственных значений, преобразуя исходную матрицу А в диагональную

## Численный анализ решения задачи

Вычисления проводим с точностью

|  |  |  |  |
| --- | --- | --- | --- |
| Число обусловленностей | Отделимость собственных чисел | Вектор | Кол-во итераций |
| 4.3478 | Хорошая |  | 89 |
| 1.1335e+005 | Плохая | 9.721732e-007 | 68 |
| 37.0889 | Плохая | 4.295100e-007 | 45 |
| 1.7637e+005 | Хорошая | 6.569260e-007 | 92 |

## Краткие выводы

Метод Якоби более эффективен для матриц с плохой отделимостью собственных чисел, если же отделимость хорошая, то количество итераций увеличивается. Также количество итераций возрастает при увеличении числа обусловленности.

# Заключение по итогам выполнения курсового проекта

В результате выполнения курсовой работы был исследован ряд методов численного решения алгебраических задач.

В части 1 были рассмотрены методы решения алгебраических и трансцендентных уравнений, а именно метод половинного деления и метод простых итераций. Сравнение этих двух итерационных методов друг с другом показало, что более эффективным методом является метод простых итераций.

В частях 2 и 3 были изучены методы решения СЛАУ. Прямые методы дают более точное решение, но они уступают итерационным методам в рациональности использования памяти.

В 4 части было произведено решение алгебраической системы собственных значений с помощью метода Якоби. Результаты показали, что метод наиболее эффективен на матрицах с плохой отделимостью собственных чисел и с небольшим числом обусловленности.

На основе работ 2-4 можно сделать вывод, что число обусловленности играет большую роль для результата вычислений. Оно показывает, насколько внесение возмущений в матрицу может изменить результат, а это оказывает существенное влияние т.к. при компьютерных вычислениях погрешности вносятся постоянно. Можно заметить, что лучший результат показывают матрицы с маленьким числом обусловленности.

## Приложение

Задание хорошо обусловленной матрицы и вектора b:

n = 10;

D = eye(n);

D(n, n) = 2;

cond(D)

W = rand(n, 1);

nrm = norm(W, 2);

E = eye(n);

H = E-2\*W\*W'\*(nrm^2);

A = H\*D\*H';

cond(A)

B = rand(n, 1)\*n;

Задание плохо обусловленной матрицы и вектора b:

for i = 1:1:n

for j = 1:1:n

A1(i, j) = 1/(i+j-1);

end

end

cond(A1)

determinant = det(A1)

B1 = randn(n, 1);

LU-разложение матрицы.

void LU(double \*\*A, double \*\*L, double \*\*U, int size)

{

double sum;

int i, j, k;

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

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

U[i][j] = A[i][j];

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

{

for(i = k-1; i < size; i++)

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

L[j][i]=U[j][i]/U[i][i];

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

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

U[i][j]=U[i][j]-L[i][k-1]\*U[k-1][j];

}

}

Нахождение векторов Х и У.

double\* Decision(double \*\*L, double \*\*U, double \*B, int size)

{

double \*Y, \*X, sum;

int i, k;

Y = (double\*)malloc(size \* sizeof(double));

X = (double\*)malloc(size \* sizeof(double));

Y[0] = B[0];

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

{

sum = 0;

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

sum += L[i][k] \* Y[k];

Y[i] = B[i] - sum;

}

X[size-1] = Y[size-1]/U[size-1][size-1];

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

{

sum = 0;

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

sum += U[size-1-i][size-1-k] \* X[size-1-k];

X[size-1-i] = (Y[size-1-i] - sum)/U[size-1-i][size-1-i];

}

free(Y);

return X;

}

Нахождение вектора невязки.

double\* Nevyazka(double \*\*A, double \*B, double\* X, int size)

{

double\* C;

int i, j;

C = (double \*)malloc(size \* sizeof(double));

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

{

C[i] = 0;

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

C[i] += A[i][j] \* X[j];

C[i] = C[i] - B[i];

}

return C;

}

Внесение возмущения в матрицу А

double\*\* VozmushcheniyeA(double\*\* A, int size)

{

double\*\* A1;

int i, j;

A1 = СreatingArray(SIZE, SIZE);

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

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

A1[i][j] = A[i][j] + Random(-A[i][j]/100, A[i][j]/100);

return A1;

}

Внесение возмущения в вектор b.

double\* VozmushcheniyeB(double\* B, int size)

{

double\* B1;

int i;

B1 = (double \*)malloc(SIZE \* sizeof(double));

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

B1[i] = B[i] + Random(-B[i]/100, B[i]/100);

return B1;

}

Вычисление коэффициента при внесении возмущения в вектор b.

double SearchK1(double\* B, double\* B1, double\* X, double\* X1, int size)

{

double normB, normDB, normX, normDX;

normB = NormaStolb(B, size);

normDB = NormaStolb(Diff(B, B1, size), size);

normX = NormaStolb(X, size);

normDX = NormaStolb(Diff(X, X1, size), size);

printf("\n\nK1: %e\n", normB \* normDX / (normDB \* normX));

return normB \* normDX / (normDB \* normX);

}

Вычисление коэффициента при внесении возмущения в матрицу А.

double SearchK2(double\*\* A, double\*\* A1, double\* X, double\* X1, int size)

{

double normA, normDA, normDX, normXDX;

normA = Norma(A, size);

normDA = Norma(DiffMatr(A, A1, size), size);

normDX = NormaStolb(Diff(X, X1, size), size);

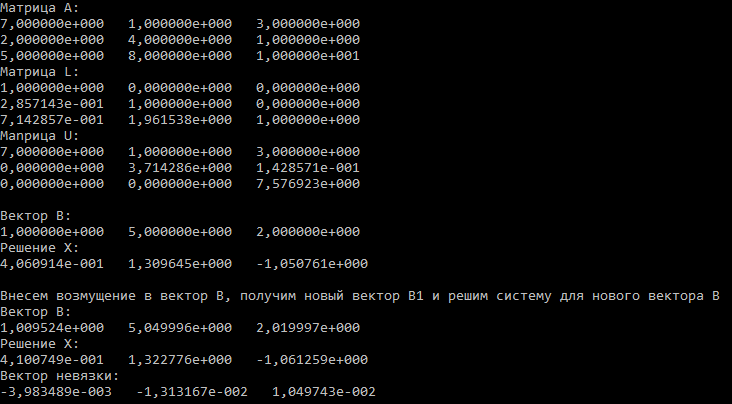
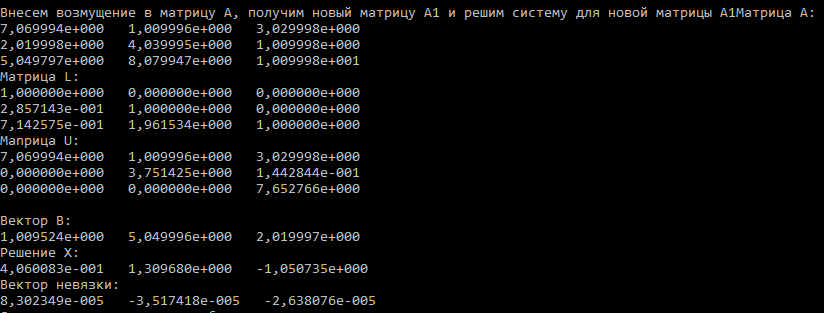
normXDX = NormaStolb(Sum(X, Diff(X, X1, size), size), size);

Diff(X, X1, size);

printf("\n\nK2: %e\n", normA \* normDX / (normDA \* normXDX));

return normA \* normDX / (normDA \* normXDX);

}

Результат работы программы для задачи малой размерности.