# Санкт-Петербургский политехнический университет Петра Великого

ИПММ

Кафедра “Прикладная математика”

Курсовой проект по дисциплине

“Численные методы”

Студентка группы № 23631/3:

Курова Анна Николаевна

Преподаватель:

Павлова Л.В.

Оглавление

[Санкт-Петербургский государственный политехнический университет 0](file:///C:\Users\ASUS\Desktop\Численные%20методы\Численные%20методы2.docx#_Toc533073708)

[Работа №1: Численные методы решения нелинейных уравнений. Метод хорд и метод половинного деления. 2](#_Toc533073709)

[Формулировка задачи: 2](#_Toc533073710)

[Алгоритмы методов и условия их применимости: 2](#_Toc533073711)

[Предварительный анализ решения задачи: 2](#_Toc533073712)

[Тестовый пример: 3](#_Toc533073713)

[Модульная структура программы: 4](#_Toc533073714)

[Проверка условий применимости метода для алгебраического: 4](#_Toc533073715)

[Проверка условий применимости метода для трансцендентного: 6](#_Toc533073716)

[Численный анализ решения задачи: 7](#_Toc533073717)

[Вывод: 8](#_Toc533073718)

[Работа №2: Решение СЛАУ прямыми методами. LDR-разложение. 8](#_Toc533073719)

[Формулировка задачи: 8](#_Toc533073720)

[Алгоритм метода и условия его применимости: 8](#_Toc533073721)

[Проверка условий применимости метода: 9](#_Toc533073722)

[Тестовый пример: 9](#_Toc533073723)

[Модульная структура программы: 10](#_Toc533073724)

[Численный анализ решения задачи: 12](#_Toc533073725)

[Вывод: 12](#_Toc533073726)

[Работа №3: Решение СЛАУ итерационными методами. Метод Зейделя. 13](#_Toc533073727)

[Формулировка задачи: 13](#_Toc533073728)

[Алгоритм метода и условия его применимости: 13](#_Toc533073729)

[Проверка условий применимости метода: 14](#_Toc533073730)

[Тестовый пример: 14](#_Toc533073731)

[Модульная структура программы: 14](#_Toc533073732)

[Численный анализ решения задачи: 15](#_Toc533073733)

[Вывод: 16](#_Toc533073734)

[Работа №4: Решение алгебраической проблемы собственных значений. Метод Якоби. 16](#_Toc533073735)

[Формулировка задачи: 16](#_Toc533073736)

[Алгоритм метода и условия его применимости: 16](#_Toc533073737)

[Проверка условий применимости метода: 16](#_Toc533073738)

[Тестовый пример: 16](#_Toc533073739)

[Модульная структура программы: 17](#_Toc533073740)

[Численный анализ решения задачи: 18](#_Toc533073741)

[Вывод: 19](#_Toc533073742)

# Работа №1: Численные методы решения нелинейных уравнений. Метод хорд и метод половинного деления.

## Формулировка задачи:

Поиск корней алгебраического и трансцендентного уравнений с помощью метода хорд и метода половинного деления

## Алгоритмы методов и условия их применимости:

Метод хорд:

Для корректной работы метода хорд должны выполняться следующие условия:

1. f(a)\*f(b)<0
2. (f’(x) > 0) || (f’(x) < 0) для любого x из [a,b])
3. (f’’(x) > 0) || (f’’(x) < 0) для любого x из [a,b])

Выбор фиксированной точки:

Если f(a)\*f’’(a)>0, то x0 = a

Если f(b)\*f’’(b)>0, то x0 = b

Итерационная формула:

xi+1 = xi - f(xi)/(f(xi)-f(x0))\*(xi – x0)

пока выполняется условие |(xi+1-xi)| > e\*m/(M-m)

m = inf|f'(x)|

M = max|f'(x)|

Метод половинного деления:

Для корректной работы метода половинного деления должны выполняться следующие условия:

1. f(a)\*f(b)<0
2. f(x) непрерывна и ограничена на промежутке [a,b]

Если f(a)\*f(c)< 0, то положим b = c,в противном случае a = c

Если b-a >= e, то снова ищем с

## Предварительный анализ решения задачи:

Выберем промежуток [0.8,1.2]

На нем первая и вторая производные сохраняют свой знак.

f(0.8)\*f(1.2)<0

Следовательно, на данном промежутке существует корень

Выберем промежуток [-0.3,-0.2]

На нем первая и вторая производные сохраняют свой знак.

f(-0.3)\*f(-0.2)<0

Следовательно, на данном промежутке существует корень

Теорема о верхней границе:

Пусть а0>0, тогда все положительные корни не превосходят величину:

m – номер первого отрицательного

a’ – max{|ai| | ai < 0} 1<=i<=n

p = 8.7345x4 + 0.1021x3 - 4.3251x2 - 3.6443x - 0.6985

– верхняя граница положительных корней

Заменим :

Тогда – нижняя граница положительных корней

Заменим :

Тогда – верхняя граница отрицательных корней

Заменим :

Тогда – нижняя граница отрицательных корней

## Тестовый пример:

y = x2 – 1

Промежуток:[0.5,1.5]

e = 0.001

Функция возрастает на всем промежутке:

m = 0.5;

M = 1.5;

y’ = 2x > 0 на промежутке [0.5,1.5]

y’’ = 2 > 0 на промежутке [0.5,1.5]

y(0.5)\*y’’(0.5) = -0.75\*2 = -1.5 < 0

y(1.5)\*y’’(1.5) = 1,25\*2 = 2.5 > 0 – постоянная точка

xi+1 = xi - f(xi)/(f(xi)-f(x0))\*(xi – x0)

x2 = 0.5 - ((-0.75)/(-0.75-1.25))\*(0.5 – 1.5) = 0,875

|x2-x1| = |0,875 - 0.5| = 0.375 > 0.001\*0.5/(1.5+0.5) = 0.00025

x3 = 0.875 - ((−0,234375)/( −0,234375-1.25))\*(0.875 – 1.5) = 0,97368421052

|x3-x2| = |0,97368421052 - 0,875| = 0,09868421052 > 0.00025

x4 = 0,97368421052 - ((−0,05193905819)/( −0,05193905819-1.25))\*( 0,97368421052 – 1.5) = 0,99468085108

|x4-x3| = |0,99468085108 - 0,97368421052| = 0,02099664056 > 0.00025

И т.д.

## Модульная структура программы:

double GetFunction(double x) – функция, которая принимает значение и подставляет его в данный по заданию полином, она возвращает значение полинома в данной точке

int CheckSigns(double x) – проверяет знак функции в точке x

int CheckDer(double a, double b, int der) – функция, которая проверяет знак первой и второй производной, она принимает сам промежуток [a,b] и переменная der показывает проверяем мы первую или вторую производную

double GetDerivative(double result1, double result2) – функция, которая возвращает значение производной, она принимает значение result1 (значение функции в точке x) и result2 (значение функции в x + h)

double GetSecDerivative(double x) – функция, которая возвращает значение второй производной в точке x

double ChordsMethod(double x0, double x1, int\* iteration) – функция, которая вычисляет корень с помощью метода хорд, принимает промежуток и указатель на переменную для подсчета итераций, возвращает значение корня на промежутке

double BisectionMethod(double a, double b, int\* iteration) – функция, которая вычисляет корень с помощью метода половинного деления, принимает промежуток и указатель на переменную для подсчета итераций, возвращает значение корня на промежутке

1. Алгебраическое уравнение

Данный полином:

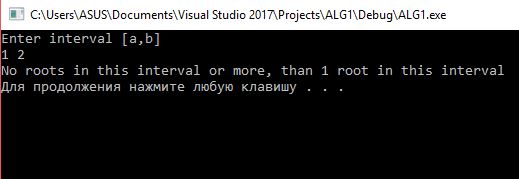
f(x) = 8.7354x4+0.1021x3-4.3251x2-3.6443x-0.6985

## Проверка условий применимости метода для алгебраического:

Рассмотрим программу:

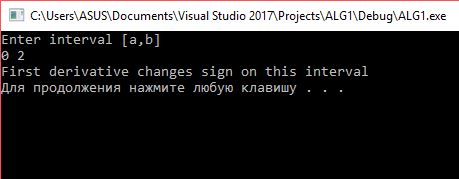
1. Сначала проверяется условие номер 1.

Если оно не выполняется, то программа выводит, что корней в данном промежутке нет или в этом промежутке больше одного корня



1. Затем условие номер 2

Если оно не выполняется, то программа выводит, что производная функции не сохраняет свой знак

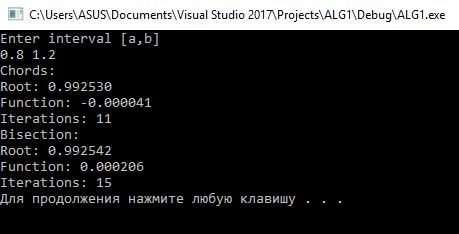


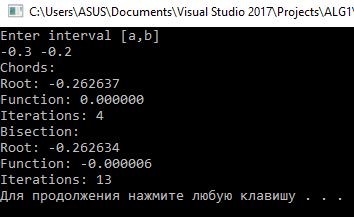
1. Далее проверяется условие номер 3

Если оно не выполняется, то программа выводит, что вторая производная функции не сохраняет свой знак

Для варианта номер 6 нет такого промежутка

Если все условия выполняются, то программа рассчитывает корень с помощью двух методов и выводит значение вместе с количеством итераций, потраченных на поиск корня



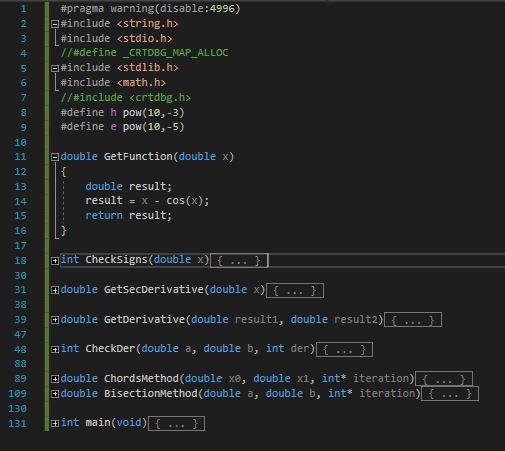


1. Трансцендентное уравнение

Данное уравнение:

f(x) = x – cosx

В коде программы на Си нет изменений, меняется только заданная функция в double GetFunction(double x)



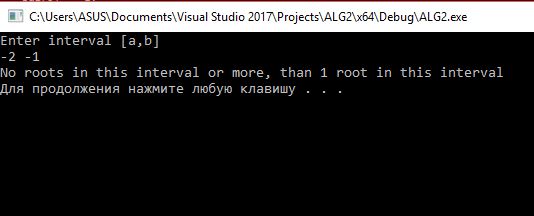
Весь функционал сохраняется

## Проверка условий применимости метода для трансцендентного:

Рассмотрим программу:

1. Сначала проверяется условие номер 1.

Если оно не выполняется, то программа выводит, что корней в данном промежутке нет или в этом промежутке больше одного корня



1. Затем условие номер 2

Если оно не выполняется, то программа выводит, что производная функции не сохраняет свой знак

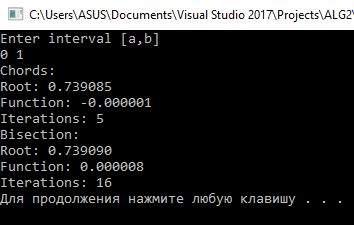
Для варианта номер 6 нет такого промежутка

1. Далее проверяется условие номер 3

Если оно не выполняется, то программа выводит, что вторая производная функции не сохраняет свой знак

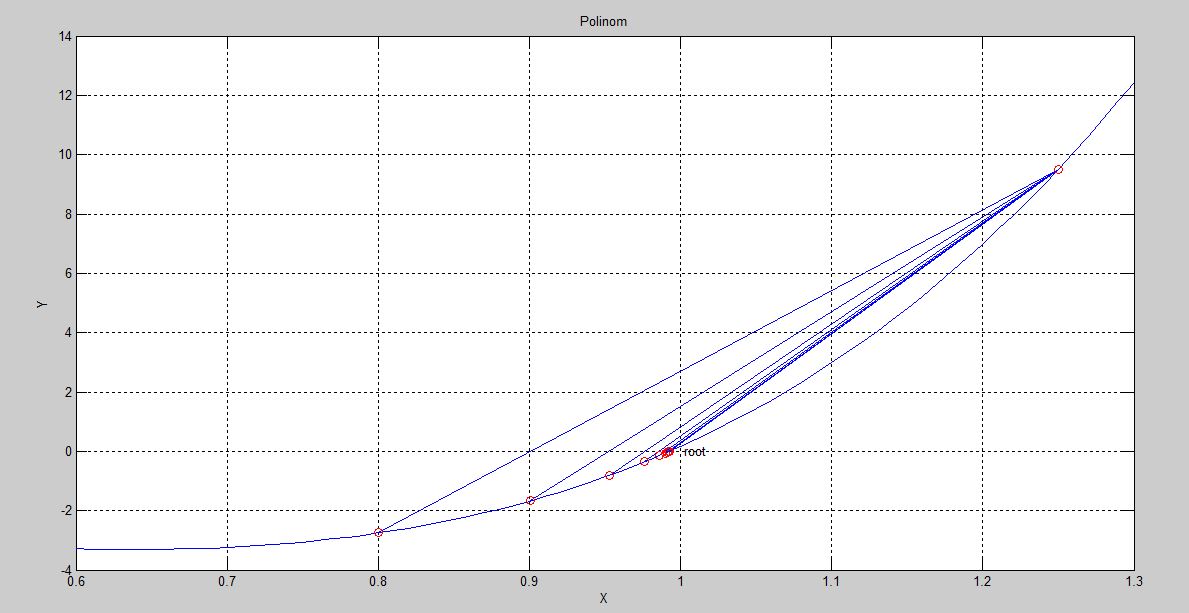
Для варианта номер 6 нет такого промежутка

Если все условия выполняются, то программа рассчитывает корень с помощью двух методов и выводит значение вместе с количеством итераций, потраченных на поиск корня



Работа с пакетом MATLAB:

Проиллюстрируем метод хорд с помощью MatLab:



Вычислим корень полинома с помощью fzero:

х1 = 0,992542

Вычислим корень полинома с помощью метода хорд:

х1 = 0,992550

## Численный анализ решения задачи:

Выберем постоянную точку:

Промежуток: [0.8;1.2]

f(0.8)\*f’’(0.8) = −2,7520776\*1,34666 = −3,70611282082 < 0

f(1.2)\*f’’(1.2) = 6,988484\*8,3751= 58,5292523484 > 0 – постоянная точка b

Промежуток: [-0.3;-0.2]

f(-0.3)\*f’’(-0.3) = 0,07352375\*−3,569625= −0,26245221609 < 0

f(-0.2)\*f’’(-0.2) = −3,99614 \*−0,1294856= 0,51744258558 > 0 – постоянная точка b

Корни алгебраического:

Метод хорд:

На промежутке [0.8,1.2] – x = 0.992530

На промежутке [-0.3,-0.2] – x = -0.262637

Половинное деление:

На промежутке [0.8,1.2] – x = 0.992542

На промежутке [-0.3,-0.2] – x = -0.262634

Исследуем влияние заданной точности:

Промежуток: [0.8,1.2]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| е | Метод хорд | | Метод половинного деления | |
| итерации | значение корня | итерации | значение корня |
| 10-2 | 5 | 0.991333 | 5 | 0.987500 |
| 10-3 | 7 | 0.992393 | 8 | 0.992188 |
| 10-5 | 11 | 0.992530 | 15 | 0.992542 |
| 10-7 | 16 | 0.992532 | 21 | 0.992532 |

Корни трансцендентного:

Метод хорд:

На промежутке [0,1] – x = 0.739085

Половинное деление:

На промежутке [0,1] – x = 0.739090

Исследуем влияние заданной точности:

Промежуток: [0,1]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Точность | Метод хорд | | Метод половинного деления | |
| итерации | значение корня | итерации | значение корня |
| 10-2 | 3 | 0.738945 | 6 | 0.734375 |
| 10-3 | 4 | 0.739078 | 9 | 0.740234 |
| 10-5 | 5 | 0.739085 | 16 | 0.739085 |
| 10-7 | 7 | 0.739085 | 23 | 0.739085 |

## Вывод:

На рассмотренных уравнениях, мы убедились, что на данных примерах итерационный метод хорд работает быстрее и точнее метода половинного деления

# Работа №2: Решение СЛАУ прямыми методами. LDR-разложение.

## Формулировка задачи:

Методом LDR-разложения матрицы найти решение заданной системы:

Ax=b

Сравнить точность решения СЛАУ при маленьких и больших числах обусловленности

## Алгоритм метода и условия его применимости:

Все главные миноры матрицы А отличны от нуля

A = LDR

1 0 … 0

l11 1 0 … 0

L = l21 l22 1

…

ln1 … 1

d1 0 … 0

0 d2 0 … 0

D = 0 0 d3

…

0 … dn

1 r12  … r1n

0 1 r23 … r2n

D = 0 0 1

…

0 … rnn

a11 = d1 (d1 != 0, если выполняется теорема о единственности разложения)

r1j =

li1 =

dm =

j>m:

rmj =

i>m:

lim =

## Проверка условий применимости метода:

Все главные миноры матрицы должны быть отличны от 0

Исходя из определения главных миноров можно заключить, что:

det(A) != 0

Проверяем с помощью MATLAB

## Тестовый пример:

1 2 3 3

A = 3 5 7 = 0

1 3 4 1

det(A) =  1·5·4 + 2·7·1 + 3·3·3 - 3·5·1 - 1·7·3 - 2·3·4 = 20 + 14 + 27 - 15 - 21 - 24 = 1 => det(A)!= 0

L:

1.000000 0.000000 0.000000

3.000000 1.000000 0.000000

1.000000 -1.000000 1.000000

D:

1.000000 0.000000 0.000000

0.000000 -1.000000 0.000000

0.000000 0.000000 -1.000000

R:

1.000000 2.000000 3.000000

0.000000 1.000000 2.000000

0.000000 0.000000 1.000000

L\*y = b:

1.000000 0.000000 0.000000 3.000000 3.000000

3.000000 1.000000 0.000000 \* 0.000000 => y = -9.000000

1.000000 -1.000000 1.000000 1.000000 -11.000000

D\*z = y:

1.000000 0.000000 0.000000 3.000000 3.000000

0.000000 -1.000000 0.000000 \* -9.000000 => z = 9.000000

0.000000 0.000000 -1.000000 -11.000000 11.000000

R\*x = z:

1.000000 2.000000 3.000000 3.000000 -4.000000

0.000000 1.000000 2.000000 \* 9.000000 => x = -13.000000

0.000000 0.000000 1.000000 11.000000 11.000000

## Модульная структура программы:

void CreateMemory(matrix\_data \*M)

Входные данные: данная матрица

Выделяет память для матрицы

void DestroyMemory(matrix\_data \*M)

Входные данные: данная матрица

Освобождает память для матрицы

void GetZeroMatrix(matrix\_data \*M)

Входные данные: данная матрица

Заполнят матрицу нулями

void GetUnitMatrix(matrix\_data \*M)

Входные данные: данная матрица

Заполнят матрицу единицами

void GetMatrix(FILE\* f, matrix\_data \*Matr)

Входные данные: Файл с матрицей и матрица

Читает матрицу из файла и заполняет Matr

void PrintMatrix(matrix\_data \*Matrix)

Входные данные: данная матрица

Печатает матрицу

void CopyMatrix(matrix\_data \*M, matrix\_data \*N, int size)

Входные данные: данная матрица, матрица, в которую нужно скопировать данные и размер матрицы

Копирует матрицу

void CheckResult(matrix\_data \*x, matrix\_data \*Matr1, matrix\_data \*vect2)

Входные данные: вектор ответа, данная матица, вектор свободных коэффициентов

Проверяет правильность нахождения x

double SumD(int size, matrix\_data \*L, matrix\_data \*D, matrix\_data \*R)

double SumR(int size, int j, matrix\_data \*L, matrix\_data \*D, matrix\_data \*R)

double SumL(int size, int m, matrix\_data \*L, matrix\_data \*D, matrix\_data \*R)

Входные данные: матрицы L, D, R, размер, счётчик

Считает сумму элементов для формул в LDR разложении

double SumY(int m, matrix\_data \*L, matrix\_data \*y)

double SumX(int m, matrix\_data \*R, matrix\_data \*z)

Входные данные: счетчик, данные матрицы разложения и векторы

Вычисляет вспомогательные векторы

matrix\_data\* EnterError(matrix\_data \*M)

Входные данные: данная матрица

Вносит погрешность в матрицу

double GetNorm(matrix\_data \*M)

Входные данные: данная матрица

Вычисляет норму матрицы

double GetK(matrix\_data \*dx, matrix\_data \*x, matrix\_data \*dy, matrix\_data \*y)

Входные данные: данные матрицы

Вычисляет коэффициент

matrix\_data\* GetError(matrix\_data\* x, matrix\_data \*x1)

Входные данные: вектор и вектор с погрешностью

Вычисляет разность векторов

matrix\_data\* LDR(matrix\_data \*A, matrix\_data \*v)

Входные данные: матрица и вектор свободных коэффициентов

Вычисляет результат с помощью LDR метода

## Численный анализ решения задачи:

Хорошо обусловленная матрица:

Cond(A) = 6.0341;

||𝑥|/||𝑥||<= cond(A)∗||𝑏||/|𝑏||

Посмотрим, когда достигается равенство, если число обусловленности мало:

Погрешность в b:

||𝑥|/||𝑥||=𝑘1∗||𝑏||/|𝑏||

𝑘1 = 1.411736

Погрешность в A.

||𝑥||/|𝑥+𝑥||=𝑘2∗||A|/||𝐴||

𝑘2 = 2.469231

Плохо обусловленная матрица:

Cond(A) = 1.1077e+007;

||𝑥|/||𝑥||<= cond(A)∗||𝑏||/|𝑏||

Посмотрим когда достигается равенство, если число обусловленности велико:

Погрешность в b:

||𝑥|/||𝑥||=𝑘1∗||𝑏||/|𝑏||

𝑘1 = 266.354977

Погрешность в A.

||𝑥||/|𝑥+𝑥||=𝑘2∗||A|/||𝐴||

𝑘2 = 155.621327

## Вывод:

Коэффициенты всегда меньше числа обусловленности матрицы.

# Работа №3: Решение СЛАУ итерационными методами. Метод Зейделя.

## Формулировка задачи:

Методом Зейделя найти решение заданной системы:

Ax=b

Исследовать точность решения, когда определитель матрицы близок к 0

## Алгоритм метода и условия его применимости:

Модификация метода простых итераций.

Ax = b приводим к виду, пригодному для итераций

x(k) = Cx(k-1) + g

Достаточное условие сходимости метода:

||C|| < 1

Доказано, что метод всегда работает для матрицы с данными свойствами:

A – положительно-определенная

Ат = А

Но тогда выход из итерационного процесса не может осуществляться через С

x1(k) =c11x1(k-1)+c12x2(k-1)+…+b1

…

xn(k) =cn1x1(k-1)+cn2x2(k-1)+…+bn

После первой итерации x1(k) известно и так как предположили сходимость, то он точнее, чем х1(k-1)

Матричный вид:

x(k) = (C1+Dc)x(k) + (C2+Dc)x(k-1)+g

0 … 0

C1 = … …

cij … 0

0 … cij

C2 = … …

1. … 0

c11 … 0

Dc = … …

0 … cnn

||x||< e – условие выхода из цикла для положительно-определенной симметричной матрицы

## Проверка условий применимости метода:

Задаем матрицу А в MatLab так, чтобы она была симметричной и положительно определенной

Условия применимости: Достаточное - ||С|| < 1

Необходимое – (Ах,х) > 0 && Aт = А

## Тестовый пример:

10.000000 1.000000 1.000000

A = 2.000000 10.000000 1.000000

2.000000 2.000000 10.000000

12.000000

b = 13.000000

14.000000

0.000000 -0.100000 -0.100000

C = -0.200000 0.000000 -0.100000

-0.200000 -0.200000 0.000000

1.200000

g = 1.300000

1.400000

x(k) = Cx(k-1)+g:

1.000000

x = 1.000000

1.000000

## Модульная структура программы:

void CreateMemory(matrix\_data \*M)

Входные данные: данная матрица

Выделяет память для матрицы

void DestroyMemory(matrix\_data \*M)

Входные данные: данная матрица

Освобождает память для матрицы

void GetZeroMatrix(matrix\_data \*M)

Входные данные: данная матрица

Заполнят матрицу нулями

void GetMatrix(FILE\* f, matrix\_data \*Matr)

Входные данные: Файл с матрицей и матрица

Читает матрицу из файла и заполняет

double GetNorm(matrix\_data \*M)

Входные данные: данная матрица

Вычисляет норму матрицы

void CheckResult(matrix\_data \*x, matrix\_data \*Matr1, matrix\_data \*vect2)

Входные данные: вектор х, матрица, вектор результата

Перемножает вектор х и матрицу и сравнивает с вектором результата

void CreateMatrixC(matrix\_data \*C, matrix\_data \*A)

Входные данные: пустая матрица и данная матрица

Создаёт вспомогательную матрицу С

void CreateMatrixG(matrix\_data \*g, matrix\_data \*b, matrix\_data \*A)

Входные данные: пустой вектор, свободный вектор и данная матрица

Создаёт вспомогательный вектор g

int Check(matrix\_data \*x, matrix\_data \*p, matrix\_data \*C)

Входные данные: вектор результата, проверочный вектор и матрица С

Проверят условие выхода из цикла

matrix\_data\* Seidel(matrix\_data \*C, matrix\_data \*b, matrix\_data \*x)

Входные данные: матрица С, свободный вектор и вектор х

Находит значения вектора методом Зейделя

## Численный анализ решения задачи:

|  |  |  |  |
| --- | --- | --- | --- |
| det(A) | e | | |
| 10^-3 | 10^-5 | 10^-8 |
| 1.6256e^-5 | 13 | 20 | 43 |
| 2.7543e^-12 | 28 | 34 | 67 |
| 2,981e^-21 | 52 | 78 | 119 |
| 4,326e^-24 | 118 | 164 | 234 |

При е = 10^-3:

|  |  |
| --- | --- |
| det(A) | Норма вектора невязки |
| 1.6256e^-5 | 0,017392093 |
| 2.7543e^-12 | 0,0012837527 |
| 2,981e^-21 | 0,0006735415 |
| 4,326e^-24 | 0,000894251 |

## Вывод:

С уменьшением det(A) количество итераций увеличивается, а норма вектора невязки уменьшается

# Работа №4: Решение алгебраической проблемы собственных значений. Метод Якоби.

## Формулировка задачи:

Решить АПСЗ итерационным методом Якоби и исследовать сходимость при хорошей и плохой отделимости искомого собственного числа

## Алгоритм метода и условия его применимости:

AX = yX

Собственный вектор X

Условия применимости метода:

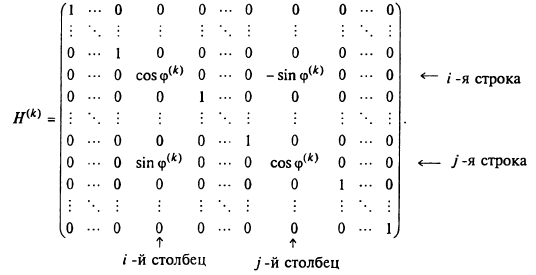
A – вещественная и симметричная

Алгоритм:

1. Выделяем в верхней треугольной матрице A(k) наддиагональную матрицу
2. Из наддиагональной матрицы выбираем максимальный по модулю элемент aij
3. Если выбранный элемент меньше эпсилон, то выход из итерационного процесса, в противном случае – находим угол поворота по формуле

φ (k)=0.5\*arctg(2a(k)ij / a(k)ii−a(k)jj)

1. Составляем матрицу поворота H



1. Вычисляем приближение и переходим к пункту 1.

A(k+1) = (H(k))TA(k)H(k)

## Проверка условий применимости метода:

Задаем матрицу А в MatLab так, чтобы она была симметричной

## Тестовый пример:

3.000000 2.000000

A(0)=

2.000000 1.000000

Наддиагональная:

0.000000 2.000000

К =

0.000000 0.000000

Максимальный элемент (он же единственный в данном примере):

2.000000

Вычислим угол поворота:

φ (0)=0.5\*arctg(2\*2/ 3−1)

φ (0)= 0.662909

Заполним матрицу поворота:

0.788205 -0.615412

H =

0.615412 0.788205

Транспонируем:

0.788205 0.615412

Hт =

-0.615412 0.788205

Перемножим: A(1) = Hт A(0)H

4.182821 -0.485071

A(1) =

-0.485071 -0.182821

Повторяем процесс (точность e = 0,01)

Результат:

Собственные числа:

4.236068

-0.236068

Собственные векторы:

0.850651

0.525731

-0.525731

0.850651

## Модульная структура программы:

void CreateMemory(matrix\_data \*M)

Входные данные: данная матрица

Выделяет память для матрицы

void DestroyMemory(matrix\_data \*M)

Входные данные: данная матрица

Освобождает память для матрицы

void GetZeroMatrix(matrix\_data \*M)

Входные данные: данная матрица

Заполнят матрицу нулями

void GetUnitMatrix(matrix\_data \*M)

Входные данные: данная матрица

Заполнят матрицу единицами

void GetMatrix(FILE\* f, matrix\_data \*Matr)

Входные данные: Файл с матрицей и матрица

void PrintMatrix(matrix\_data \*Matrix)

Входные данные: данная матрица

Печатает матрицу

void CopyMatrix(matrix\_data \*M, matrix\_data \*N, int size)

Входные данные: данная матрица, матрица, в которую нужно скопировать данные и размер матрицы

Копирует матрицу

matrix\_data\* MultiplyMatrix(matrix\_data \*A, matrix\_data \*B)

Входные данные: матрицы для перемножения

Перемножает матрицы

matrix\_data\* TransMatrix(matrix\_data\* H)

Входные данные: матрица для транспонирования

Транспонирует матрицу

double GetMaxEl(matrix\_data \*K, int \*i, int \*j)

Входные данные: матрица и указатели на индексы элементов

Находит максимальный элемент

void GetY(matrix\_data \*A)

Входные данные: Преобразованная после итераций матрица А

Собственные числа

void CreateMatrixK(matrix\_data \*K, matrix\_data \*A)

Входные данные: новая нулевая матрица К и данная матрица

Наддиагональная матрица К

void Jacob(matrix\_data \*A)

Входные данные: данная матрица

Вычисляет собственные числа и собственные вектора данной матрицы

## Численный анализ решения задачи:

Хорошая отделимость собственных чисел матрицы А:

Зависимость итераций

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cond(A) | e | | | Максимум нормы  A-yE  (при е = 10-2) |
| 10^-2 | 10^-5 | 10^-8 |
| 5.9660 | 64 | 97 | 118 | 0.000001328 |
| 16.5931 | 71 | 101 | 120 | 0.000005012 |
| 23,2604 | 76 | 102 | 126 | 0.0000023892 |
| 249.2341 | 91 | 121 | 139 | 0.00007123 |
| 4.3542e^12 | 102 | 156 | 189 | 0.00012382 |

Плохая отделимость собственных чисел матрицы А:

Зависимость итераций

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cond(A) | e | | | Максимум нормы  A-yE  (при е = 10-2) |
| 10^-2 | 10^-5 | 10^-8 |
| 9.5016 | 40 | 84 | 105 | 0.00004378 |
| 16.8977 | 40 | 89 | 110 | 0.00001542 |
| 34.9749 | 51 | 95 | 118 | 0.000003892 |
| 150.5932 | 53 | 99 | 122 | 0.000001542 |
| 4.3542e^12 | 68 | 118 | 158 | 0.00006754 |

## Вывод:

Для матриц с плохой отделимостью собственных чисел метод вращения Якоби находит собственные числа и собственные векторы быстрее, чем для матриц с хорошей отделимостью. Метод хорошо работает для чисел с любым числом обусловленности.