**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Вычислительная математика»**

Тема: Метод хорд

|  |  |  |
| --- | --- | --- |
| Студент гр. 3342 |  | Иванов Д. М. |
| Преподаватель |  | Лисс А. Р. |

Санкт-Петербург

2025

## Задание

В лабораторной работе №4 предлагается, используя программы -

функции HORDA и Round из файла methods.cpp (файл заголовков metods.h, директория LIBR1), найти корень уравнения f(x)=0 с заданной точностью Eps методом хорд, исследовать скорость сходимости и обусловленности метода.

Для данной работы, как и для лабораторной работы №3 задаются индивидуальные варианты нелинейных уравнений (см. подраздел 3.6).

Порядок выполнения лабораторной работы №4:

1) Графически или аналитически отделить корень уравнения f(x)=0 (т.е. найти отрезки [Left, Right], на которых функция f(x) удовлетворяет условиям применимости метода).

2) Составить подпрограмму - функцию вычисления функции f(x), предусмотрев округление значений функции с заданной точностью Delta с использованием программы Round.

3) Составить головную программу, вычисляющую корень уравнения f(x)=0 и содержащую обращение к подпрограмме f(x), HORDA, Round и индикацию результатов.

4) Провести вычисления по программе. Теоретически и экспериментально исследовать скорость сходимости и обусловленность метода.Функция для индивидуального варианта:

f(x) = x4 – 13\*x2 + 36 – 1/x

**Теоретическая часть**

Пусть найден отрезок [a, b], на котором функция f(x) меняет знак. Для определенности положим f (a)>0, f (b)<0. В методе хорд процесс итераций состоит в том, что в качестве приближений к корню уравнения f(x)=0 принимаются значения c0, c1, . . . точек пересечения хорды с осью абсцисс.

Сначала находится уравнение хорды АВ:

(y – f(a)) / (f(b) – f(a)) = (x - a) / (b - a)

Для точки пересечения ее с осью абсцисс (x=c0, y=0) получается уравнение

c0 = a – (b - a) / (f(b) – f(a)) \* f(a)

Далее сравниваются знаки величин f (a) и f (с0) и для рассматриваемого

случая оказывается, что корень находится в интервале (a, c0), так как f (a)f

(с0)<0. Отрезок [c0,b] отбрасывается. Следующая итерации состоит в определении нового приближения c1 как точки пересечения хорды АВ1 с осью абсцисс и т.д. Итерационный процесс продолжается до тех пор, пока значение

f (cn) не станет по модулю меньше заданного числа  (см. подраздел 3.1).

Алгоритмы методов бисекции и хорд похожи, однако метод хорд в ряде случаев дает более быструю сходимость итерационного процесса, причем успех его применения, как и метода бисекции, гарантирован.

**Аналитический анализ функции**

Для выполнения теоремы Коши нужны найти такие промежутки [a, b], в которых содержится корень уравнения f(x)=0. Преобразуем выражение.

f(x) = (x5 – 13\*x3 + 36\*x - 1) / x. Можем заметить, что корни числителя g(x) = x5 – 13\*x3 + 36\*x – 1 буду совпадать с корнями f(x) за исключением x=0, так как в этой точке функция прерывается из-за деления на 0. Также уравнение может иметь максимум 5 корней. Рассмотрим пределы функции при различных x.

Будем подставлять в f(x) различные значения и следить за тем, как изменяется знак f(x).

f(-4) = 84,25 > 0

f(-3) = 0,33 > 0

f(-2,5) = -5,7875 < 0

f(-2) = 0,5 > 0

f(-1) = 25 > 0

f(1) = 23 > 0

f(2) = -0,5 < 0

f(3) = -0,33 < 0

f(4) = 83,75 > 0

Можем уже увидеть некоторые интервалы, где знаки меняются: [-4, -2,5], [-2,5, -2], [1, 2], [3, 4].

Также рассмотрим предел функции при x->0+ (см. выше) и f(1). Можем заметить, что начиная с x=1 и идя к x=0 значение функции идет бесконечно вниз и, таким образом, пересечет прямую Ox в промежутке (0, 1]. Найдем число, которое близко к 0. f(0.01)= -64.0012, так что еще один промежутком у нас будет [0.01, 1].

В итоге мы получили нужные нам промежутки, в которых содержатся корни уравнения f(x)=0.

**Написание программы для реализации алгоритма хорд**

Реализуем алгоритм на языке C++. Для этого напишем некоторые функции.

double func(double x) – функция для вывола значения функции f(x) по переданному x.

double roundValue(double x, double delta) – функция для округления переданного числа на заданную точность. Сначала идет деление на эту точность, после этого округление до целого с помощью команды round и затем умножение на переданную точность.

double horda(double a, double b, double eps, int iterationsCount, double delta) – сам алгоритм хорд. Передаются границы отрезка [a, b], точность Eps и счетчик количества итераций (для анализа). Берется на каждой итерации значение c0 – пересечение хорды с осью абцисс. Если значение функции в этой точке меньше или равна eps, возвращается найденный корень. В противном случае продолжается нахождение персечения на новой границе, которую делит точка c0. И выбирается та часть отрезка, на концах которой значения функции имеет противоположные знаки. То есть идет вызов либо bisection(a, c0, eps, ++iterationsCount, delta), либо bisection(c0, b, eps, ++iterationsCount, delta).

Тестирование программы приведено в таблице 1.

Таблица 1 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | bisection(-4, -2.5, 0.001, iterations, 0.000001) | -2.988647143 | f(-2.988647143)=-0.0007193167928 |
|  | bisection(-2.5, -2, 0.1, iterations, 0.000001) | -2.025258857 | f(-2.025258857)= -0.004265658988 |
|  | bisection(0.01, 1, 0.01, iterations, 0.000001) | 0.02779310654 | f(0.02779310654)=0.00981825 |
|  | bisection(3, 4, 0.00001, iterations, 0.000001 ) | 3.010907432 | f(3.010907432)= -0.000009385556697 |

**Исследование зависимости числа итераций от Eps**

Для исследование данной зависимости напишем программу, которая будет перебирать различные eps и высчитывать количество итераций (количество вызовов функции), которое потребовалось данной программе. Будем перебирать, к примеру, для отрезка [-2,5, -2]. Значения Eps будем брать от 0.1 до 0.000001. Получим следующие результаты: (см. рис. 1).

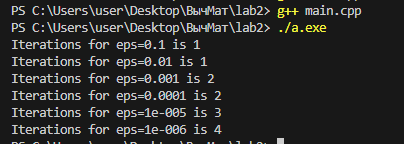


Рисунок 1 – результаты зависимости количества итераций от Eps

Построим график для данных полученных значений. Также сравним его теоретической зависимостью для метода бисекции N=log2((b - a) / eps). Получили следующую картину (см. рис. 2).

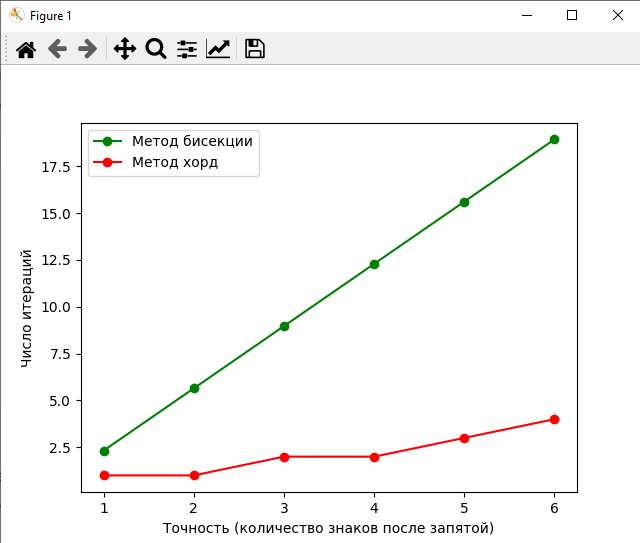


Рисунок 2 – графики зависимостей числа итераций от точности для алгоритмов бисекции и хорд

По данным графикам видно, что алгоритм хорд работает быстрее алгоритма бисекции на даонном отрезке. Однако эта связь зависит от поведения функции на данном промежутке промежутке и от длины выбранного отрезка.

**Исследование чувствительности метода к ошибкам в исходных данных**

Для выполнения данного исследование необходимо изменять точность выходного значения функции, округляя на некоторое количество знаков после запятой. Тем самым будут возникать ошибки в наших исходных данных, и мы проанализируем, насколько точно алгоритм будет работать.

Оценим абсолютное значение обусловленности нашей задачи по формуле:

Получаем:

Мы возьмем некоторый промежуток (к примеру [-4, -2,5]) и будем перебирать для него delta и eps. Будем сравнивать наши результаты с максимально точным вызовом для этого промежутка с eps=10^-6 и delta=10^-6. bisection(-4, -2.5, 0.000001, iterations, 0.000001) -> -2.988671968. νΔ(x)=0.0345259. Также оценим, насколько хорошо обусловлена задача при различных eps и delta. Если *νΔ* ≤ *νΔ\_*max, то можно говорить о том, что задача хорошо обусловлена в данном случае.

Результаты исследования см. в таблице 2.

Таблица 2 – Результаты перебора eps и delta для промежутка [-4, -2,5]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| delta | eps | Значение корня | Значение  *νΔ\_*max | Значение  νΔ | Значение  νΔ ≤ νΔ\_max |
| 0.1 | 0.1 | -2.987205733 | 1 | 0.03466842626897813 | 1 |
| 0.1 | 0.01 | -2.987205733 | 0.1 | 0.03466842626897813 | 1 |
| 0.1 | 0.001 | -2.987205733 | 0.01 | 0.03466842626897813 | 0 |
| 0.1 | 0.0001 | -2.987205733 | 0.001 | 0.03466842626897813 | 0 |
| 0.1 | 0.00001 | -2.987205733 | 0.0001 | 0.03466842626897813 | 0 |
| 0.1 | 0.000001 | -2.987205733 | 0.00001 | 0.03466842626897813 | 0 |
| 0.01 | 0.1 | -2.985915951 | 10 | 0.03479461917383552 | 1 |
| 0.01 | 0.01 | -2.988559781 | 1 | 0.03453677380622007 | 1 |
| 0.01 | 0.001 | -2.988559781 | 0.1 | 0.03453677380622007 | 1 |
| 0.01 | 0.0001 | -2.988559781 | 0.01 | 0.03453677380622007 | 0 |
| 0.01 | 0.00001 | -2.988559781 | 0.001 | 0.03453677380622007 | 0 |
| 0.01 | 0.000001 | -2.988559781 | 0.0001 | 0.03453677380622007 | 0 |
| 0.001 | 0.1 | -2.985954776 | 100 | 0.034790809211849544 | 1 |
| 0.001 | 0.01 | -2.988346393 | 10 | 0.03455746526588063 | 1 |
| 0.001 | 0.001 | -2.988658536 | 1 | 0.0345272049607086 | 1 |
| 0.001 | 0.0001 | -2.988658536 | 0.1 | 0.0345272049607086 | 1 |
| 0.001 | 0.00001 | -2.988658536 | 0.01 | 0.0345272049607086 | 0 |
| 0.001 | 0.000001 | -2.988658536 | 0.001 | 0.0345272049607086 | 0 |
| 0.0001 | 0.1 | -2.985951062 | 1000 | 0.034791173642421146 | 1 |
| 0.0001 | 0.01 | -2.988349887 | 100 | 0.03455712629710463 | 1 |
| 0.0001 | 0.001 | -2.988647624 | 10 | 0.034528262057074294 | 1 |
| 0.0001 | 0.0001 | -2.988670431 | 1 | 0.03452605269869278 | 1 |
| 0.0001 | 0.00001 | -2.988670431 | 0.1 | 0.03452605269869278 | 1 |
| 0.0001 | 0.000001 | -2.988670431 | 0.01 | 0.03452605269869278 | 0 |
| 0.00001 | 0.1 | -2.985951758 | 10000 | 0.03479110534799494 | 1 |
| 0.00001 | 0.01 | -2.98834974 | 1000 | 0.03455714055812796 | 1 |
| 0.00001 | 0.001 | -2.988647117 | 100 | 0.034528311173856734 | 1 |
| 0.00001 | 0.0001 | -2.988669084 | 10 | 0.03452618317857683 | 1 |
| 0.00001 | 0.00001 | -2.988671845 | 1 | 0.03452591572961225 | 1 |
| 0.00001 | 0.000001 | -2.988671845 | 0.1 | 0.03452591572961225 | 1 |
| 0.000001 | 0.1 | -2.985951724 | 100000 | 0.034791108684211675 | 1 |
| 0.000001 | 0.01 | -2.988349779 | 10000 | 0.03455713677459018 | 1 |
| 0.000001 | 0.001 | -2.988647143 | 1000 | 0.03452830865504452 | 1 |
| 0.000001 | 0.0001 | -2.988669039 | 100 | 0.03452618753760737 | 1 |
| 0.000001 | 0.00001 | -2.988671751 | 10 | 0.034525924835024785 | 1 |
| 0.000001 | 0.000001 | -2.988671968 | 1 | 0.03452590381508913 | 1 |

По данным результатам можем заметить следующее. Для маленьких delta задача имеет хорошую обусловленность при любых eps: νΔ ≤ νΔ\_max. Это неравенство не выполняется только в тех случаях, когда eps значительно меньше delta.

Также можем заметить, что при самых неточных округлениях значение отличается от точного начиная с 3-его разряда после запятой. При увеличении точности корень будет постепенно приближаться к нужному нам.

В итоге можно сказать, что алгоритм хорошо устойчив к ошибкам в исходных данных и показывает хорошую обусловленность. При небольших изменениях выходное значение будет максимально приближено к нашему искомому.

## Исследование скорости сходимости

В ходе лабораторной работы была написана программа на языке C++, выполняющая

## Вывод

В ходе лабораторной работы была написана программа на языке C++, выполняющая поиск корней некоторой функции f(x) методом хорд. Данный алгоритм был протестирован на некоторых входных данных. Также провелись некоторые исследования. По их результатам мы получили зависимость количества итераций от точности корня и сравнили ее с зависимостью для алгоритма бисекции. По результатам мы получили, что метод хорд на некоторых промежутках может быть эффективнее метода бисекции. Однако такое может быть не всегда. Если функция имеет крутые изгибы или асимптотическое поведение вблизи корня или неудачно выбран отрезок, метод хорд может сходиться медленнее.

Также было проведено исследование на чувтсвительность метода к ошибкам. Мы убедились, что алгоритм хорд достаточно устойчив к небольшим ошибкам в исходных данных, но при значительном увеличении Delta точность результата может снизиться на небольшое значение.