**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра математического обеспечения и применения ЭВМ**

отчет

**по практической работе №3**

**по дисциплине «Вычислительная математика»**

Тема: **Метод бисекции**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8383 |  | Ларин А. |
| Преподаватель |  | Сучков А.И. |

Санкт-Петербург

2019

**Цель работы.**

Формирование практических навыков нахождения корней алгебраических и трансцендентных уравнений методом бисекции.

**Основные теоретические положения.**

Задача нахождения корней нелинейных уравнений вида , где – некоторая непрерывная функция, встречается в различных областях инженерной и научной деятельности. Нелинейные уравнения делятся на два класса – алгебраические и трансцендентные. Алгебраическими называются уравнения, содержащие только алгебраические функции (целые, рациональные, иррациональные). Уравнения, которые содержат другие функции (тригонометрические, показательные, логарифмические и т.п.), называются трансцендентными. Методы решения нелинейных уравнений подразделяются на прямые и итерационные. Прямые методы позволяют записать корни в виде некоторого конечного соотношения. Такие методы для решения ряда трансцендентных, а также простейших алгебраических уравнений известны из школьного курса алгебры. Однако встречающиеся на практике уравнения не удается решить столь простыми методами. Для их решения применяются итерационные методы, при которых алгоритм нахождения корня уравнения в общем случае включает два этапа:

* отыскания приближенного значения корня или содержащего его отрезка;
* уточнения приближенного значения до некоторой заданной степени точности.

Пусть задана непрерывная функция вещественного аргумента x и требуется численным методом решить уравнение , т.е. найти приближение к вещественному корню этого уравнения. Если уравнение имеет несколько вещественных корней, то сначала производят их отделение (изоляцию), а затем уточняют положение отдельного корня. Считается, что отделение корня произведено, если выделен такой интервал области определения функции , на концах которого значения функции и имеют разные знаки и внутри которого имеется ровно один корень уравнения . Для уточнения метода используют итерационные методы, такие как метод бисекции (половинного деления), метод хорд (секущих или ложного положения), метод Ньютона (касательных), метод итераций (последовательных приближений). В указанных методах вычисляются либо последовательность значений границ сужающихся интервалов содержащих корень, либо последовательность приближений к корню В первом случае итерационный процесс заканчивается, как только длина текущего интервала становится достаточно малой (например, ). Во втором случае условием остановки вычислений является малость очередного приращения . В обоих случаях параметр определяет момент остановки вычислений. Иногда в качестве условия остановки используют условие , где – текущее приближение к корню, например, в методе бисекции. Выполнение этого условия свидетельствует о малости значения функции в точке , т.е. позволяет считать, что . Для каждого итерационного метода можно указать некоторые условия сходимости. Однако не всегда легко проверить или гарантировать выполнение этих условий. Кроме того необходимо учесть особенности машинных вычислений при реализации итерационных методов. На практике эти затруднения обходят, вводя ограничение на число итераций. Такое ограничение предохраняет от «зацикливания метода», а также позволяет выявить практическое отсутствие сходимости вычислительного процесса.

В чем состоит метод бисекции. Если найден отрезок , такой, что , существует точка , в которой значение функции равно нулю, т.е. , . Метод бисекции состоит в построении последовательности вложенных друг в друга отрезков , на концах которых функция имеет разные знаки. Каждый последующий отрезок получается делением пополам предыдущего. Процесс построения последовательности отрезков найти нуль функции (корень уравнения ) с любой заданной точностью.

Рассмотрим один шаг итерационного процесса. Пусть на-м шаге найден отрезок , такой, что . Разделим его пополам точкой и вычислим . Если , то – корень уравнения. Если , то из двух половин отрезка выбирается та, на концах которой функция имеет противоположные знаки, поскольку искомый корень лежит на этой половине, т.е. , если , если , Если требуется найти корень с точностью , то деление пополам продолжается до тех пор, пока длина отрезка не станет меньше . Тогда координата середины отрезка есть значение корня с требуемой точностью . Метод бисекции является простым и надежным методом поиска простого корня уравнения (простым называется корень дифференцируемой функции , если и . Этот метод сходится для любых непрерывных функций , в том числе недифференцируемых. Скорость его сходимости невысока. Для достижения точности необходимо совершить итераций. Это означает, что для получения каждых трех верных десятичных знаков необходимо совершить около 10 итераций.

**Постановка задачи.**

Используя программы-функции BISECT и Round из файла methods.cpp (файл заголовков metods.h), найти корень уравнения методом бисекции с заданной точностью Eps, исследовать зависимость числа итераций от точности Eps при изменении Eps от 0.1 до 0.000001, исследовать обусловленность метода (чувствительность к ошибкам в исходных данных). Порядок выполнения работы следующий:

* Графически или аналитически отделить корень уравнения , т.е. найти отрезки , на которых функция удовлетворяет условиям теоремы Больцано-Коши.
* Составить подпрограмму вычисления функции f(x)
* Составить головную программу, содержащую обращение к подпрограмме F, BISECT, Round и индикацию результатов.
* Провести вычисления по программе. Построить график зависимости числа итераций от Eps, сопоставить его с графиком по формуле выше.
* Исследовать чувствительность метода к ошибкам в исходных данных. Ошибки в исходных данных моделировать с использованием программы Round, округляющей значения функции с заданной точностью Delta.

**Выполнение работы.**

Проанализируем функцию :

Отделим графическим методом корни уравнения , т.е. найдем отрезки [Left, Right], на которых функция удовлетворяет условиям применимости метода бисекции. По графику на рис. 1 видно что корень принадлежит отрезку и функция на его концах принимает разные знаки.

Оценим абсолютное значение обусловленности нашей задачи по формуле:

Получаем:

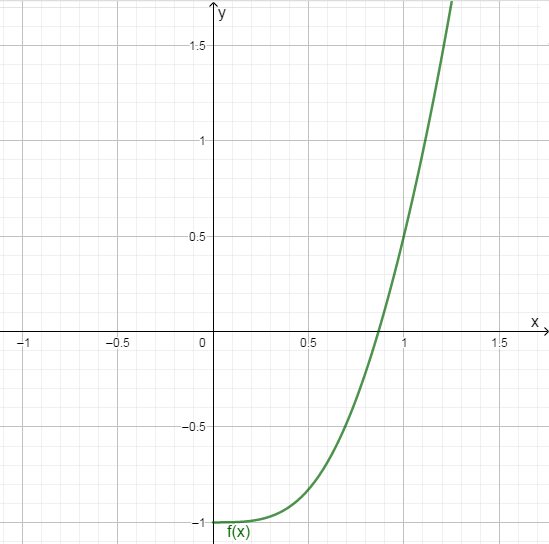


Рисунок 1 – Локализация корня функции

Проведем вычисление корня функции при помощи программы, приведенной в приложении А. Программа вычисляет корень уравнения методом бисекций. На вход ей подаются следующие параметры: eps – требуемая точность вычисления корня, delta – погрешность вычисления значений функции, a,b – отрезок , локализующий корень. В таблице 1 приведены расчеты корня при различных значениях eps, и сопоставлены значения количества итераций для с ходимости с теоретическими значениями.

Таблица 1 – расчет корня методом бисекции с варьированием значения eps

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Значение eps | Значение delta | Значение a | Значение b | Значение | Значение | Значение |
| 0.1 | 0.0001 | 0.5 | 1 | 0.875 | 2 | 2 |
| 0.01 | 0.0001 | 0.5 | 1 | 0.859375 | 5 | 6 |
| 0.001 | 0.0001 | 0.5 | 1 | 0.865234 | 8 | 9 |
| 0.0001 | 0.0001 | 0.5 | 1 | 0.867065 | 12 | 12 |
| 0.00001 | 0.0001 | 0.5 | 1 | 0.867096 | 13 | 16 |

Теперь, имея приближение корня, рассчитаем и оценим абсолютное значение обусловленности задачи. Примем . Тогда . Отсюда получаем

Теперь можно оценить хорошо ли обусловлена задача при различных значениях и . Если значение *νΔ* ≤ *νΔ\_*max  - хор., то будем считать, что задача хорошо обусловлена, иначе пл. – плохо. Результаты эксперимента занесены в таблицу 2.

Таблица 2 – Обусловленность задачи при различных и

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Значение eps | Значение delta | Значение | Значение  *νΔ\_*max | Значение  *νΔ* | Значение  *νΔ* ≤ *νΔ\_*max |
| 0.1 | 0.01 | 0.875000 | 10 | 0.291945 | хор. |
| 0.01 | 0.01 | 0.859375 | 1 | 0.299954 | хор. |
| 0.001 | 0.01 | 0.867188 | 0.1 | 0.295887 | пл. |
| 0.0001 | 0.01 | 0.867188 | 0.01 | 0.295887 | пл. |
| 0.00001 | 0.01 | 0.867188 | 0.01 | 0.295887 | пл. |
| 0.000001 | 0.01 | 0.867188 | 0.001 | 0.295887 | пл. |
| 0.1 | 0.001 | 0.875000 | 100 | 0.291945 | хор. |
| 0.01 | 0.001 | 0.859375 | 10 | 0.299954 | хор. |
| 0.001 | 0.001 | 0.867188 | 1 | 0.295887 | хор. |
| 0.0001 | 0.001 | 0.867188 | 0.1 | 0.295887 | пл. |
| 0.00001 | 0.001 | 0.867188 | 0.01 | 0.295887 | пл. |
| 0.000001 | 0.001 | 0.867188 | 0.001 | 0.295887 | пл. |
| 0.01 | 0.0001 | 0.859375 | 100 | 0.299954 | хор. |
| 0.001 | 0.0001 | 0.865234 | 10 | 0.296892 | хор. |
| 0.0001 | 0.0001 | 0.867065 | 1 | 0.295950 | хор. |
| 0.00001 | 0.0001 | 0.867096 | 0.1 | 0.295934 | пл. |
| 0.00001 | 0.0001 | 0.867096 | 0.01 | 0.295934 | пл. |

**Выводы.**

Проанализировав данные вычислительного эксперимента можно сделать вывод, что обусловленность задачи нахождения корня уравнения

имеет довольно низкое значение обусловленности, потому хорошо обусловлена почти при любых параметрах, кроме случая, когда значение Eps значительно ниже значения Delta. Проанализировав результаты применения метода бисекции, можно сказать, что при данном значении обусловленности он дает приемлемые результаты, и сходится за небольшое число итераций, которое в общем случае меньше или равно теоретического значения, и похоже на него тем больше, чем меньше значение delta.

Приложение А

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

#include <stdio.h>

#include <math.h>

#include <stdlib.h>

#include <iostream>

#include <conio.h>

double delta;

extern double F(double);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция F(X) , задаваемая пользователем \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifdef \_\_NEWTON

extern double F1(double);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Производная функции F(X) , задаваемая пользователем \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#endif

#ifdef \_\_ITER

extern double PHI(double);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция PHI(X) , задаваемая пользователем \*/

/\* Данная функция используется в методе \*/

/\* простых итераций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#endif

double Round(double, double);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция Round (X, Delta) , предназначена для округления \*/

/\* X с точностью Delta \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double BISECT(double, double, double, int&);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция BISECT предназначена для решения уравнения F(X)=0 \*/

/\* методом деления отрезка пополам. Использованы обозначения: \*/

/\* Left - левый конец промежутка \*/

/\* Right - правый конец промежутка \*/

/\* Eps - погрешность вычисления корня уравнения; \*/

/\* N - число итераций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double ITER(double, double, int&);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция ITER предназначена для решения уравнения F(X)=X \*/

/\* методом простой итерации. Использованы обозначения: \*/

/\* X0 - начальное приближение корня \*/

/\* Eps - погрешность вычисления корня уравнения; \*/

/\* N - число итераций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double HORDA(double, double, double, int&);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция HORDA предназначена для решения уравнения F(x)=0 \*/

/\* методом хорд. Использованы обозначения: \*/

/\* Left - левый конец промежутка \*/

/\* Right - правый конец промежутка \*/

/\* Eps - погрешность вычисления корня уравнения; \*/

/\* N - число итераций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double NEWTON(double, double, int&);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Функция NEWTON предназначена для решения уравнения F(X)=0 \*/

/\* методом касательных. Использованы обозначения: \*/

/\* X - начальное приближение корня \*/

/\* Eps - погрешность вычисления корня уравнения; \*/

/\* N - число итераций \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

double Round(double X, double Delta) {

if (Delta <= 1E-9) {

puts("Неверное задание точности округления\n");

exit(1);

}

if (X > 0.0) {

return Delta \* long(X / Delta + 0.5);

} else {

return Delta \* long(X / Delta - 0.5);

}

}

double F(double x) {

// функция f(x)

extern double delta;

double s;

long S;

s = pow(x, M\_PI) - 1/(pow(x,4)+1);

s = Round(s, delta);

return s;

}

double F1(double x) {

// функция f'(x)

//double f = (M\_PI \* pow(x, M\_PI + 7) + 2 \* M\_PI\*pow(x, M\_PI + 3) + M\_PI \* pow(x, M\_PI - 1) + 4 \* pow(x, 3)) / (pow(x, 8) + 2 \* pow(x, 4) + 1);

return x;

}

double PHI(double x) {

// функция φ(x) - для метода простых итераций

return x;

}

double BISECT(double Left, double Right, double Eps, int &N) {

double E = fabs(Eps) \* 2.0;

double FLeft = F(Left);

double FRight = F(Right);

double X = 0.5 \* (Left + Right);

double Y;

if (FLeft \* FRight > 0.0) {

puts("Неверное задание интервала\n");

exit(1);

}

if (Eps <= 0.0) {

puts("Неверное задание точности\n");

exit(1);

}

if (FLeft == 0.0) {

return Left;

}

if (FRight == 0.0) {

return Right;

}

for (N = 0; Right - Left >= E; N++) {

X = 0.5 \* (Right + Left); // вычисление середины отрезка

Y = F(X);

if (Y == 0.0) {

return X;

}

if (Y \* FLeft < 0.0) {

Right = X;

} else {

Left = X;

FLeft = Y;

}

}

return X;

}

#ifdef \_\_ITER

double ITER(double X0, double Eps, int &N) {

extern double PHI(double);

if (Eps <= 0.0) {

puts("Неверное задание точности\n");

exit(1);

}

double X1 = PHI(X0);

double X2 = PHI(X1);

for (N = 2;

(X1 - X2) \* (X1 - X2) > fabs((2 \* X1 - X0 - X2) \* Eps);

N++) {

X0 = X1;

X1 = X2;

X2 = PHI(X1);

}

return X2;

}

#endif

#ifdef \_\_NEWTON

double NEWTON(double X, double Eps, int &N) {

extern double F1(double);

double Y, Y1, DX, Eps0;

N = 0;

double m1 = 0.0, // наименьшее значение модуля 1-ой производной

M2 = 0.0; // наибольшее значение модуля 2-ой производной

Eps0 = sqrt(2 \* m1 \* Eps / M2);

do {

Y = F(X);

if (Y == 0.0) {

return X;

}

Y1 = F1(X);

if (Y1 == 0.0) {

puts("Производная обратилась в ноль\n");

exit(1);

}

DX = Y / Y1;

X -= DX;

N++;

} while (fabs(DX) > Eps0);

return X;

}

#endif

double HORDA(double Left, double Right, double Eps, int &N) {

double FLeft = F(Left);

double FRight = F(Right);

double X, Y;

if (FLeft \* FRight > 0.0) {

puts("Неверное задание интервала\n");

exit(1);

}

if (Eps <= 0.0) {

puts("Неверное задание точности\n");

exit(1);

}

N = 0;

if (FLeft == 0.0) {

return Left;

}

if (FRight == 0.0) {

return Right;

}

do {

X = Left - (Right - Left) \* FLeft / (FRight - FLeft);

Y = F(X);

if (Y == 0.0) {

return X;

}

if (Y \* FLeft < 0.0) {

Right = X;

FRight = Y;

} else {

Left = X;

FLeft = Y;

}

N++;

} while (fabs(Y) >= Eps);

return X;

}

#ifndef M\_PI

#define M\_PI 3.14159265358979323846

#endif // !M\_PI

#define FF(x) ( (M\_PI \* pow(x, M\_PI + 7) + 2 \* M\_PI\*pow(x, M\_PI + 3) + M\_PI \* pow(x, M\_PI - 1) + 4 \* pow(x, 3)) / (pow(x, 8) + 2 \* pow(x, 4) + 1) )

int main()

{

int k;

long int s;

float a1, b1, eps1, delta1;

double a, b, eps, x;

a = 0.5;

b = 1;

printf("eps\t\tdelta\t\ta\t\tb\t\tx\t\tn\_m\t\tn\t\t\tk\tk\_t\tC\n");

for (delta = 0.1; delta >= 0.000001; delta /= 10)

{

for (eps = 0.1; eps >= 0.000001; eps /= 10)

{

x = BISECT(a, b, eps, k);

printf("%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%d\t%d\t%d\n", eps, delta, a, b, x, eps / delta, 1 / (double)FF(x), k, (int)round(log2((b - a) / eps)), eps / delta> 1 / FF(x));

}

}

return 0;

}