МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по практической работе №4

по дисциплине «Вычислительная математика»

Тема: Метод хорд

Студент гр. 8383 _____ Ларин А.

Преподаватель _____ Сучков А.И.

Санкт-Петербург

2019

Цель работы.

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

Основные теоретические положения.

Пусть найден отрезок $[a, b]$, на котором функция 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 c_0, c_1, ... точек пересечения хорды с осью абсцисс, как это показано на рис. 1.

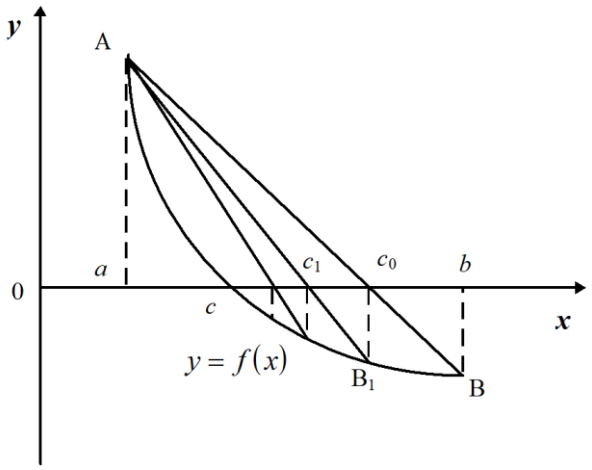

Рисунок 1 – Графическая демонстрация метода хорд

Сначала находится уравнение хорды AB:

$$\frac{y - f(a)}{f(b) - f(a)} = \frac{x - a}{b - a}$$

Для точки пересечения ее с осью абсцисс ($x = c_0, y = 0$) получается уравнение

$$c_0 = a - \frac{b - a}{f(b) - f(a)} f(a)$$

Далее сравниваются знаки величин $f(a)$ и $f(c_0)$ и для рассматриваемого случая оказывается, что корень находится в интервале (a, c_0), так как $f(a)f(c_0) < 0$. Отрезок $[c_0, b]$ отбрасывается. Следующая итерации состоит в определении нового приближения c_1 как точки пересечения хорды AB_1 с осью абсцисс и т.д. Итерационный процесс продолжается до тех пор, пока значение $f(c_n)$ не станет

по модулю меньше заданного числа ε. 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.

Постановка задачи.

Используя программы-функции HORDA и Round из файла methods.cpp (файл заголовков metods.h), найти корень уравнения $f(x) = 0$ заданной точностью Eps методом хорд, исследовать скорость сходимости и обусловленность метода. Порядок выполнения работы следующий:

1. Графически или аналитически отделить корень уравнения $f(x) = 0$, т.е. найти отрезки $[a, b]$, 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, HORDA, Round и индикацию результатов.
4. Провести вычисления по программе. Теоретически и экспериментально исследовать скорость сходимости и обусловленность метода.

Выполнение работы.

Проанализируем функцию $f(x)$:

$$f(x) = x^{\pi} - \frac{1}{1 + x^4}$$

Отделим графическим методом корни уравнения, т.е. найдем отрезки [Left, Right], на которых функция удовлетворяет условиям теоремы Больцано-Коши. По графику на рис.2 видно что корень принадлежит отрезку $[0.5, 1]$ и функция на его концах принимает разные знаки.

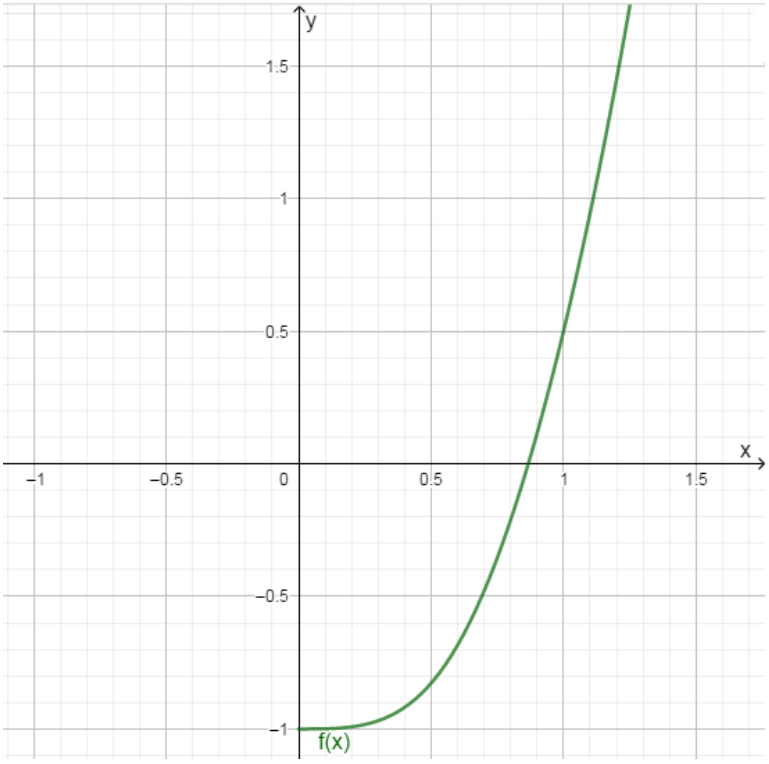

Рисунок 2 – Локализация корня функции $f(x)$

Проведем вычисление корня функции $f(x)$ при помощи программы, приведенной в приложении А. Программа вычисляет корень уравнения методом хорд. На вход ей подаются следующие параметры: `eps` – требуемая точность вычисления корня, `delta` – погрешность вычисления значений функции, `a`, `b` – отрезок $[a, b]$, локализующий корень. В табл. 1 приведены расчеты корня $\overline{x}$ при различных значениях `eps,` и представлены значения количества итераций.

Таблица 1 – Расчет корня x методом хорд с варьированием значения $\boldsymbol{eps}$

Значение eps	Значение delta	Значение a	Значение b	Значение $\boldsymbol{x}$	Значение $\boldsymbol{k}$
0.1	0.0001	0.5	1	0.86116	2
0.01	0.0001	0.5	1	0.866474	3
0.001	0.0001	0.5	1	0.867033	4
0.0001	0.0001	0.5	1	0.867086	4
0.00001	0.0001	0.5	1	0.867086	4

Теперь, имея приближение корня и зная экспериментальное значение скорость сходимости метода, сравним его с методом бисекции. Из проведенного ранее исследования известно, что порядок сходимости метода бисекции линеен. Из значения количества итераций видно, что скорость сходимости метода хорд выше линейной. Действительно, согласно теории порядок сходимости метода равна золотому сечению $\alpha = \frac{1+\sqrt{5}}{2} \approx 1.618$, то есть скорость сходимости метода хорд выше скорости сходимости метода бисекции.

В табл.2 приведено сравнение методов хорд и бисекции для расчета корня. x_H - корень, высчитанный по методу хорд. x_B - корень, высчитанный по методу, бисекций. k_H – количество итераций, затраченное на приближение корня методом хорд. k_B – количество итераций, затраченное на приближение корня методом хорд.

Выводы.

Проанализировав результаты применения метода хорд, можно сказать, что при расчете данной функции он дает хорошие результаты, и сходится за малое число итераций, которое соответствует теоретическому значению порядка

Проанализировав данные вычислительного эксперимента по двум методам: хорд и бисекций, можно сделать вывод, что для данной функции метод хорд сходится быстрее и имеет преимущество перед методом бисекции

Таблица 2 – Сравнение методов хорд и бисекции

Значение eps	Значение delta	Значение x_B	Значение x_H	Значение k_B	Значение k_H
0.01	0.001	0.859375	0,86651	5	3
0.001	0.001	0.867188	0,867042	5	3
0.0001	0.001	0.867188	0,867042	5	3
0.00001	0.001	0.867188	0,867042	5	3
0.000001	0.001	0.867188	0,867042	5	3
0.1	0.0001	0.875	0.86166	2	2
0.01	0.0001	0.859375	0.866474	5	3
0.001	0.0001	0.865234	0.867033	8	4
0.0001	0.0001	0.867065	0.867086	12	4
0.00001	0.0001	0.867096	0.867086	13	4
0.000001	0.0001	0.867096	0.867086	13	4
0.1	0.00001	0.875	0.861154	2	2
0.01	0.00001	0.859375	0.86478	5	3
0.001	0.00001	0.865234	0.867024	8	4
0.0001	0.00001	0.867065	0.86708	12	5
0.00001	0.0001	0.8667081	0.867085	15	5
0.000001	0.0001	0.867085	0.867085	16	5

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <iostream>
#include <conio.h>

double delta;

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif // !M_PI

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;
}
```

```
#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_B,k_H;
	long int s;
	float a1, b1, eps1, delta1;
	double a, b, eps, x_B,x_H;

	a = 0.5;
	b = 1;
	printf("eps\t\tdelta\t\ta\t\tb\t\tx_B\t\tx_H\t\tn_m\t\tn\t\t\tk_B\t
k_H\tk_t\tC\n");
	for (delta = 0.1; delta >= 0.000001; delta /= 10)
	{
		for (eps = 0.1; eps >= 0.000001; eps /= 10)
		{
			x_B = BISECT(a, b, eps, k_B);
			x_H = HORDA(a, b, eps, k_H);

	printf("%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%d\t%d\t%d\t%d\n",
eps, delta, a, b, x_B, x_H, eps / delta, 1 / (double)FF(x_H), k_B, k_H,
(int)round(log2((b - a) / eps)), eps / delta> 1 / FF(x_H));
		}
	}
	return 0;
}
```