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

ОТЧЕТ

по практической работе №5

по дисциплине «Вычислительная математика»

Тема: Метод Ньютона

Студент гр. 8383 _____ Ларин А.

Преподаватель _____ Сучков А.И.

Санкт-Петербург

2019

Цель работы.

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

Основные теоретические положения.

В случае, когда известно хорошее начальное приближение решения уравнения $f(x) = 0$, эффективным методом повышения точности является метод Ньютона (касательных). Он состоит в построении итерационной последовательности $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$, сходящейся к корню уравнения $f(x) = 0$. Достаточные условия сходимости метода формулируются теоремой.

Теорема.

Пусть $f(x)$ определена и дважды дифференцируема на $[a, b]$ причём $f(a)f(b) < 0$, а производные $f'(x), f''(x)$ сохраняют знак на отрезке $[a, b]$. Тогда, исходя из начального приближения $x_0 \in [a, b]$, удовлетворяющего неравенству $f(x_0)f''(x_0) > 0$, можно построить последовательность

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, n = 0,1,2, \ldots,$$

сходящуюся к единственному на $[a, b]$ решению ξ уравнения $f(x) = 0$.

Метод Ньютона допускает простую геометрическую интерпретацию, представленную на рис. 1. Если через точку с координатами $(x_n; f(x_n))$ провести касательную, то абсцисса точки пересечения этой касательной с осью OX будет очередным приближением x_{n+1} корня уравнения $f(x) = 0$.

Для оценки погрешности n-го приближения корня предлагается пользоваться неравенством:

$$|\xi - x_n| \leqslant \frac{M_2}{2m_1}|x_n - x_{n-1}|^2,$$

где M_2 – наибольшее значение модуля второй производной $|f''(x)|$ на отрезке $[a, b]$; m_1 – наименьшее значение модуля первой производной $|f'(x)|$ на отрезке $[a, b]$.

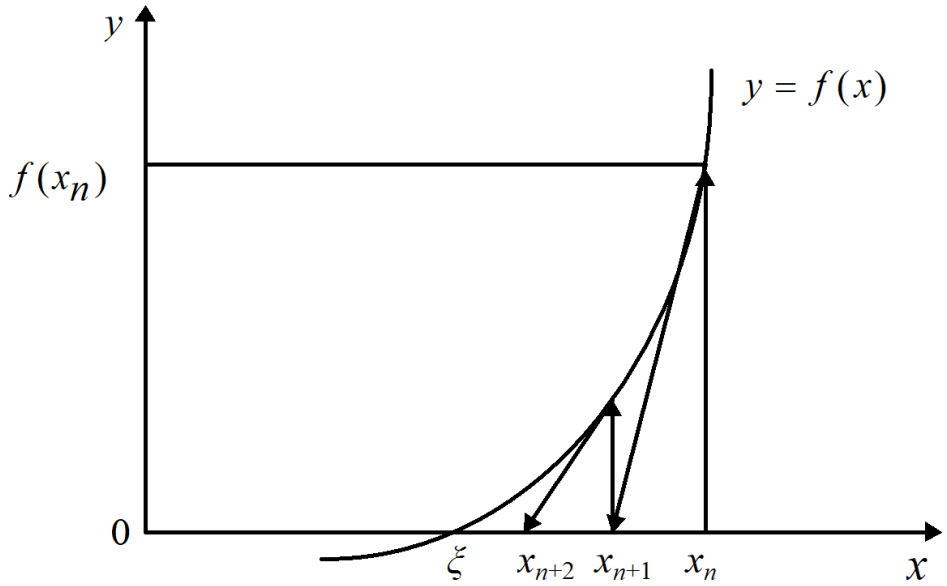

Рисунок 1 − Геометрическая интерпретация метода Ньютона

Таким образом, если

$$|x_n - x_{n-1}| < \varepsilon,$$

то

$$|\xi - x_n| \leqslant \frac{M_2 \varepsilon^2}{2m_1}.$$

Это означает, что при хорошем начальном приближении корня после каждой итерации число верных десятичных знаков в очередном приближении удваивается, т.е. процесс сходится очень быстро (имеет место квадратическая сходимость). Из указанного следует, что при необходимости нахождения корня с точностью ε итерационный процесс можно прекращать, когда

$$|x_n - x_{n-1}| < \varepsilon_0 = \sqrt{\frac{2m_1 \varepsilon}{M_2}}.$$

Рассмотрим один шаг итераций. Если на $(n-1)$-м шаге очередное приближение x_{n-1} не удовлетворяет условию окончания процесса, то вычисляются величины $f(x_{n-1}), f'(x_{n-1})$ и следующие приближение корня $x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$. При выполнении условия остановки, описанного выше, величина x_n принимается за приближенное значение корня ξ, вычисленное с точностью ε.

Постановка задачи.

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

1. Графически или аналитически отделить корень уравнения $f(x) = 0$. Убедиться, что на найденном отрезке $[a, b]$ функция $f(x)$ удовлетворяет условиям сходимости метода Ньютона.
2. Выбрать начальное приближение корня $x_0 \in [a, b]$ так, чтобы $f(x_0)f''(x_0) > 0$.
3. Оценить снизу величину $m_1 = \min\limits_{x\in[a,b]} |f'(x)|$, оценить сверху величину $M_2 = \max\limits_{x\in[a,b]} |f''(x)|$.
4. По заданному ε выбрать значение ε_0 для условия окончания итерационного процесса $\varepsilon_0 = \sqrt{\frac{2m_1\varepsilon}{M_2}}$.
5. Составить подпрограммы-функции вычисления $f(x)$, $f'(x)$, предусмотрев округление их значений с заданной точностью $Delta$.
6. Составить головную программу, вычисляющую корень уравнения $f(x) = 0$ и содержащую обращение к подпрограммам F, F1, Round, NEWTON и индикацию результатов.
7. Провести вычисления по программе. Исследовать скорость сходимости метода и его чувствительность к ошибкам в исходных данных, сравнить скорости сходимости методов Ньютона, бисекции и хорд.

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. 1 видно что корень принадлежит отрезку $[0.5, 1.25]$ и функция на его концах принимает разные знаки.

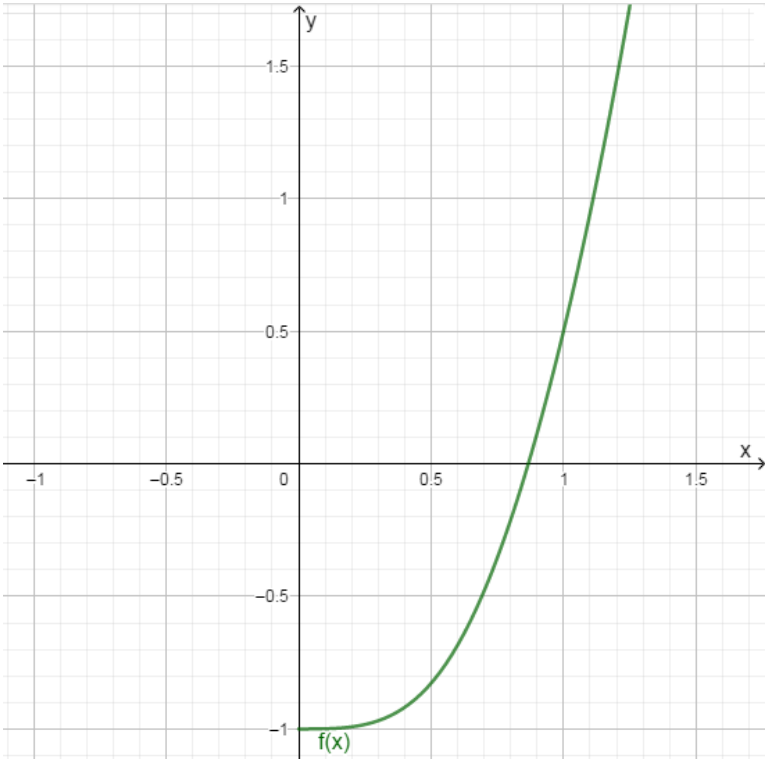

Рисунок 1 – Локализация корня функции $f(x)$

Проверим, что на выбранном отрезке функция удовлетворяет условиям сходимости метода Ньютона:

- Функция дважды дифференцируема на $[0.5, 1.25]$
- $f(0.5)f(1.25) < 0$
- Производные $f'(x), f''(x)$ сохраняют знак

Возьмем начальное приближение $x_0 = b = 1.25$.

$f(x_0) > 0, f''(x_0) > 0 => f(x_0)f''(x_0) > 0$. Данное приближение соответствует условию теоремы.

Оценим величины m_1, M_2:

$$m_1 = \min_{x\in[a,b]} |f'(x)| = 1.154884,$$

$$M_2 = \max_{x\in[a,b]} |f''(x)| = 7.268115.$$

Тогда $\varepsilon_0 = \sqrt{\frac{2m_1\varepsilon}{M_2}} = 0.178268.$

Проведем вычисление корня функций $f(x), f'(x)$ при помощи программы, приведенной в приложении А. Программа вычисляет корень уравнения методом Ньютона. На вход ей подаются следующие параметры: `X` – начальное приближение корня, `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 Ньютона с варьированием значения *eps*

Значение eps	Значение $delta$	Значение a	Значение b	Значение x	Значение k
0.1	0.00001	0.5	1	0.872045	2
0.01	0.00001	0.5	1	0.867107	3
0.001	0.00001	0.5	1	0.867107	3
0.0001	0.00001	0.5	1	0.867107	3
0.00001	0.00001	0.5	1	0.867086	4

Имея экспериментальное значение скорость сходимости метода видим, что скорость сходимости метода хорд выше линейной. Действительно, согласно теории порядок сходимости метода равен 2, т.е. квадратичный.

Теперь, имея приближение корня, примем $\overline{x} \approx 0.867086$. С помощью данного приближения вычислим $\Delta x = |\overline{x} - x|$, и оценим с его помощью чувствительность метода к ошибкам в исходных данных. При $\Delta x \leq eps$ будем считать, что задача хорошо обусловлена – хор., иначе пл. – плохо. Результаты эксперимента занесены в табл. 2. Теперь, имея значение количества итераций, необходимое для локализации корня, сравним метод Ньютона с методом бисекции и методом хорд. Из проведенного ранее исследования известно, что порядок сходимости метода бисекции и метода хорд линейны

Таблица 2 – Обусловленность задачи при различных eps и $delta$

Значение eps	Значение $delta$	Значение x	Значение k	Значение Δx	Значение $eps \geq \Delta x$
0.1	0.01	0.87243	2	0.005344	хор.
0.01	0.01	0.866566	3	0.00052	хор.
0.001	0.01	0.866566	3	0.00052	хор.
0.0001	0.01	0.866566	3	0.00052	пл.
0.00001	0.01	0.866566	3	0.00052	пл.
0.000001	0.01	0.866566	3	0.00052	пл.
0.1	0.001	0.872102	2	0.005016	хор.
0.01	0.001	0.867114	3	0.000028	хор.
0.001	0.001	0.867114	3	0.000028	хор.
0.0001	0.001	0.867114	3	0.000028	хор.
0.00001	0.001	0.867114	3	0.000028	пл.
0.000001	0.001	0.867114	3	0.000028	пл.
0.1	0.0001	0.872037	2	0.004951	хор.
0.01	0.0001	0.867108	3	0.000022	хор.
0.001	0.0001	0.867108	3	0.000022	хор.
0.0001	0.0001	0.867108	3	0.000022	хор.
0.00001	0.0001	0.867078	4	0.000008	хор.

В табл. 3 приведено сравнение методов хорд, бисекции и метода Ньютона для расчета корня. x_* - корень, высчитанный по методу ‘*’, k_* – количество итераций, затраченное на приближение корня методом ‘*’, где ‘*’ принимает значения: H – метод хорд, B – метод бисекции и N – метод Ньютона.

Выводы.

Проанализировав результаты применения метода Ньютона, можно сказать, что при расчете данной функции он дает очень хорошие результаты, и сходится за минимальное среди рассмотренных методов число итераций, которое соответствует теоретическому значению порядка.

По результатам эксперимента по определению обусловленности метода Ньютона можно оценить абсолютную обусловленность как значение примерно равное 0.1, что говорит об очень хорошей обусловленности метода.

Проанализировав данные вычислительного эксперимента по трем методам: Ньютона, хорд и бисекции, можно сделать вывод, что для данной функции метод Ньютона сходится наиболее быстрым образом и имеет преимущество перед рассмотренными в сравнении методами. Из недостатков метода можно выделить необходимость расчета аналитической формулы первой производной функции, и значения второй производной.

Таблица 3 – Сравнение методов Ньютона, хорд и бисекции

Значение eps	Значение $delta$	Значение x_B	Значение x_H	Значение x_N	Значение k_B	Значение k_H	Значение k_N
0.1	0.001	0.875	0.858529	0.872102	2	3	2
0.01	0.001	0.863281	0.865001	0.867114	6	4	3
0.001	0.001	0.867676	0.867002	0.867114	9	5	3
0.0001	0.001	0.866943	0.867002	0.867114	9	5	3
0.00001	0.001	0.866943	0.867002	0.867114	9	5	3
0.000001	0.001	0.866943	0.867002	0.867114	9	5	3
0.1	0.0001	0.875	0.858435	0.872037	2	3	2
0.01	0.0001	0.863281	0.864908	0.867108	6	4	3
0.001	0.0001	0.867676	0.866953	0.867108	9	6	3
0.0001	0.0001	0.867126	0.867086	0.867108	12	8	3
0.00001	0.0001	0.867081	0.867086	0.867078	13	8	4
0.000001	0.0001	0.876081	0.867086	0.867078	13	8	4
0.1	0.00001	0.875	0.858425	0.872045	2	3	2
0.01	0.00001	0.863281	0.864907	0.867107	6	4	3
0.001	0.00001	0.867676	0.866949	0.867107	9	6	3
0.0001	0.00001	0.867126	0.867075	0.867107	12	8	3
0.00001	0.0001	0.867092	0.867084	0.86786	16	9	4
0.000001	0.0001	0.867084	0.867084	0.86786	17	9	4

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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

double delta;

#ifndef __NEWTON
#define __NEWTON
#endif

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

#ifndef FF(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) )
#define FFF(x) ( (PI2*pow(x, M_PI + 10) - M_PI * pow(x, M_PI + 10) + 3 *
PI2*pow(x, M_PI + 6) - 3 * M_PI*pow(x, M_PI + 6) + 3 * PI2*pow(x, M_PI +
2) - 3 * M_PI*pow(x, M_PI + 2) - 20 * pow(x, 6) + PI2 * pow(x, M_PI - 2)
- M_PI*pow(x,M_PI-2) + 12 * pow(x, 2)) / (pow(x, 12) + 3 * pow(x, 8) + 3
* pow(x, 4) + 1) )
#endif

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 f;
}
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 = 1.154884,  // наименьшее значение модуля 1-ой производной
         M2 = 7.268115;  // 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,k_N;
     long int s;
     float a1, b1, eps1, delta1;
     double a, b, eps, x_B, x_H, x_N;

     a = 0.5;
     b = 1.25;

     double x = 0.867086;
     printf("eps\t\tdelta\t\ta\t\tb\t\tx_B\t\tx_H\t\tx_N\t\t\tk_B\tk_H\t
k_N\tDx\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);
               x_N = NEWTON(b, eps, k_N);

     printf("%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%d\t%d\t%d\t%lf\t%d\n",
eps, delta, a, b, x_B, x_H, x_N, k_B, k_H, k_N, abs(x-x_N), eps>=abs(x -
x_N));
          }
     }
     return 0;
}
```