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

ОТЧЕТ

по практической работе №6

по дисциплине «Вычислительная математика»

Тема: Метод простых итераций

Студент гр. 8383 _____ Ларин А.

Преподаватель _____ Сучков А.И.

Санкт-Петербург

2019

Цель работы.

Формирование практических навыков нахождения корней алгебраических и трансцендентных уравнений методом простых итераций.

Основные теоретические положения.

Метод простых итераций (метод последовательных приближений) решения уравнения $f(x) = 0$ состоит в замене исходного уравнения эквивалентным ему уравнением $x = \varphi(x)$ и построении последовательности $x_{n+1} = \varphi(x_n)$, сходящейся при $n \to \infty$ к точному решению. Достаточные условия сходимости метода простых итераций формулируются следующей теоремой.

Теорема. Пусть функция $\varphi(x)$ определена и дифференцируема на $[a, b]$, причём все её значения $\varphi(x) \in [a, b]$. Тогда, если существует число q, такое, что $|\varphi'(x)| \leqslant q < 1$ на отрезке $[a, b]$, то последовательность $x_{n+1} = \varphi(x_n)$, $n = 0,1,2, \ldots$ сходится к единственному на $[a, b]$ решению уравнения $x = \varphi(x)$ при любом начальном значении $x_0 \in [a, b]$, т.е.

$$\lim_{n\to\infty} x_n = \lim_{n\to\infty} \varphi(x_n) = \xi; \ f(\xi) = 0; \ \xi \in [a, b].$$

При этом если на отрезке $[a, b]$ производная $\varphi'(x)$ положительна, то

$$|\xi - x_n| < \frac{q}{1-q}|x_n - x_{n-1}|,$$

если φ′(x) отрицательна, то

$$|\xi - x_n| < |x_n - x_{n-1}|.$$

Рассмотрим один шаг итерационного процесса. Исходя из найденного на предыдущем шаге значения x_{n-1}, вычисляется $y = \varphi(x_{n-1})$. Если $|y - x_{n-1}| > \varepsilon$, то полагается $x_n = y$ и выполняется очередная итерация. Если же $|y - x_{n-1}| < \varepsilon$, то вычисления заканчиваются и за приближенное значение корня принимается величина $x_n = y$. Погрешность результата вычислений зависит от знака производной $\varphi'(x)$: при $\varphi'(x) > 0$ погрешность определения корня составляет $\frac{q\varepsilon}{1-q}$, а при $\varphi'(x) < 0$ погрешность не превышает ε. Существование числа q является условием сходимости метода в соответствии с отмеченной выше

теоремой. Для применения метода простых итераций определяющее значение имеет выбор функции $\varphi(x)$ в уравнении $x = \varphi(x)$, эквивалентном исходному. Функцию необходимо подбирать так, чтобы $|\varphi'(x)| \leqslant q < 1$. Это обусловливается тем, что если $\varphi'(x) < 0$ на отрезке $[a, b]$, то последовательные приближения $x_n = \varphi(x_{n-1})$ будут колебаться около корня ξ, если же $\varphi'(x) > 0$, то последовательные приближения будут сходиться к корню ξ монотонно. Следует также помнить, что скорость сходимости последовательности $\{x_n\}$ к корню ξ функции тем выше, чем меньше число q.

Постановка задачи.

Используя программы-функции `ITER` и `Round` из файла methods.cpp (файл заголовков metods.h), найти корень уравнения с заданной точностью `Eps` методом простых итераций, исследовать скорость сходимости и обусловленности метода. Порядок выполнения работы следующий:

1. Графически или аналитически отделить корень уравнения $f(x) = 0$.
2. Преобразовать уравнение $f(x) = 0$.
3. к виду $x = \varphi(x)$ так, чтобы в некоторой окрестности $[a, b]$ корня производная $\varphi'(x)$ удовлетворяла условию $|\varphi'(x)| \leqslant q < 1$. При этом следует иметь в виду, что чем меньше величина q, тем быстрее последовательные приближения сходятся к корню.
4. Выбрать начальное приближение, лежащее на отрезке $[a, b]$.
5. Составить подпрограмму для вычисления значений $\varphi(x)$, предусмотрев округление вычисленных значений с точностью `Delta`.
6. Составить головную программу, вычисляющую корень уравнения и содержащую обращение к программам `PHI` и `ITER` и индикацию результатов.
7. Провести вычисления по программе. 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 [a, b], на которых функция удовлетворяет начальным условиям теоремы о сходимости метода простых итераций. По графику на рис. 1 видно что корень принадлежит отрезку $[0.5, 1.25]$ и первая производная в окрестности корня положительна.

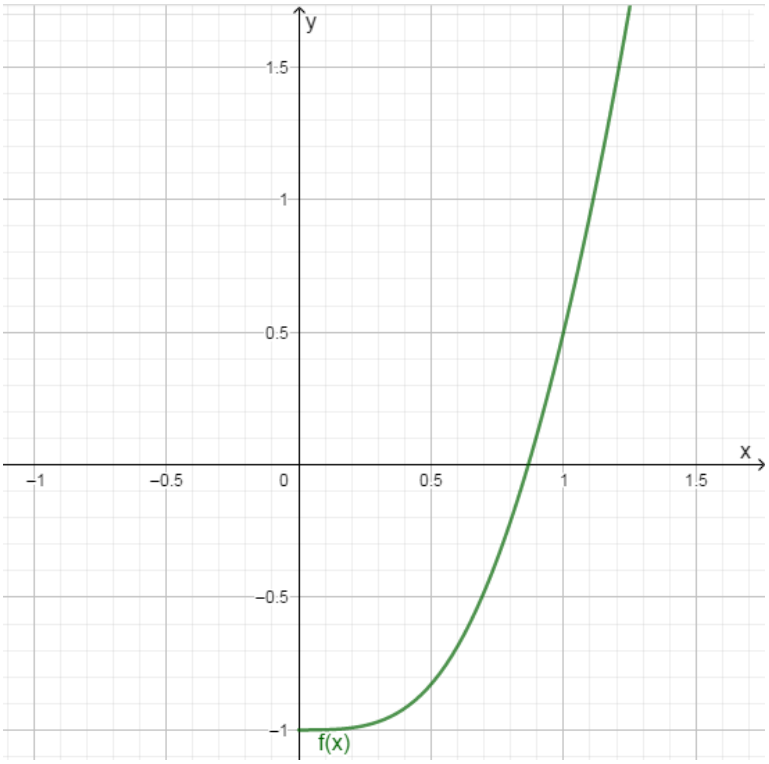

Рисунок 1 – Локализация корня функции $f(x)$

Метод простых итераций $f(x) = 0$ состоит в замене исходного уравнения эквивалентным ему уравнением $x = \varphi(x)$. Возьмем в качестве $\varphi(x)$ следующую функцию: $\varphi(x) = x - \lambda * f(x)$. Найдем оптимальное значение λ, удовлетворяющее условиям сходимости. По условию требуется существования q, такого, что $|\varphi(x)'| \leq q < 1$. Имеем $|\varphi(x)'| = 1 - \lambda * f'(x)$. Отсюда получаем следующие ограничения: Во-первых $\lambda * f'(x) > 0$. По графику видим, что производная исходной функции положительна на всем отрезке, следовательно коэффициент λ должен быть положительным. Во вторых $|1 - \lambda * f'(x)| \leq q <$

$1 \Rightarrow \lambda * f'(x) \leq q + 1 < 2$. Для этого проверим, что $\lambda M_1 \leq q < 2$, где $M_1 = \max_x |f'(x)|$, следовательно $\frac{1}{\lambda} > \frac{M_1}{2}$. Для нашей функции $M_1 = 5.725966$. Отсюда $\lambda < \frac{2}{M_1} = 0.349286041$. Примем $\lambda = 0.3$. Получаем:

$$\varphi(x) = x - 0.3 * f(x).$$

За произвольное приближение возьмем $x_0 = b$.

Исследуем экспериментально скорость сходимости метода. Согласно неравенству $|\xi - x_n| < |x_n - x_{n-1}|$ метод имеет линейный порядок сходимости. Результаты эксперимента занесены в табл. 1.

Проведем вычисление корня функций $f(x)$ и исследуем скорость сходимости метода 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, PHI – функция, итеративно приближающая корень. В табл. 1 приведены расчеты корня $\overline{x}$ при различных значениях `delta` и `eps`, и представлены значения количества итераций.

Таблица 1 – Расчет корня x методом простой итерации с варьированием значения *eps* и *delta*

Значение eps	Значение $delta$	Значение a	Значение b	Значение x	Значение k
0.1	0.00001	0.5	1.25	0.8526	2
0.01	0.00001	0.5	1.25	0.8671	3
0.001	0.00001	0.5	1.25	0.86709	4
0.0001	0.00001	0.5	1.25	0.86709	4
0.1	0.000001	0.5	1.25	0.852598	2
0.01	0.000001	0.5	1.25	0.867099	3
0.001	0.000001	0.5	1.25	0.867084	4

Имея экспериментальное значение скорость сходимости метода видим, что скорость сходимости метода хорд линейна. Действительно, согласно неравенству $|\xi - x_n| < |x_n - x_{n-1}|$ порядок сходимости метода равен 1, т.е. линеен.

Теперь, имея приближение корня, примем $\overline{x} \approx 0.867084$. 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.

Выводы.

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

По результатам эксперимента по определению обусловленности метода простых итераций можно оценить абсолютную обусловленность как значение в диапазоне $(0.1; 1]$, что говорит о хорошей обусловленности метода. Из недостатков метода можно выделить дополнительные ограничения на начальные условия, в частности необходимость подбора функции $\varphi(x)$, соответствующую этим ограничениям.

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

Значение eps	Значение $delta$	Значение x	Значение k	Значение Δx	Значение $eps \geq \Delta x$
0.1	0.01	0.85	2	0.017084	хор.
0.01	0.01	0.87	3	0.002916	хор.
0.001	0.01	0.87	3	0.002916	пл.
0.0001	0.01	0.87	3	0.002916	пл.
0.00001	0.01	0.87	3	0.002916	пл.
0.000001	0.01	0.87	3	0.002916	пл.
0.1	0.001	0.853	2	0.014084	хор.
0.01	0.001	0.867	3	0.000084	хор.
0.001	0.001	0.867	3	0.000084	хор.
0.0001	0.001	0.867	3	0.000084	хор.
0.00001	0.001	0.867	3	0.000084	пл.
0.000001	0.001	0.867	3	0.000084	пл.
0.1	0.0001	0.8526	2	0.014484	хор.
0.01	0.0001	0.8671	3	0.000016	хор.
0.001	0.0001	0.8671	3	0.000016	хор.
0.0001	0.0001	0.8671	3	0.000016	хор.
0.00001	0.0001	0.8671	3	0.000016	пл.

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 __ITER
#define __ITER
#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 lambda = 0.30;
      extern double delta;
  // функция ϕ(x) - для метода простых итераций

      double Phi = x-lambda*F(x);
      Phi = Round(Phi, delta);
      return Phi;
}

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_I\t\tk_I\tDx\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);
                  x_I = ITER(b, eps, k_I);
                  printf("%lf\t%lf\t%lf\t%lf\t%lf\t%d\t%lf\t%d\n", eps,
delta, a, b, x_I, k_I, abs(x-x_I), eps>=abs(x - x_I));
            }
      }     return 0;
}
```