Министерство образования науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский авиационный институт (национальный исследовательский университет)»
«Информационные технологии и прикладная математика»

Курсовой проект
По курсу «Вычислительные системы»
1 семестр

Задание 4:
«Процедуры и функции в качестве параметров»

Группа:	М8О-106Б-21
Студент:	Орусский В.Р.
Преподаватель:	Дубинин А.В.
Оценка:	
Дата:	

Москва, 2021

Оглавление

Введение

Изучить и реализовать три численных метода нахождения приближённого значения корня трансцендентных и не только уравнений: Метод бисекции, метод итераций и метод Ньютона.

Постановка задачи

Составить программу на языке Си с процедурами решения трансцендентных алгебраических уравнений различными численными методами (итераций, Ньютона и половинного деления — дихотомия). Нелинейные уравнения оформить как параметры-функции, разрешив относительно неизвестной величины в случае необходимости. Применить каждую процедуру к решению двух уравнений, заданных двумя строками таблицы, начиная с варианта с заданным номером.

Вариант 8:

Уравнение:

$$0{,}6 \cdot 3^{x} - 2{,}3x - 3 = 0$$

Отрезок содержащий корень: [2;3]

Базовый метод: Бисекция

Приближённое значение корня:

2.4200

Вариант 9:

Уравнение:

$$x^{2} - \ln(1 + x) - 3 = 0$$

Отрезок содержащий корень: [2;3]

Базовый метод: Метод итераций

Приближённое значение корня:

2.0267

Теоретическая часть

Метод бисекции или метод деления отрезка пополам — простейший численный метод для решения нелинейных уравнений вида f(x)=0. Предполагается только непрерывность функции f(x). Поиск основывается на теореме о промежуточных значениях.

Алгоритм основан на следующем следствии из теоремы Больцано — Коши:

Пусть непрерывная функция $f(x) \in \mathrm{C}([a,\ b])$, тогда, если $sign(f(a)) \neq sign(f(b))$, то $\exists c \in [a,\ b] :\ f(c) = 0$.

Описание алгоритма

Задача заключается в нахождении корней нелинейного уравнения $\mathcal{F}(x) = 0$

Для начала итераций необходимо знать отрезок [x_{left}; x_{right}] значений x, на концах которого функция принимает значения противоположных знаков.

Противоположность знаков значений функции на концах отрезка можно определить множеством способов. С помощью функции определения знака числа и сравнения двух значений, а можно использовать умножение значений функции на концах отрезка и определять знак произведения путём сравнения результата с нулём. Именно этим методом мы и будем пользоваться в программе.

Для каждой из итераций нам надо будет находить середину отрезка формулой:

$$x_c = (x_{left} + x_{right})/2$$

Далее, при вычислении функции от x_c, если оно равно 0, то корень найден, если нет, то наш отрезок делится на два под отрезка, и внутри них происходит вычисление противоположности знаков на концах этих отрезков. Если $f(x_{left}) * f(x_c)$>0, то мы приравниваем левую границу к x_c, если же результат вычисления меньше, то правую границу.

Метод итераций — довольно простой численный метод решения уравнений. Метод основан на принципе сжимающего отображения, который применительно к численным методам в общем виде так же может называться методом простой итерации. Идея данного метода состоит в замене исходного уравнения f(x)=0 на эквивалентное ему $x = \varphi(x)$ так, чтобы отображение $\varphi(x)$ было сжимающим. При чём должно выполнятся условие сходимости $|\varphi'(x)| < 1$ на всём отрезке от левого до правого края.

Итерации начинаются со значения x_cсередины отрезка. Однако $\varphi(x)$ может быть выбрана неоднозначно. Сохраняет корни уравнения такое преобразование:

$$\varphi(x) = x - \lambda(x) * f(x)$$

Здесь λ – постоянная, которая не зависит от количества шагов. В данном случае мы возьмём

$$\lambda = \frac{1}{f'(x_{c)}}$$

что приводит к простому методу одной касательной и имеет условие сходимости

$$\lambda * f'(x) > 0$$

Тогда итерационный процесс выглядит так:

$$x^{k+1} = x^k - \lambda * f(x^k)$$

Условием окончания итераций является достижение нужной точности между предыдущим и следующим значением.

Метод Ньютона — итерационный численный метод нахождения корня заданной функции, который является частным случаем метода итераций. А именно за λ берётся значение производной в каждой новой точке. Тогда итерационный процесс имеет вид

$$x^{k+1} = \frac{x^k - f(x^k)}{f'(x^k)}$$

Условие окончания итераций и начальное значение абсолютно такие же, как и в методе итерации. Условие сходимости метода можно записать как

$$|f(x) * f''(x)| < (f'(x))^2$$

Аналитическое решение уравнений

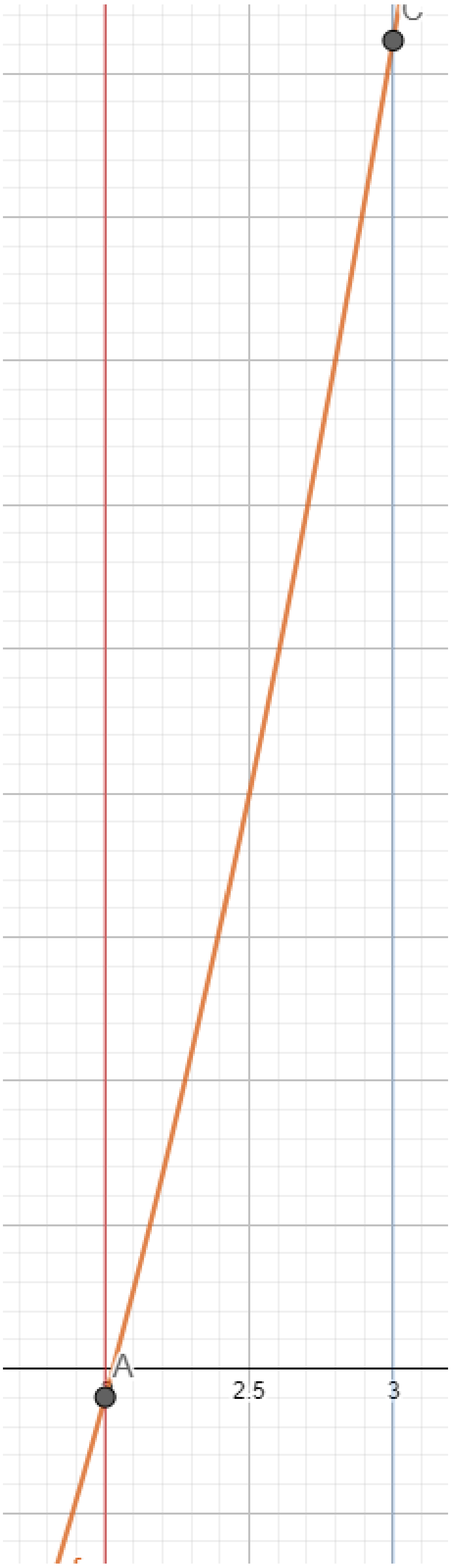
Метод бисекции:

$$f(x) = 0.6 * 3^x - 2.3x - 3, x \in [2; 3]$$

Из графика видно, что функция на концах отрезка [2;3] (точки A и B) имеет разные знаки => метод бисекции применим.

$$f(x) = x^2 - \ln(x+1) - 3, x \in [2; 3]$$

Из графика ниже, мы можем наблюдать, что значения функции на концах отрезка (точки A и C) имеют противоположные знаки, что позволяет применять нам метод половинного деления.



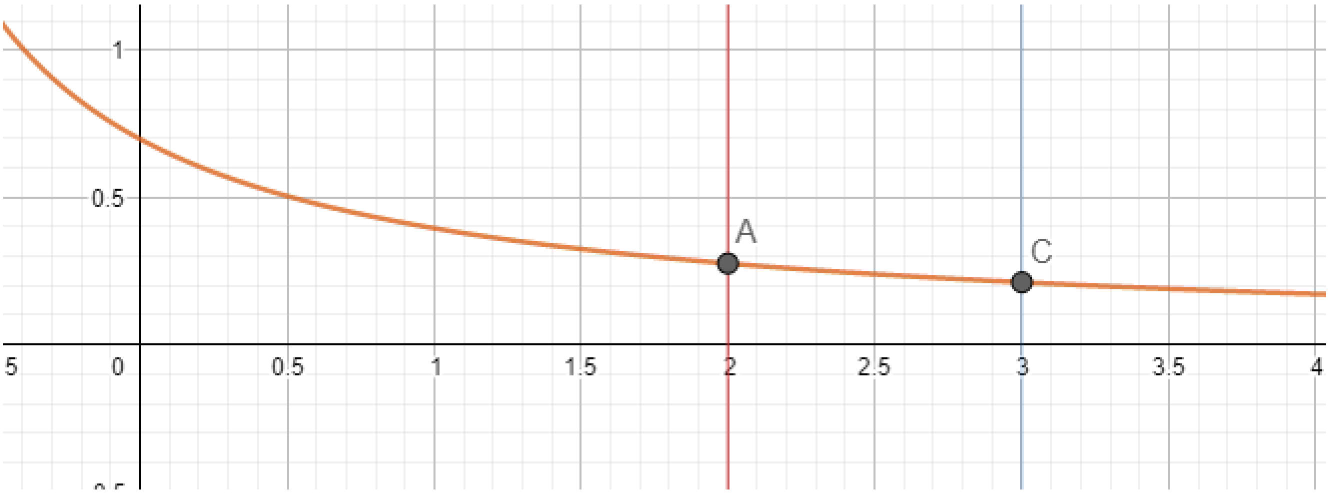
Метод простой итерации:

$$f(x) = 0.6 * 3^x - 2.3x - 3, x \in [2; 3]$$

$$\varphi(x) = \log_3 \frac{2.3x + 3}{0.6}$$

$$\varphi'^{(x)} = \frac{2.09355}{(3 + 2.3\,x)}$$

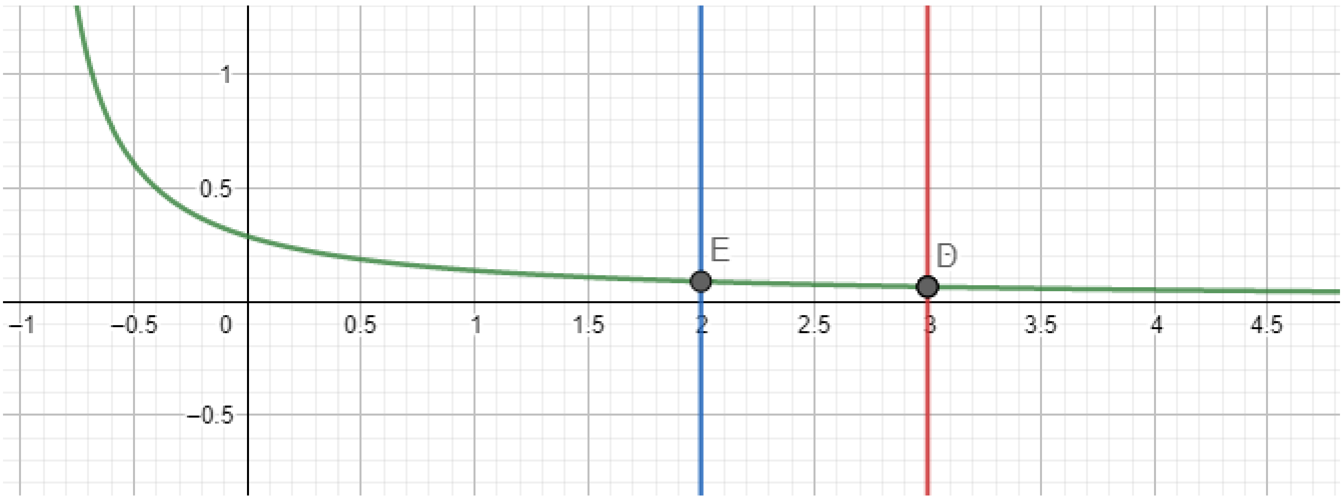
Производная функции меньше 1 при всех значениях отрезка [2;3]



$$f(x) = x^2 - \ln(x + 1) - 3, x \in [2; 3]$$

$$\varphi(x) = \sqrt{(\ln(x + 1) + 3)}$$

$$\varphi'^{(x)} = \frac{2.09355}{(3 + 2.3\,x)}$$



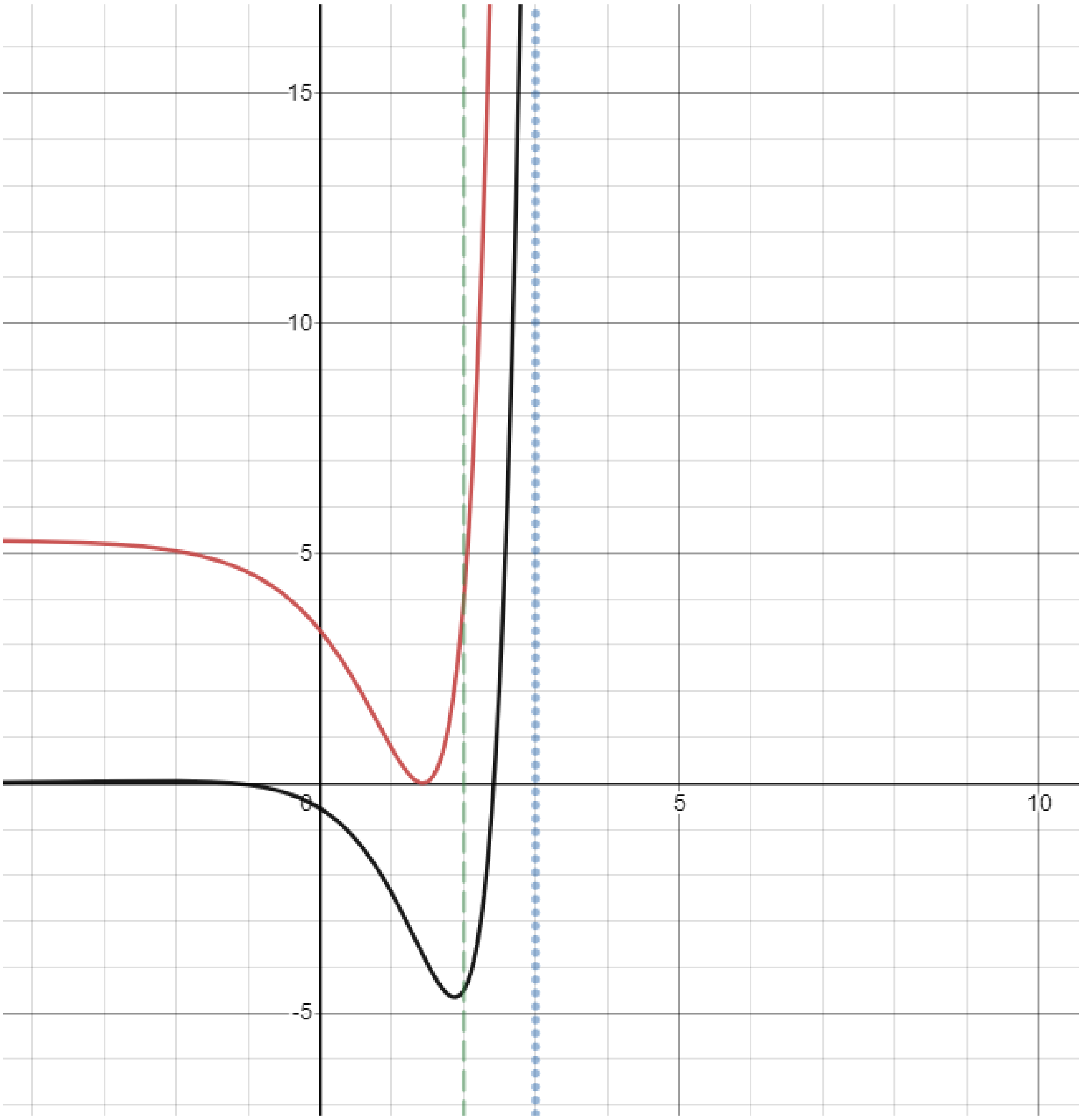
На данном графике мы наблюдаем, что все точки производной между левой и правой гранью (точки E и D) лежат ниже 1 по модулю.

Метод Ньютона:

$$f(x) = 0.6 * 3^x - 2.3x - 3, x \in [2; 3]$$

$$f'(x) = \ln 3 * 3^x - 2.3$$

$$f''(x) = 3^x * (\ln 3)^2$$

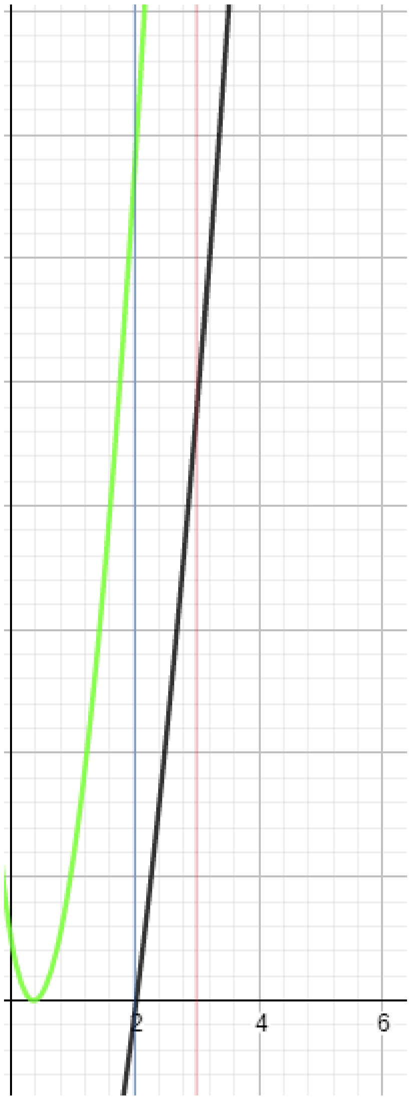


Из графика видно, что на отрезке от 2 до 3 $f(x) * f''(x) < f'(x)^2$

$$f(x) = x^2 - \ln(x+1) - 3, x \in [2;3]$$

$$f'(x) = 2x - \frac{1}{(x+1)}$$

$$f''(x) = \frac{1}{(1+x)^2} + 2$$



Несложно заметить, что график первой производной в квадрате (зелёный цвет) на промежутке от 2 до 3 находится выше произведения функции на её вторую производную.

Вывод по аналитическому разбору:

Оба уравнения должны быть решены всеми тремя методами, поскольку для каждого из них они работают.

Описание алгоритма

Первым делом необходимо найти машинный эпсилон, на котором будет основываться точность вычисления. Это делается до тех пор пока $1 + \varepsilon > 1$, при том что ε изначально равен 1.0, а на каждой итерации мы его делим пополам.

Далее, введём функции, с которыми нам предстоит работать, вычислим для них вручную зеркальное отображение и запишем его в функцию. Далее, реализуем производную первого и второго порядка, и реализуем функции, которые проверяют условие выполнимости каждого из численного методов.

Все функции производных, зеркальных отображений зададим в виде функций-параметров. В случае неприменимости алгоритма, кол-во итераций приравниваем к «-1», после чего делаем дополнительную проверку на это условие и в таком случае сообщаем о том, что данный алгоритм не применим к уравнению. Но, после аналитического решения, мы понимаем, что такого не может быть.

Исходный код программы:

```
/*
      КП №4
     Вариант №8/9
     P.S. метод дихотомии это про поиск минимума на отрезке функции
     а это называется метод бисекции

   Время актуализировать все справочные сведения по лабам и КП.
 */

#include <stdio.h>
#include <math.h>

typedef double (*func)(double);

typedef struct {
  double root;
  int iterations;
} solution;

// Получение машинного эпсилона
double get_machine_epsilon(int accuracy) {
  double epsilon = 1.0;
  while ((1.0 + epsilon / 2.0) > 1.0) {
   epsilon /= 2.0;
  }
  epsilon = epsilon * pow(10, 16 - accuracy);
  return epsilon;
}

//первая используемая функция
double first_func(double x) {
  return 0.6 * pow(3, x) - 2.3 * x - 3;
}

//вторая используемая функция
double second_func(double x) {
  return pow(x, 2) - log(1 + x) - 3;
}

//зеркальное отображение первой функции
double mirrored_first_func(double x) {
  return log((2.3 * x + 3) / 0.6) / log(3);
}

//зеркальное отображение второй функции
double mirrored_second_func(double x) {
  return sqrt(log(x + 1) + 3);
}

//достаточное условие для метода бисекции
int bisection_condition(func f, double start, double end) {
  return f(start) * f(end) < 0;
}

//реализация самого метода бисекции
solution bisection(func f, double start, double end, double epsilon) {
  solution sol;
  sol.iterations = 0;
  sol.root = NAN;
  if (f(start) == 0) {
   sol.root = start;
```

```
  } else if (f(end) == 0) {
   sol.root = end;
  } else {
   while ((end - start) > epsilon && bisection_condition(f, start, end)) {
     double x = (start + end) / 2;
     if (f(x) * f(start) > 0) {
      start = x;
     } else {
      end = x;
     }
     sol.iterations++;
   }
  }
  if (bisection_condition(f, start, end)) {
   sol.root = (start + end) / 2;
  } else {
   sol.iterations = -1;
  }
  return sol;
}

//производная функции
double df(func f, double x) {
  double dx = pow(2, -13);
  return (f( x + dx) - f(x - dx)) / (2 * dx);
}

// необходимое условие метода итераций
int iteration_condition(func f, double x) {
  return fabs(df(f, x)) < 1;
}

// реализация метода итераций
solution iterations(func f, double start, double end, double epsilon) {
  solution sol;
  sol.iterations = 1;
  sol.root = NAN;
  double x = (start + end) / 2;
  double x_next = f(x);
  while (fabs(x - x_next) > epsilon && iteration_condition(f, x_next)) {
   x = x_next;
   x_next = f(x);
   sol.iterations++;
  }
  if (iteration_condition(f, x_next)){
   sol.root = x_next;
  } else {
   sol.iterations = -1;
  }
  return sol;
}

double ddf(func f, double x) {
  double dx = pow(2, -13);
  return (f(x + dx) - 2 * f(x) + f(x - dx)) / (dx * dx);
}

int Newton_condition(func f, double x) {
  return fabs(f(x) * ddf(f, x)) < pow(df(f, x), 2);
}

solution Newton_method(func f, double start, double end, double epsilon) {
  solution sol;
  sol.iterations = 1;
```

```
  sol.root = NAN;
  double x = (start + end) / 2;
  double x_next = x - f(x) / df(f, x);
  while (fabs(x - x_next) > epsilon && Newton_condition(f, x_next)) {
   x = x_next;
   x_next = x - f(x) / df(f, x);
   sol.iterations++;
  }
  if (Newton_condition(f, x)) {
   sol.root = x;
  } else {
   sol.iterations = -1;
  }
  return sol;
}

void print_separator() {
  printf("%s", "--------------------------------------------------------------------
----------------------------------------------------");
}

int main() {
  int accuracy_factor;
  scanf("%d", &accuracy_factor);
  double epsilon = get_machine_epsilon(accuracy_factor);
  printf("Machine epsilon equals %.8e\n", epsilon);
  solution answer;
  double a, b;
  // 8 и 9 варианты
  a = 2;
  b = 3;
  answer = bisection(first_func, a, b, epsilon);
  printf("\t Root for 0.6 * 3^x - 2.3x - 3 = 0: \n");
  if (answer.iterations == -1) {
   printf("This method doesn't work!\n");
  } else {
   printf("The bisection method allowed us to find the root %.*f in %d iterations\n",
accuracy_factor, answer.root, answer.iterations);
  }
  print_separator();

  answer = iterations(mirrored_first_func, a, b, epsilon);
  if (answer.iterations == -1) {
   printf("This method doesn't work!\n");
  } else {
   printf("The iterations method allowed us to find the root %.*f in %d
iterations\n", accuracy_factor, answer.root, answer.iterations);
  }
  print_separator();

  answer = Newton_method(first_func, a, b, epsilon);
  if (answer.iterations == -1) {
   printf("This method doesn't work!\n");
  } else {
   printf("The Newton method allowed us to find the root %.*f in %d iterations\n",
accuracy_factor, answer.root, answer.iterations);
  }
  print_separator();
  printf("\v\t Root for x^2 - ln(1 + x) - 3 = 0: \n");

  answer = Newton_method(second_func, a, b, epsilon);
  if (answer.iterations == -1) {
   printf("This method doesn't work!\n");
  } else {
```

```
    printf("The bisection method allowed us to find the root %.*f in %d iterations\n",
accuracy_factor, answer.root, answer.iterations);
  }
  print_separator();

  answer = iterations(mirrored_second_func, a, b, epsilon);
  if (answer.iterations == -1) {
    printf("This method doesn't work!\n");
  } else {
    printf("The iterations method allowed us to find the root %.*f in %d
iterations\n", accuracy_factor, answer.root, answer.iterations);
  }
  print_separator();

  answer = Newton_method(second_func, a, b, epsilon);
  if (answer.iterations == -1) {
    printf("This method doesn't work!\n");
  } else {
    printf("The Newton method allowed us to find the root %.*f in %d iterations\n",
accuracy_factor, answer.root, answer.iterations);
  }
  print_separator();
}
```

Переменные в программе:

Название	Тип данных	За что отвечает
a	double	Начальное значение отрезка
b	double	Конечное значение отрезка
accuracy_factor	integer	Точность
epsilon	double	Машинный эпсилон
answer	solution	Структура содержащая ответ (корень и кол-во итераций)

Пользовательские структуры данных в программе:

Название	Поля	Значения
solution	int iterations;	Кол-во итераций
	double root;	Значение корня

Функции в программе:

Название	Возвращаемый тип	Функционал
get_machine_epsilon	double	Подсчитывает машинный эпсилон с учётом необходимой точности
first_func	double	Первая функция (8 вариант)
second_func	double	Вторая функция (9 вариант)
mirrored_first_func	double	Зеркальное отражение первой функции
mirrored_second_func	double	Зеркальное отражение второй функции
bisection_condition	integer	Проверка сходимости метода бисекции
bisection	solution	Метод бесикции
df	double	Прозводная первого порядка
iteration_condition	int	Проверка сходимости метода простых итераций
iterations	solution	Метод итераций
ddf	double	Производная второго порядка
Newton_condition	int	Проверка сходимости метода Ньютона
Newton_method	solution	Метод Ньютона
print_separator	void	Вывод разделителя

Входные данные:

На вход с клавиатуры подаётся единственное целое число accuracy_factor от 0 до 16, которое обозначает коэффициент для точности вычисления искомого корня.

Выходные данные

В начале программа выводит вычисленное нами значение машинного эпсилона, а затем 6 информационных строк о решении заданных нами уравнений и найденном корне за какое-то количество итераций.

Каждая строка содержит найденный корень (если его возможно найти данным методом), количество итераций, затраченных на поиск. Если метод неприменим, выводится сообщение об этом.

Методы применяются и выводятся в следующем порядке: метод бисекции, метод итераций, метод Ньютона.

Протокол исполнения и тесты

Тест №1

```
2
Machine epsilon equals 2.22044605e-002
        Root for 0.6 * 3^x - 2.3x - 3 = 0:
The bisection method allowed us to find the root 2.41 in 6 iterations
------------------------------------------------------------------------------------------------------------------------
The iterations method allowed us to find the root 2.42 in 2 iterations
------------------------------------------------------------------------------------------------------------------------
The Newton method allowed us to find the root 2.42 in 2 iterations
------------------------------------------------------------------------------------------------------------------------
♂       Root for x^2 - ln(1 + x) - 3 = 0:
The bisection method allowed us to find the root 2.03 in 3 iterations
------------------------------------------------------------------------------------------------------------------------
The iterations method allowed us to find the root 2.03 in 3 iterations
------------------------------------------------------------------------------------------------------------------------
The Newton method allowed us to find the root 2.03 in 3 iterations
------------------------------------------------------------------------------------------------------------------------
```

Тест №2

```
10
Machine epsilon equals 2.22044605e-010
        Root for 0.6 * 3^x - 2.3x - 3 = 0:
The bisection method allowed us to find the root 2.4199816462 in 33 iterations
------------------------------------------------------------------------------------------------------------------------
The iterations method allowed us to find the root 2.4199816463 in 15 iterations
------------------------------------------------------------------------------------------------------------------------
The Newton method allowed us to find the root 2.4199816464 in 4 iterations
------------------------------------------------------------------------------------------------------------------------
♂       Root for x^2 - ln(1 + x) - 3 = 0:
The bisection method allowed us to find the root 2.0266892632 in 5 iterations
------------------------------------------------------------------------------------------------------------------------
The iterations method allowed us to find the root 2.0266892632 in 10 iterations
------------------------------------------------------------------------------------------------------------------------
The Newton method allowed us to find the root 2.0266892632 in 5 iterations
------------------------------------------------------------------------------------------------------------------------
```

Тест №3

```
0
Machine epsilon equals 2.22044605e+000
        Root for 0.6 * 3^x - 2.3x - 3 = 0:
The bisection method allowed us to find the root 3 in 0 iterations
------------------------------------------------------------------------------------------------------------------------
The iterations method allowed us to find the root 2 in 1 iterations
------------------------------------------------------------------------------------------------------------------------
The Newton method allowed us to find the root 3 in 1 iterations
------------------------------------------------------------------------------------------------------------------------
♂       Root for x^2 - ln(1 + x) - 3 = 0:
The bisection method allowed us to find the root 3 in 1 iterations
------------------------------------------------------------------------------------------------------------------------
The iterations method allowed us to find the root 2 in 1 iterations
------------------------------------------------------------------------------------------------------------------------
The Newton method allowed us to find the root 3 in 1 iterations
------------------------------------------------------------------------------------------------------------------------
```

Тест №4

```
15
Machine epsilon equals 2.22044605e-015
        Root for 0.6 * 3^x - 2.3x - 3 = 0:
The bisection method allowed us to find the root 2.419981646227694 in 49 iterations
------------------------------------------------------------------------------------------------------------------------
The iterations method allowed us to find the root 2.419981646227694 in 23 iterations
------------------------------------------------------------------------------------------------------------------------
The Newton method allowed us to find the root 2.419981646227694 in 5 iterations
------------------------------------------------------------------------------------------------------------------------
♂       Root for x^2 - ln(1 + x) - 3 = 0:
The bisection method allowed us to find the root 2.026689263243525 in 6 iterations
------------------------------------------------------------------------------------------------------------------------
The iterations method allowed us to find the root 2.026689263243525 in 15 iterations
------------------------------------------------------------------------------------------------------------------------
The Newton method allowed us to find the root 2.026689263243525 in 6 iterations
------------------------------------------------------------------------------------------------------------------------
```

Вывод

В работе было задействовано изучение реализации производных первых двух порядков в представлении ЭВМ, три численных метода по нахождении приблизительного значения корня в трансцендентном уравнении: бисекции, простых итераций, Ньютона. В процессе написания программы было изучено аналитические условия применимости данных методов и их реализация в коде.

Были составлены форматы ввода и вывода, проведены тесты программы на разных значениях точности и их расхождения с фактическим значением корня.

Список литературы

1. https://ru.wikipedia.org/wiki/Метод_бисекции - Метод бисекции.
2. https://ru.wikipedia.org/wiki/Метод_простой_итерации - Метод простой итерации.
3. https://ru.wikipedia.org/wiki/Метод_Ньютона – Метод Ньютона.
4. https://www.geogebra.org/calculator - Построение графиков функций.