МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования

«Дальневосточный федеральный университет» (ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ

Департамент математического и компьютерного моделирования

ОТЧЁТ по лабораторной работе № 7
«Численное решение уравнений»

Вариант № 8

Выполнила: студент гр. Б9122-02.03.01сцт
Ф.И.О.
Ильяхова Алиса Алексеевна

Проверил: преподаватель
Ф.И.О.
Павленко Елизавета Робертовна

Владивосток

2024

Цель работы:

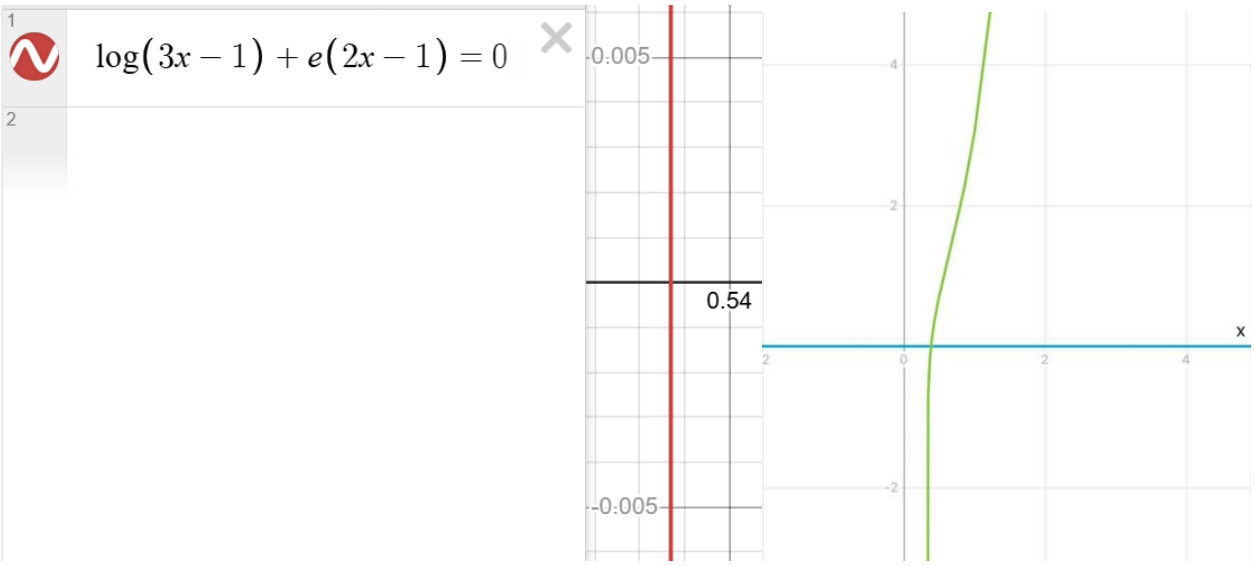
1. Определить примерный интервал, в котором может располагаться необходимый корень.
2. Численно найти решение тремя известными методами.
3. Сформировать сравнительную таблицу, отражающую сходимость методов.
4. Сделать вывод о проделанной работе.

Основное:

1.1. Данные:

0 = lg(3x – 1) + exp(2x – 1) =>

≈0.539



Следовательно, возьмём приблизительный промежуток [0.375, 1.5].

1.2. Методы:

Метод Хорд.

Численный метод для приближенного нахождения корня нелинейного уравнения. Он основан на идее использования хорды (отрезка, соединяющего две точки на графике функции), чтобы приблизиться к корню.

```
def chord_method(eps, lst: list):
    table = PrettyTable()
    table.field_names = ["Итерация", "Значение x"]

    counter = 1
    b = lst[1]
    x_prev = lst[0]
```

```
    x_next = x_prev - (b - x_prev) * function(x_prev) /
(function(b) - function(x_prev))
    table.add_row([counter, x_next])

    while abs(x_next - x_prev) > eps:
        x_prev = x_next
        x_next = x_prev - (b - x_prev) * function(x_prev) /
(function(b) - function(x_prev))
        counter += 1
        print(x_next)
        table.add_row([counter, x_next])

    print(table.get_string(border=True, header=False, hrules=1))
    return x_next
```

Алгоритм:

1. Вычисляется новое значение x_next на основе текущего приближения x_prev и значения функции в точке x_prev.
2. Затем проверяется условие остановки: (|x_next - x_prev| > eps).
3. Если условие не выполнено, то происходит обновление x_prev, вычисление нового x_next и увеличение счетчика итераций.
4. После завершения итераций выводится таблица с результатами итераций, сгенерированная с помощью метода get_string объекта table.
5. Наконец, возвращается последнее найденное значение x_next, которое является приближенным значением корня уравнения.

```
Метод хорд:
0.3847017313323218
0.38724378454083397
0.38627490792473745
0.38662854685910897
0.3864972876451194
0.3865457115428931
0.38652780665022435
0.38653442152637657
0.38653197693880764
0.38653288025547305
0.38653254645055984
0.38653266980042317
0.38653262421908025
```

Сходится к 9 итерации.

Метод Ньютона.

```python
def newton_method(eps, x0):
    table = PrettyTable()
    table.field_names = ["Итерация", "Значение x"]

    count = 1
    x_prev = x0
    x_next = x_prev - function(x_prev) /
derivative_function(x_prev)
    table.add_row([count, x_next])

    while abs(x_next - x_prev) > eps:
        count += 1
        x_prev = x_next
        x_next = x_prev - function(x_prev) /
derivative_function(x_prev)
        print(x_next)
        table.add_row([count, x_next])
    print(table.get_string(border=True, header=False, hrules=1))
    return x_next
```

Алгоритм:

1. Выбирается начальное приближение x_0.
2. Вычисляется следующее приближение по формуле: $x_{n+1} = x_n - f(x_n) / f'(x_n)$.
3. Процесс повторяется до тех пор, пока разница между последовательными приближениями не станет меньше заданной точности ϵ.

```
 Метод Ньютона:
0.4853022492704575
0.3540672292973203
0.37453656865765966
0.38527261545833413
0.38652025248442917
0.3865326353368584
0.3865326365179037
```

Сходится к 4 итерации.

Метод бисекции.

Численный метод для нахождения корней уравнений вида $f(x) = 0$. Этот метод требует, чтобы функция $f(x)$ была непрерывной на интервале $[a, b]$ и чтобы значения функции на концах этого интервала имели разные знаки (т.е., $f(a)$ и $f(b)$ должны иметь противоположные знаки).

```python
def bisection_method(eps, lst: list):
    l = lst[0]
    r = lst[1]
    c = 0
    count = 1

    x_prev = r
    x_next = c

    while abs(x_next - x_prev) > eps:
        c = (r + l) / 2

        if function(c) * function(l) > 0:
            l = c
        elif function(c) * function(r) > 0:
            r = c

        x_prev = x_next
        x_next = c
        print(count, x_next)
        count += 1

    return x_next
```

Алгоритм:

1. Выбирается начальный интервал [a, b], в котором функция меняет знак.
2. Вычисляется середина интервала c = (a + b) / 2.
3. Проверяется знак функции в точке c:
   - Если f(c) имеет тот же знак, что и f(a), то c становится новой левой границей интервала.
   - Если f(c) имеет тот же знак, что и f(b), то c становится новой правой границей интервала.
4. Процесс повторяется до тех пор, пока длина интервала не станет меньше заданной точности ϵ.

```
Метод бисекции:
1 1.0
2 1.25
3 1.375
4 1.4375
5 1.46875
6 1.484375
7 1.4921875
8 1.49609375
9 1.498046875
10 1.4990234375
11 1.49951171875
12 1.499755859375
13 1.4998779296875
14 1.49993896484375
15 1.499969482421875
16 1.4999847412109375
17 1.4999923706054688
18 1.4999961853027344
19 1.4999980926513672
20 1.4999990463256836
21 1.4999995231628418
22 1.499999761581421
23 1.4999998807907104
24 1.4999999403953552
```

Сходится к 7 итерации.

Используемые библиотеки:

В ходе работы мне потребовалось использовать следующие библиотеки: numpy, prettytable, math.

Библиотека **numpy** предоставляет поддержку для работы с многомерными массивами и матрицами, а также большое количество математических функций для выполнения операций над этими массивами.

Библиотека **prettytable** предоставляет инструменты для создания красиво оформленных таблиц в Python. Она позволяет отображать данные в удобочитаемом виде, что упрощает их анализ и визуализацию.

Библиотека **math** в Python предоставляет функции для выполнения математических операций над числами. Она включает в себя функции для работы с простыми и сложными математическими операциями, такими как тригонометрия, логарифмы, округления чисел и т.д.

Вывод:

1. Метод хорд сходится за 9 итераций, метод Ньютона за 4, а метод бисекций за 7.
2. Таким образом, в результате проделанной работы и анализу методов можно сделать вывод о том, что в данном случае метод Ньютона сошелся быстрее всех, а именно за 4 итерации.
3. В ходе экспериментов использовалась функция $f(x) = lg(3x – 1) + exp(2x – 1)$ с начальными условиями $x_0 = 1$ для метода Ньютона и интервалом [0.5, 2] для метода бисекций.
4. Вычислительная точность была задана как $\epsilon = 10^{-6}$, что обеспечило необходимую точность результатов.
5. Метод Ньютона показал наивысшую эффективность и быструю сходимость.

6. Метод хорд не требует вычисления производной, но может сходиться медленнее в зависимости от выбора начальных точек.
7. Метод бисекций является надежным и простым в реализации, но может потребовать больше итераций для достижения требуемой точности.

Полный код:

```
import numpy as np
from prettytable import PrettyTable
import math
def function(x):
    return np.log10(3 * x - 1) + np.exp(2 * x - 1)

def derivative_function(x):
    return (3 / (math.log(10) * (3 * x - 1))) + 2 * np.e ** (2 * x
- 1)

def chord_method(eps, lst: list):
    table = PrettyTable()
    table.field_names = ["Итерация", "Значение x"]

    counter = 1
    b = lst[1]
    x_prev = lst[0]
    x_next = x_prev - (b - x_prev) * function(x_prev) /
(function(b) - function(x_prev))
    table.add_row([counter, x_next])

    while abs(x_next - x_prev) > eps:
        x_prev = x_next
        x_next = x_prev - (b - x_prev) * function(x_prev) /
(function(b) - function(x_prev))
        counter += 1
        print(x_next)
        table.add_row([counter, x_next])

    print(table.get_string(border=True, header=False, hrules=1))
    return x_next

def newton_method(eps, x0):
    table = PrettyTable()
    table.field_names = ["Итерация", "Значение x"]
```

```
    count = 1
    x_prev = x0
    x_next = x_prev - function(x_prev) /
derivative_function(x_prev)
    table.add_row([count, x_next])

    while abs(x_next - x_prev) > eps:
        count += 1
        x_prev = x_next
        x_next = x_prev - function(x_prev) /
derivative_function(x_prev)
        print(x_next)
        table.add_row([count, x_next])
    print(table.get_string(border=True, header=False, hrules=1))
    return x_next

def bisection_method(eps, lst: list):
    l = lst[0]
    r = lst[1]
    c = 0
    count = 1

    x_prev = r
    x_next = c

    while abs(x_next - x_prev) > eps:
        c = (r + l) / 2

        if function(c) * function(l) > 0:
            l = c
        elif function(c) * function(r) > 0:
            r = c

        x_prev = x_next
        x_next = c
        print(count, x_next)
        count += 1

    return x_next

print("Метод хорд:")
```

```python
chord_method(0.0000001, [0.375, 1.5])

print("\n Метод Ньютона:")
newton_method(0.0000001, 1.5)

print("\n Метод бисекции:")
bisection_method(0.0000001, [0.5, 1.5])
```