# Отчет по лабораторной работе № 7
# Вариант № 3

Винницкая Дина Сергеевна

Группа: Б9122-02-03-01сцт

## 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

## Входные данные:

1. **Уравнение:** $0 = 2^x - 2x^2 - 1$

## Определение области рассмотрения

Для функции $2^x - 2x^2 - 1$ необходимо определить область рассмотрения, учитывая особенности экспоненциальной и полиномиальной функций.

**Определение области рассмотрения**

- Для начала, исследуем корни полинома. Простейший способ найти корень полинома - это приравнять его к нулю и решить полученное уравнение. В данном случае, уравнение можно численно решить, и находится, что корень приблизительно равен 1.44.
- Поскольку полином третьей степени быстро возрастает, а экспонента быстро убывает в положительной области, положительный корень уравнения находится не сильно дальше от 1.44.
- Следовательно, пока что возьмем промежуток [0,3, 1,5] для определения области рассмотрения функции.

**Отрицательный корень**

- Также существует еще один корень уравнения, который находится в отрицательной области. В этой области полином третьей степени стремительно уходит в отрицательную бесконечность. Однако, учитывая свойства показательной функции, можно сделать вывод о том, что где-то существует и отрицательный корень, но в данной ситуации поиск его не представляется необходимым

## Реализация методов

### Метод Хорд

Это 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.

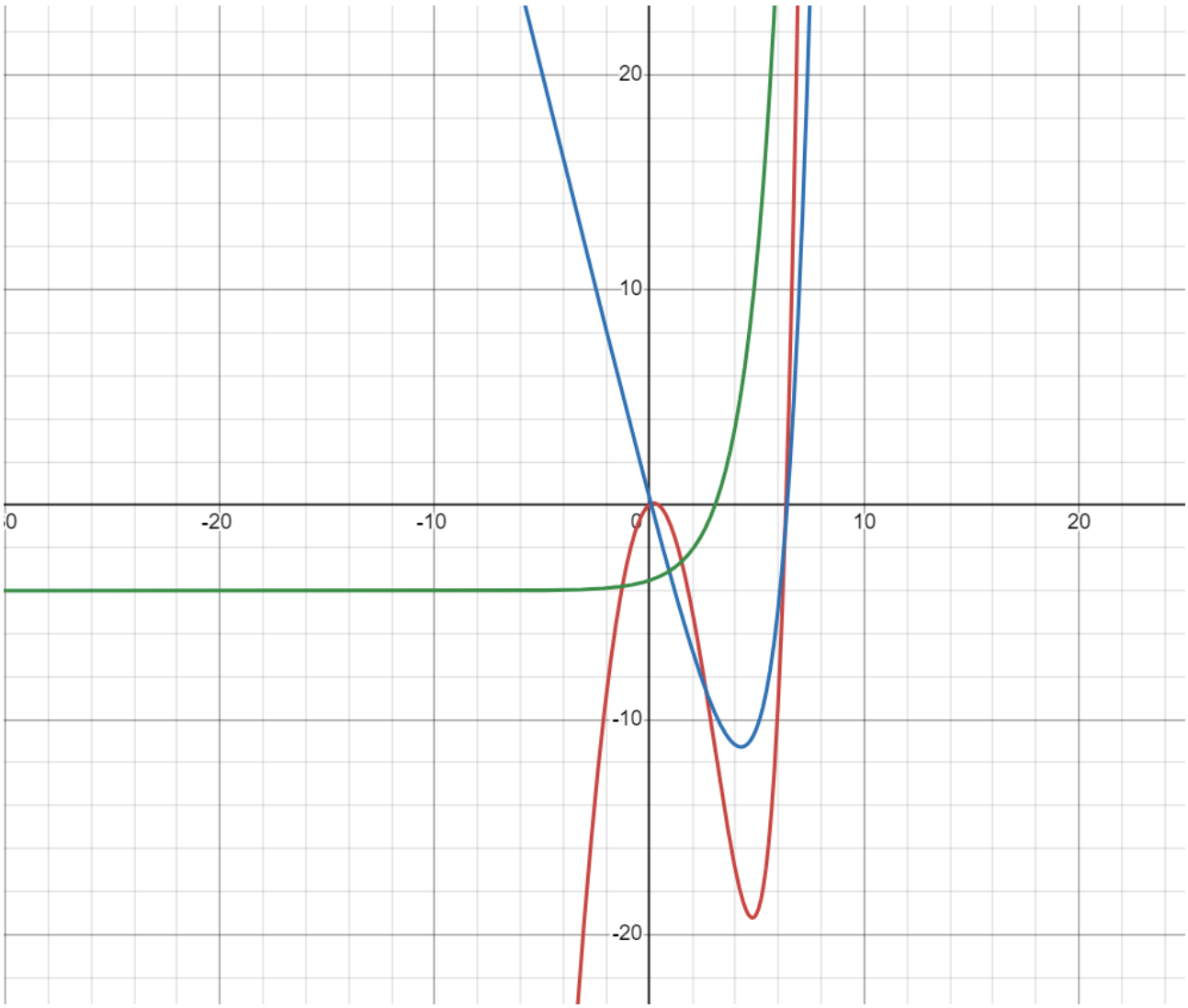


Рис. 1: График функции и ее первой и второй производной

```python
def chord_method(eps, lst: list):
    table = PrettyTable()
    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
        table.add_row([counter, x_next])

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

**Обозначения**

- eps - значение погрешности, используемое для остановки итераций (то есть критерий остановки

метода).

- lst - список значений, представляющих начальный отрезок для метода.
- counter - переменная, отвечающая за отслеживание количества итераций.
- b - второй элемент списка lst.
- x_prev - первый элемент списка lst.
- x_next - вычисляемое следующее приближение корня уравнения

**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.

**Таблица значений**

| Итерация | Значение | Итерация | Значение |
|---|---|---|---|
| 1 | 0.322541193046703 | 21 | 0.3991525127269086 |
| 2 | 0.3409645084710158 | 22 | 0.39918859088335995 |
| 3 | 0.3555675656565128 | 23 | 0.39921452106708516 |
| 4 | 0.36686352496279695 | 24 | 0.3992331568543878 |
| 5 | 0.3754373981027205 | 25 | 0.3992465498018522 |
| 6 | 0.3818520031969649 | 26 | 0.39925617466842017 |
| 7 | 0.3865995358189016 | 27 | 0.39926309148370676 |
| 8 | 0.39008519745005094 | 28 | 0.39926806212635957 |
| 9 | 0.3926293522424672 | 29 | 0.3992716341575618 |
| 10 | 0.3944783323328586 | 30 | 0.3992742010951745 |
| 11 | 0.3958178893602863 | 31 | 0.3992760457431182 |
| 12 | 0.3967861773641209 | 32 | 0.3992773713364518 |
| 13 | 0.39748494883616114 | 33 | 0.3992783239267496 |
| 14 | 0.3979886248743356 | 34 | 0.39927900847063347 |
| 15 | 0.3983513657463251 | 35 | 0.39927950039227667 |
| 16 | 0.398612446298662 | 36 | 0.3992798538929168 |
| 17 | 0.39880027425661835 | 37 | 0.3992801079224546 |
| 18 | 0.3989353593985572 | 38 | 0.3992802904708603 |
| 19 | 0.39903248985084655 | 39 | 0.3992804216521015 |
| 20 | 0.3991023181845276 | 40 | 0.3992805159203268 |

Таблица 1: Значения итераций метода Хорд

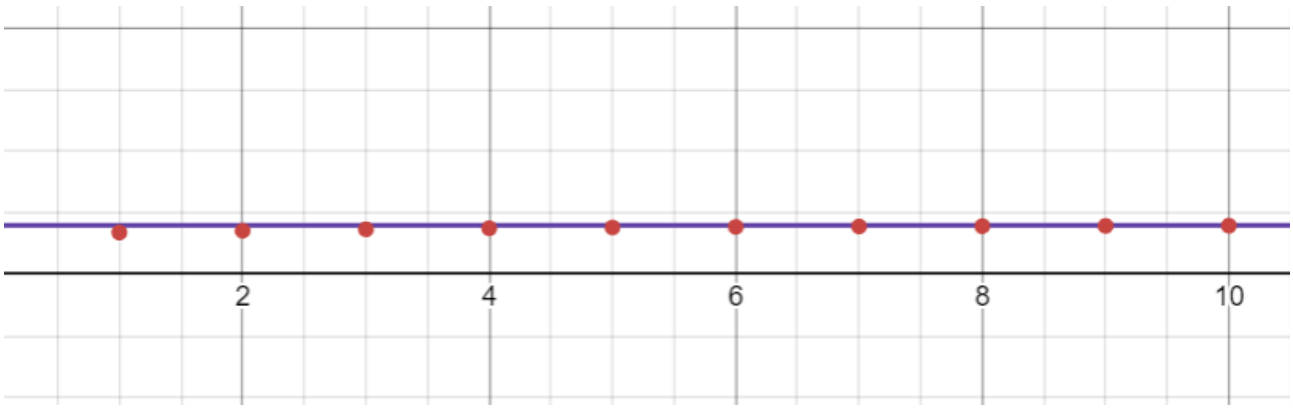


Рис. 2: График значений

- Как можно заметить из представленного графика - метод сходится к 9 итерации

## Метод Ньютона

```python
def newton_method(eps, x0):
    table = PrettyTable()
    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)
        table.add_row([count, x_next])
    print(table.get_string(border=True, header=False, hrules=1))
    return x_next
```

### Обозначения

- eps - значение погрешности, используемое для остановки итераций (критерий остановки метода).
- x0 - начальное приближение для корня.
- count - переменная, отслеживающая количество итераций.
- x_prev - предыдущее приближение.
- x_next - вычисляемое следующее приближение корня уравнения на основе предыдущего значения и значений функции и её производной.

### Алгоритм

1. Выбирается начальное приближение $x_0$.
2. Вычисляется следующее приближение по формуле:

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

3. Процесс повторяется до тех пор, пока разница между последовательными приближениями не станет меньше заданной точности $\epsilon$.

### Значения

| Итерация | Значение |
|---|---|
| 1 | 0.8386350669976804 |
| 2 | 0.5463059242914783 |
| 3 | 0.42988681718031907 |
| 4 | 0.401281869581121 |
| 5 | 0.39929052760214123 |
| 6 | 0.3992807568969161 |
| 7 | 0.3992807566616375 |

Таблица 2: Значения итераций метода Ньютона

**Сходимость**

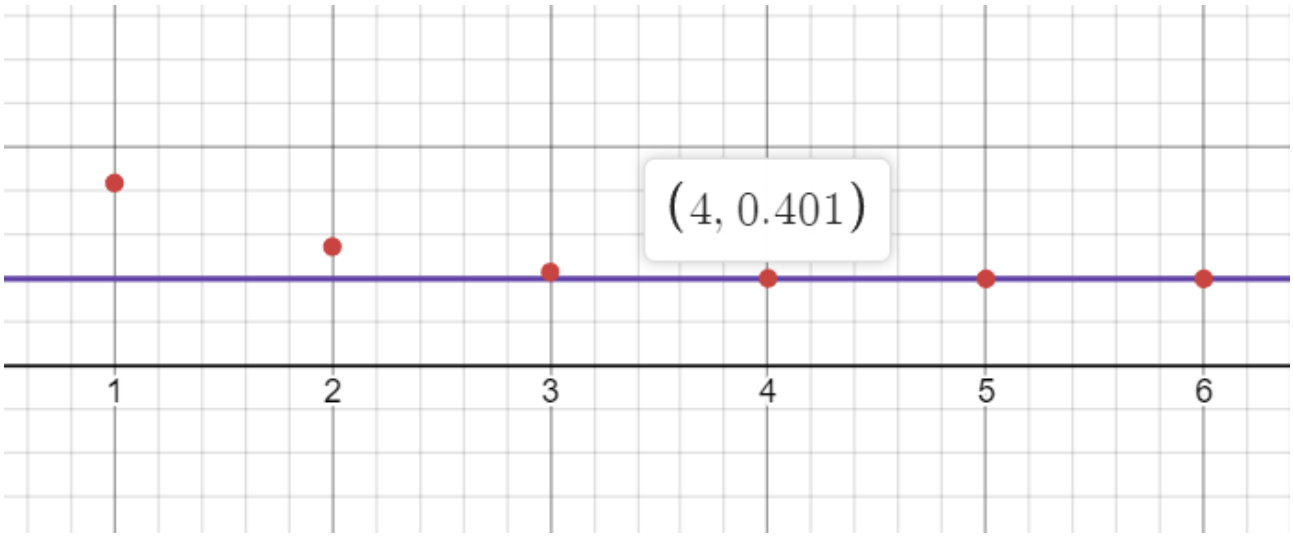


Рис. 3: График сходимости

- Как можно заметить, метод Ньютона сходится к 4 итерации.

**Метод Бисекции** Метод бисекции (или метод деления пополам) — это 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).

```
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
```

**Обозначения**

- ps - значение погрешности, используемое для остановки итераций (то есть критерий остановки метода).
- lst - список значений, представляющих начальный отрезок для метода.
- l - левый конец начального отрезка.
- r - правый конец начального отрезка.
- c - середина отрезка.
- count - переменная, отвечающая за отслеживание количества итераций.
- x_prev и x_next - переменные для хранения предыдущего и следующего приближенных значений корня.

**Алгоритм**

1. Выбирается начальный интервал $[a, b]$, в котором функция меняет знак.
2. Вычисляется середина интервала $c = \frac{a+b}{2}$.
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 $\epsilon$.

**Значения**

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

Таблица 3: Значения итераций метода Бисекций

**Сходимость**

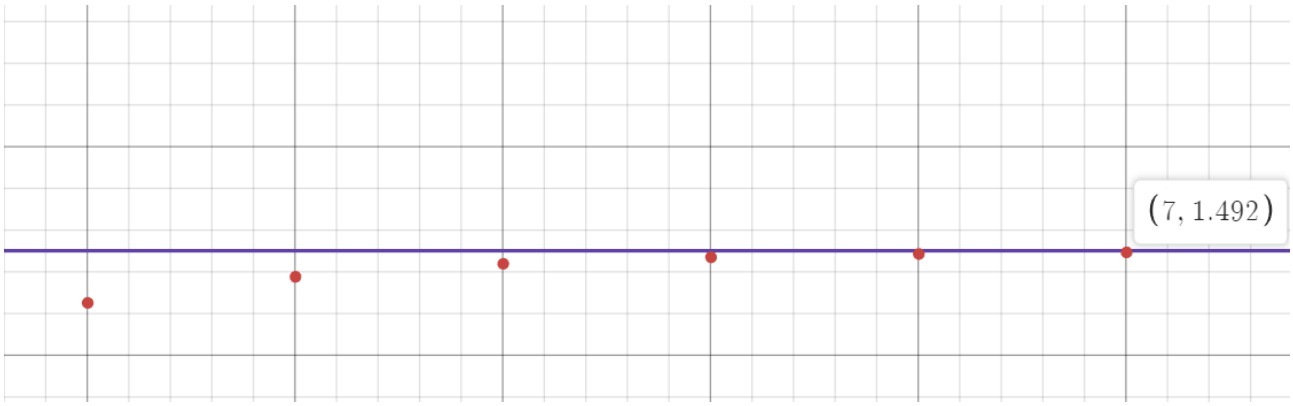


Рис. 4: График сходимости

- Исходя из приведенного графика можно сделать вывод о том, что метод бисекции сходится к 7 итерации

## Заключение

В результате проделанной работы можно сделать ряд выводов:

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 $f(x) = 2^x - 2x^2 - 1$ 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.