Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Методы численного анализа

# ОТЧЁТ

к лабораторной работе
на тему

Численное решение нелинейных уравнений

Выполнил: студент группы 253502
Богданов Александр Сергеевич

Проверил: Анисимов Владимир Яковлевич

Минск 2022

## Содержание

Вариант 3

## Цели выполнения задания

1) Изучить методы численного решения нелинейных уравнений (метод бисекции, метод хорд, метод Ньютона)

2) Исследовать скорость сходимости итерационных процедур

3) Составить программу численного решения нелинейных уравнений методами бисекции, хорд, Ньютона

4) Проверить правильность работы программы на тестовых примерах

5) Численно решить нелинейное уравнение заданного варианта

6) Сравнить число итераций, необходимого для достижения заданной точности вычисления разными методами

# Краткие теоритические сведения

Численное решение нелинейного уравнения *f(x)=0* заключается в вычислении с заданной точностью значения всех или некоторых корней уравнения и распадается на несколько задач: *во-первых,* надо исследовать количество и характер корней (вещественные или комплексные, простые или кратные), *во-вторых,* определить их приближенное расположение, т.е. значения начала и конца отрезка, на котором лежит только один корень, *в-третьих,* выбрать интересующие нас корни и вычислить их с требуемой точностью. Вторая задача называется ***отделением корней***. Решив ее, по сути дела, находят приближенные значения корней с погрешностью, не превосходящей длины отрезка, содержащего корень. Отметим два простых приема отделения действительных корней уравнения - *табличный* и *графический.* Первый прием состоит в вычислении таблицы значений функции *f(x)* в заданных точках $x_t$ и использовании следующих теорем математического анализа:

1. *Если функция y=f(x) непрерывна на отрезке [a,b] и f(a)f(b)<0, то внутри отрезка [a,b] существует по крайней мере один корень уравнения f(x)=0.*
2. *Если функция y=f(x) непрерывна на отрезке [a,b], f(a)f(b) < 0 и f '(x) на интервале (a,b) сохраняет знак, то внутри отрезка [a,b] существует единственный корень уравнения f(x)=0.*

Таким образом, если при некотором *k* числа $f(x_k)$ и $f(x_{k+1})$ имеют разные знаки, то это означает, что на интервале $(x_k,x_k+_1)$ уравнение имеет по крайней мере один действительный корень нечетной кратности (точнее – нечетное число корней). Выявить по таблице корень четной кратности очень сложно.

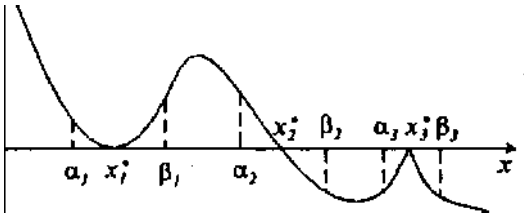


Рис. 1

На рис.1 представлены три наиболее часто встречающиеся ситуации:

а) кратный корень: $f'(x^*)=0,\ f(a_1)^* f(b_1) > 0;$

б) простой корень: $f'(x^*)=0,\ f(a_2)^* f(b_2) < 0;$

в) вырожденный корень: $f'(x^*)$ не существует, $f(a_3)^* f(b_3)>0.$

Как видно из рис.1, в первых двух случаях значение корня совпадает с точкой экстремума функции и для нахождения таких корней рекомендуется использовать методы поиска минимума функции.

Для определения числа корней на заданном промежутке используется Теорема Штурма: Если *f(x)* является многочленом и уравнение *f(x)=0* не имеет кратных корней на промежутке [a, b], то число корней этого уравнения, лежащих на таком промежутке, совпадает с числом *N(a)* – *N(b)*, где функция *N* определяется следующим образом.

Строим ряд Штурма $f_0(x)$, $f_1(x)$, $f_2(x)$, …, $f_m(x)$, где

$f_0(x) = f(x)$,

$f_1(x) = f'(x)$,

$f_i(x)$ = *остаток от деления* $f_{i-2}(x)$ *на* $f_{i-1}(x)$, *взятый с обратным знаком*

Функция $N(x)$ определяется как число перемен знака в ряде Штурма, если подставить в функции ряда значение $x$

Для отделения корней можно использовать график функции $y=f(x)$. Корнями уравнения являются те значения $x$, при которых график функции пересекает ось абсцисс. Построение графика функции даже с малой точностью обычно дает представление о расположении и характере корней уравнения (иногда позволяет выявить даже корни четной кратности). Если построение графика функции $y=f(x)$ вызывает затруднение, следует преобразовать исходное уравнение к виду $\varphi_1(x)=\varphi_2(x)$ таким образом, чтобы графики функций $y=\varphi_1(x)$ и $y=\varphi_2(x)$ были достаточно просты. Абсциссы точек пересечения этих графиков и будут корнями уравнения.

Допустим, что искомый корень уравнения отделен, т.е. найден отрезок $[a, b]$, на котором имеется только один корень уравнения. Для вычисления корня с требуемой точностью ε обычно применяют какую-либо итерационную процедуру ***уточнения корня***, строящую числовую последовательность значений $x_n$, сходящуюся к искомому корню уравнения. Начальное приближение $x_0$ выбирают на отрезке [a, b], продолжают вычисления, пока не выполнится неравенство $|x_{n-1} - x_n| < \varepsilon$, и считают, что $x_n$ есть корень уравнения, найденный с заданной точностью. Имеется множество различных методов построения таких последовательностей и выбор алгоритма – весьма важный момент при практическом решении задачи. Немалую роль при этом играют такие свойства метода, как простота, надежность, экономичность, важнейшей характеристикой является его *скорость сходимости.* Последовательность $x_n$, сходящаяся к пределу $x^*$, имеет скорость сходимости порядка α, если при $n \to \infty$: $\left|x_{n+1} - x^{\square}\right| = O\left(\left|x_n - x^{\square}\right|^{\alpha}\right)$. При α=1 сходимость называется линейной, при 1<α<2 – сверхлинейной, при α=2 – квадратичной. С ростом α алгоритм, как правило, усложняется и условия сходимости становятся более жесткими. Рассмотрим наиболее распространенные итерационные методы уточнения корня.

**Метод простых итераций**. Вначале уравнение $f(x)=0$ преобразуется к эквивалентному уравнению вида $x=\varphi(x)$. Это можно сделать многими способами, например, положив $\varphi(x)=x+\blacklozenge(x)f(x)$, где $\blacklozenge(x)$ – произвольная непрерывная знакопостоянная функция. Выбираем некоторое начальное приближение $x_0$ и вычисляем дальнейшие приближения по формуле

$$x_k = \varphi(x_{k-1}),\ k=1,2,\ \ldots$$

Метод простых итераций не всегда обеспечивает сходимость к корню уравнения. Достаточным условием сходимости этого метода является выполнение неравенства $\varphi'(x) \vee \le q<1$ на отрезке, содержащем корень и все приближения $x_n$. Метод имеет линейную скорость сходимости и справедливы следующие оценки:

$$\left|x_n - x^{\square\square}\right| < \frac{q}{1-q}\left|x_n - x_{n-1}\right|, если\, \varphi'(x) > 0,$$

$$\left|x_n - x^{\square\square}\right| < \left|x_n - x_{n-1}\right|, если\, \varphi'(x) < 0.$$

Метод имеет простую геометрическую интерпретацию: нахождение корня уравнения *f(x)=0* равносильно обнаружению неподвижной точки функции *x= φ(x),* т.е. точки пересечения графиков функций *y= φ(x)* и *y=x.* Если производная *φ'(x)<0,* то последовательные приближения колеблются около корня, если же производная *φ'(x)>0,* то последовательные приближения сходятся к корню монотонно.

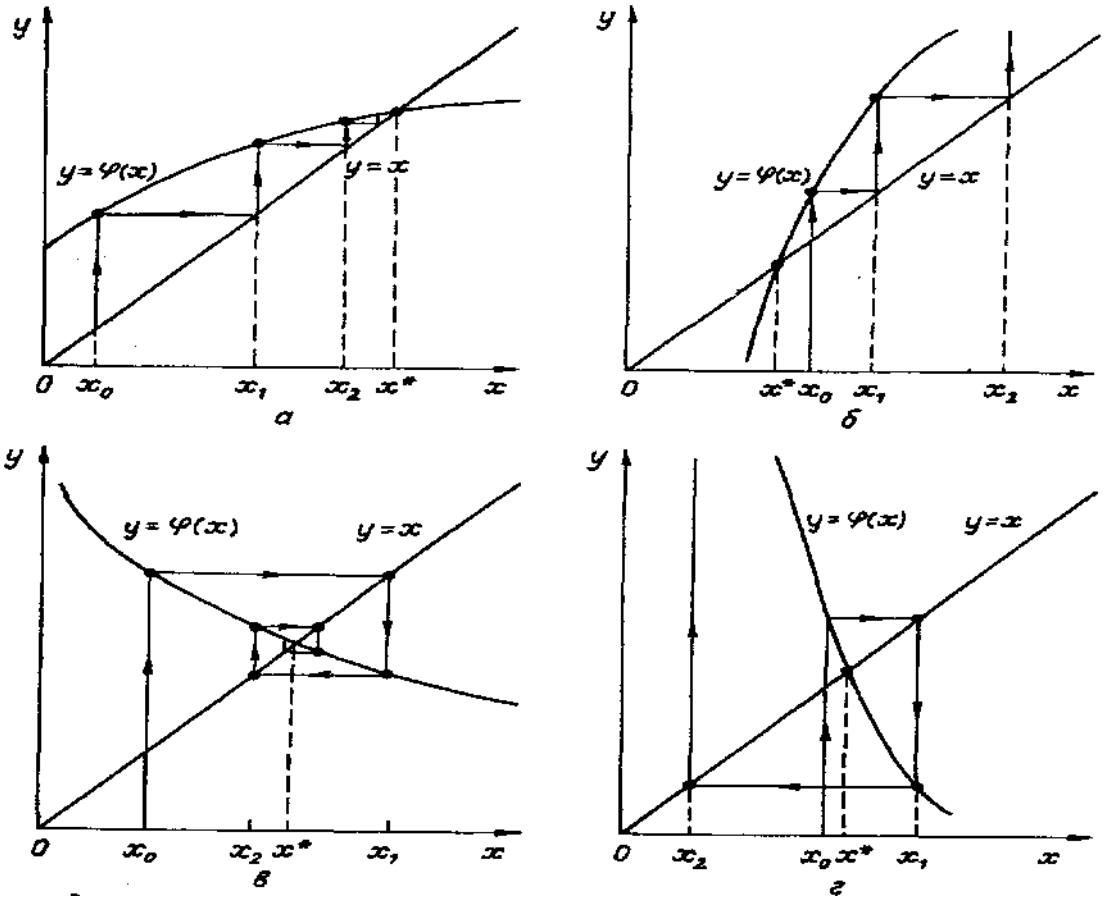


Рис. 2. Метод простых итераций: а - односторонний сходящийся процесс; б - односторонний расходящийся процесс; в - двухсторонний сходящийся процесс; г - двухсторонний расходящийся процесс

Рассмотрим процесс графически (рис. 2). Из графиков видно, что при *φ'(x)<0* и при *φ'(x)>0* возможны как сходящиеся, так и расходящиеся итерационные процессы. Скорость сходимости зависит от абсолютной величины производной *φ(x).* Чем меньше */φ'(x)/* вблизи корня, тем быстрее сходится процесс.

**Метод хорд**. Пусть дано уравнение *f(x)* = 0, $a \le x \le b$, где *f(x)* – дважды непрерывно дифференцируемая функция. Пусть выполняется условие *f(a)*f(b)*<0 и проведено отделение корней, то есть на данном интервале *(a,*

*b)* находится один корень уравнения. При этом, не ограничивая общности, можно считать, что *f(b)>0.*

Пусть функция *f* выпукла на интервале *(a, b)* (см. рис. 3).

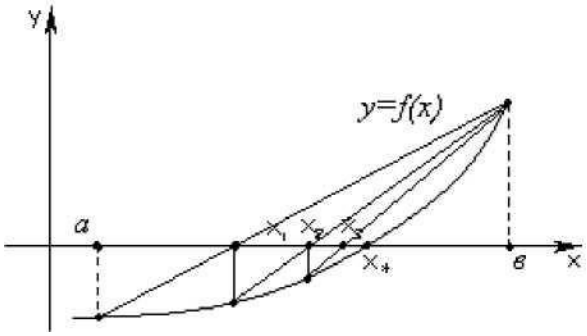


Рис. 3

Заменим график функции хордой (прямой), проходящей через точки $M_0(a, f(a))$ и $M_1(b, f(b))$. Уравнение прямой, проходящей через две заданные точки, можно записать в виде $\frac{y-y_1}{y_2-y_1}=\frac{x-x_1}{x_2-x_1}$. В нашем случае получим: $\frac{y-f(a)}{f(b)-f(a)}=\frac{x-a}{b-a}$. Найдем точку пересечения хорды с осью Ox. Полагая $y = 0$, получаем из предыдущего уравнения: $x_1=a-\frac{f(a)}{f(b)-f(a)}*(b-a)$. Теперь возьмем интервал $(x_1,b)$ в качестве исходного и повторим вышеописанную процедуру (см. рис. 3). Получим $x_2=x_1-\frac{f(x_1)}{f(b)-f(x_1)}*(b-x_1)$. Продолжим процесс. Каждое последующее приближение вычисляется по рекуррентной

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(b)-f(x_{n-1})}\cdot(b-x_{n-1}) \quad n=1,2,\ldots \ ,$$

формуле $x_0 = a$ . (3.1)

Если же функция вогнута (см. рис. 4),

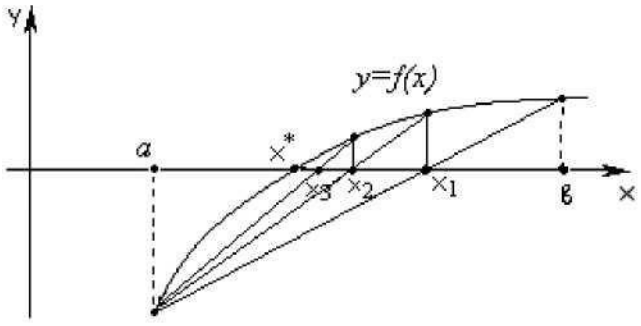


Рис. 4

уравнение прямой соединяющей точки $M_0(a, f(a))$ и $M_1(b, f(b))$ запишем в виде $\frac{y-f(b)}{f(a)-f(b)}=\frac{x-b}{a-b}$. Найдем точку пересечения хорды с осью Ox:

$x_1 = b - \frac{f(b)}{f(a)-f(b)} * (a-b)$. Теперь возьмем интервал *(a, $x_1$)* в качестве исходного и найдем точки пересечения хорды, соединяющей точки *(a, f(a))* и *($x_1$, f($x_1$))*, с осью абсцисс (см. рис. 4). Получим $x_2 = x_1 - \frac{f(x_1)}{f(a)-f(x_1)} * (a-x_1)$. Повторяя данную процедуру, получаем рекуррентную формулу:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(a)-f(x_{n-1})} \cdot (a - x_{n-1}) \quad n = 1,2,..$$

$x_0 = b$ .

(3.2)

Описанный выше метод построения рекуррентных последовательностей (3.1) и (3.2) называется методом хорд. Для использования метода хорд нужно было бы предварительно найти точки перегиба и выделить участки, на которых функция не меняет характер выпуклости. Однако на практике поступают проще: в случае $f(b)f''(b)>0$ для построения рекуррентной последовательности применяются формулы (3.1), а в случае, когда $f(a)f''(a)>0$, применяют формулы (3.2).

**Метод Ньютона** (**касательных**). Для начала вычислений требуется задание одного начального приближения *$x_0$,* последующие приближения вычисляются по формуле

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

Метод имеет квадратичную скорость сходимости для простого корня, но очень чувствителен к выбору начального приближения. При произвольном начальном приближении итерации сходятся, если всюду $|f(x)f''(x)|<(f'(x))^2$, в противном случае сходимость будет только при *$x_0$,* достаточно близком к корню. Существует несколько достаточных условий сходимости. Если производные $f'(x)$ и $f''(x)$ сохраняют знак в окрестности корня, рекомендуется выбирать $x_0$ так, чтобы $f(x)f''(x)>0$. Если, кроме этого, для отрезка *[a,b],* содержащего корень, выполняются условия $\left|\frac{f(a)}{f'(a)}\right|<b-a, \left|\frac{f(b)}{f'(b)}\right|<b-a$, то метод сходится для любых $a \leq x_0 \leq b.$

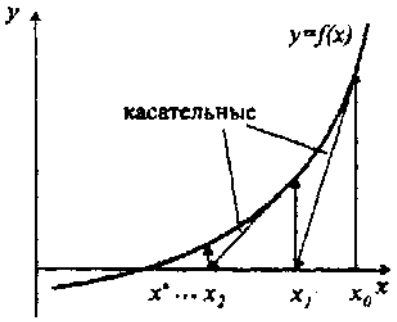


Рис. 5

Метод Ньютона получил также второе название *метод касательных* благодаря геометрической иллюстрации его сходимости, представленной на рис. 5. Метод Ньютона позволяет находить как простые, так и кратные корни. Основной его недостаток – малая область ходимости и необходимость вычисления производной

## 6.4. Комбинированный метод хорд и касательных

Пусть для определенности $f(b)f''(b) > 0$ ( рис. 6.7).

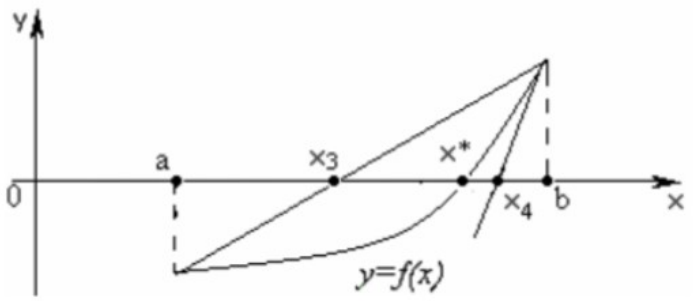


Рис. 6.7.

Тогда, применяя метод хорд при $x_1 = a$, получим

$$x_3 = x_1 - \frac{f(x_1)}{f(b) - f(x_1)}(b - x_1).$$

Применяя метод Ньютона при $x_2 = b$, получим

$$x_4 = x_2 - \frac{f(x_2)}{f'(x_2)}.$$

При этом $x^* \in (x_3, x_4)$.

Отсюда

$$x_{2n+1} = x_{2n-1} - \frac{f(x_{2n-1})}{f(x_{2n}) - f(x_{2n-1})}(x_{2n} - x_{2n-1}), \qquad n = 1,\dots .$$

$$x_{2n+2} = x_{2n} - \frac{f(x_{2n})}{f'(x_{2n})}(x_{2n} - x_{2n-1}), \qquad n = 1,\dots ,$$

причем всегда $x^* \in (x_{2n+1}, x_{2n+2})$.

Пусть теперь $f(a)f''(a) > 0$ (рис. 6.8).

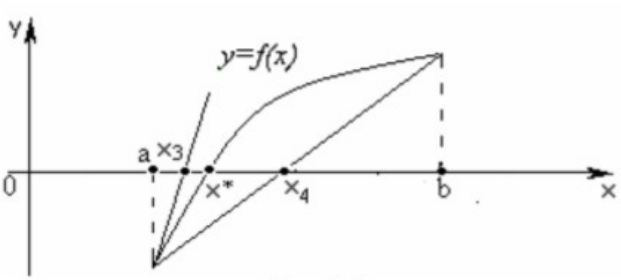


Рис. 6.8.

Применяем метод Ньютона при $x_1 = a$ и метод хорд при $x_2 = b$, и получаем

$$\begin{cases} x_{2n+1} = x_{2n-1} - \dfrac{f(x_{2n-1})}{f'(x_{2n-1})}, \ n = 1, 2... \\ x_{2n+2} = x_{2n} - \dfrac{f(x_{2n})}{f(a) - f(x_{2n})}(a - x_{2n}), \ n = 1, 2..., \end{cases}$$

причем $x^* \in (x_{2n+1}, x_{2n+2})$.

Таким образом, комбинированный метод хорд и касательных удобен тем, что корень уравнения всегда находится в интервале между двумя последовательными приближениями.

**ЗАДАНИЕ.**

1) Используя теорему Штурма определить число корней уравнения:

   $\mathbf{x^3+ax^2+bx+c=0}$ на отрезке [-10,10]. Значения коэффициентов уравнения взять из таблицы.

2) Отделить все корни, лежащие на данном отрезке.

3) Вычислить наименьший из корней сначала методом половинного деления, а за затем методом хорд и методом Ньютона. Сравнить число необходимых итераций в обоих методах. Точность до 0.0001.

| | | | |
|---|---|---|---|
| -19,7997 | 28,9378 | 562,833 | 3 |

Полученные результаты будем сверять с решением, полученным используя подмодуль numpy:

```
correct_roots = np.roots([1, task.a, task.b, task.c])
```

**Программная реализация.**

```
def get_clean_intervals(ls, a):
    traverse_list(ls, a)
    return a


def traverse_list(ls, a):
    if type(ls) == tuple:
        if ls[0] != 0 or ls[1] != 0:
            a.append(ls)
        return
    for i in range(len(ls)):
        traverse_list(ls[i], a)
```

```
def get_sturm_row(equation):
    sturm_row = list()
    y = equation
    sturm_row.append(y)

    y_2 = diff(y)
    sturm_row.append(y_2)

    r = y_2
    while degree(r) > 0:
        r = div(y, y_2)[1] * (-1)  # get remain
        sturm_row.append(r)
        y = y_2
        y_2 = r
    return sturm_row
```

```python
def get_root_intervals(sturm_row, interval: tuple):
    root_intervals = list()
    digit_change_times = dict()
    for i in range(interval[0], interval[1] + 1):
        func_values = list()
        digit_changes_count = 0
        for num, func in enumerate(sturm_row):
            func_values.append(func.subs('x', i))
            if num == 0:
                continue
            if func_values[num] * func_values[num - 1] < 0:
                digit_changes_count += 1

        digit_change_times[i] = digit_changes_count

    for i in range(interval[0] + 1, interval[1] + 1):

        if digit_change_times[i - 1] - digit_change_times[i] > 1:
            result_intervals = get_root_small_interval(sturm_row, (i - 1, i))
            clean_upped_intervals = []
            clean_upped_intervals = get_clean_intervals(result_intervals, clean_upped_intervals)

            for inter in clean_upped_intervals:
                root_intervals.append(list(inter))

        elif digit_change_times[i - 1] != digit_change_times[i]:
            root_intervals.append([i - 1, i])

    for i in range(1, len(root_intervals)):
        if root_intervals[i][0] == root_intervals[i - 1][1]:
            root_intervals[i][0] = root_intervals[i][0] + 0.00001

    return root_intervals

def get_root_small_interval(sturm_row, interval: tuple):
    result_intervals = []
    digit_changes = [0, 0]

    for inter_num, inter in enumerate(interval):
        func_values = list()
        prev_val = 0
        current_val = 0
        for i, func in enumerate(sturm_row):
            func_values.append(func.subs('x', inter))

            if i == 0:
                prev_val = func_values[i]
                continue
            current_val = func_values[i]

            if prev_val == 0:
                prev_val = current_val
                continue
            if current_val == 0:
                continue

            if prev_val * current_val < 0:
                digit_changes[inter_num] += 1
            prev_val = current_val

    if digit_changes[0] == digit_changes[1]:
        return 0, 0

    if abs(digit_changes[1] - digit_changes[0]) == 1:
        return interval[0], interval[1]
    else:
        middle = float(interval[1] - interval[0]) / 2
        result_intervals.append(get_root_small_interval(sturm_row, (interval[0], interval[0] + middle)))
        result_intervals.append(get_root_small_interval(sturm_row, (interval[0] + middle, interval[1])))

    return result_intervals
```

```python
def bisection(function, interval: tuple, tol=0.0001):
    x, y = symbols('x y')
    a, b = interval[0], interval[1]
    mid = (a + b) / 2

    f = lambdify(x, function, "numpy")

    if f(a) * f(b) > 0:
        raise Exception(f"Bisection method couldn't solve f(x) = 0, "
                        f"because f(a) * f(b) = {f(a) * f(b)} >= 0\n"
                        f"--> more than 1 root on ({(a, b)}) or no roots)")
    x_delta = tol * 2
    iteration = 1
    while f(mid) > tol or x_delta > tol:
        if f(a) == 0:
            return a, iteration
        elif f(b) == 0:
            return b, iteration
        elif f(mid) == 0:
            return mid, iteration

        if f(a) * f(mid) < 0:
            b = mid
        elif f(b) * f(mid) < 0:
            a = mid

        x_delta = mid
        mid = (a + b) / 2

        x_delta = abs(x_delta - mid)
        iteration += 1

    return mid, iteration

def secant(function, interval: tuple, tol=0.0001):
    x, y = symbols('x y')
    a, b = interval[0], interval[1]
    f = lambdify(x, function, "numpy")
    if f(a) * f(b) > 0:
        raise Exception(f"Secant method couldn't solve f(x) = 0, "
                        f"because f(a) * f(b) = {f(a) * f(b)} >= 0\n"
                        f"--> more than 1 root on ({(a, b)}) or no roots)")

    x_sec = a - (f(a) * (b - a)) / (f(b) - f(a))

    x_delta = 0
    iteration = 1
    while abs(f(x_sec)) > tol or x_delta > tol:
        if f(a) == 0:
            return a, iteration
        elif f(b) == 0:
            return b, iteration
        elif f(x_sec) == 0:
            return x_sec, iteration

        if f(x_sec) * f(a) < 0:
            b = x_sec
        elif f(x_sec) * f(b) < 0:
            a = x_sec

        x_delta = x_sec
        x_sec = a - (f(a) * (b - a)) / (f(b) - f(a))

        x_delta = abs(x_delta - x_sec)
        iteration += 1

    return x_sec, iteration
```

```python
def newton(function, interval, tol=0.0001):
    x, y = symbols('x y')
    a, b = interval[0], interval[1]
    f = lambdify(x, function, 'numpy')

    if f(a) * f(b) > 0:
        raise Exception(f"Newton method couldn't solve f(x) = 0, "
                        f"because f(a) * f(b) = {f(a) * f(b)} >= 0\n"
                        f"--> more than 1 root on ({(a, b)}) or no roots)")

    x_start = (a + b) / 2
    y_diff = function.diff()
    f_diff = lambdify(x, y_diff, 'numpy')

    x_newton = x_start - f(x_start) / f_diff(x_start)

    x_delta = tol * 2
    iteration = 1
    while abs(f(x_newton)) > tol or x_delta > tol:
        if f(x_newton) == 0:
            return x_newton, iteration

        x_delta = x_newton
        x_newton = x_newton - f(x_newton) / f_diff(x_newton)
        x_delta = abs(x_delta - x_newton)
        iteration += 1

    return x_newton, iteration
```

# Полученные результаты

## Решение задания:

```
Equation is
 y=x**3 - 19.7997*x**2 + 28.9378*x + 562.833
#####################################################

Intervals with 1 root: [[-5, -4], [8, 9]]


        Current interval is: [-5, -4]
#####################################################
**** Solving with Bisection method ****
ROOT: -4.27166748046875
Iteration: 14
#####################################################
**** Solving with Secant method ****
ROOT: -4.271614570656526
Iteration: 6
#####################################################
**** Solving with Newton method ****
ROOT: -4.271614697123747
Iteration: 3

        Current interval is: [8, 9]
#####################################################
**** Solving with Bisection method ****
ROOT: 8.416839599609375
Iteration: 15
#####################################################
**** Solving with Secant method ****
ROOT: 8.416836414497405
Iteration: 4
#####################################################
**** Solving with Newton method ****
ROOT: 8.4168362459457
Iteration: 3
#####################################################
**** Solving with Numpy ****
ROOTS: [15.65447845  8.41683625 -4.2716147 ]
```

**Тестовый пример 1.**

```
Equation is
 y=x**2 - 0.9*x - 0.36
#####################################################

Intervals with 1 root: [[-1, 0], [1, 2]]


        Current interval is: [-1, 0]
#####################################################
**** Solving with Bisection method ****
ROOT: -0.29998779296875
Iteration: 14
#####################################################
**** Solving with Secant method ****
ROOT: -0.2999606066662447
Iteration: 8
#####################################################
**** Solving with Newton method ****
ROOT: -0.30000000000000204
Iteration: 4

        Current interval is: [1, 2]
#####################################################
**** Solving with Bisection method ****
ROOT: 1.20001220703125
Iteration: 14
#####################################################
**** Solving with Secant method ****
ROOT: 1.1999505620110387
Iteration: 8
#####################################################
**** Solving with Newton method ****
ROOT: 1.2000000000005318
Iteration: 4
```

**Тестовый пример 2.**

```
Equation is
 y=4*x**2 + 16*x + 16
#######################################################

Intervals with 1 root: [[-3, -2]]


        Current interval is: [-3, -2]
#######################################################
**** Solving with Bisection method ****
ROOT: -2
Iteration: 1
#######################################################
**** Solving with Secant method ****
ROOT: -2.0
Iteration: 1
#######################################################
**** Solving with Newton method ****
ROOT: -2.00006103515625
Iteration: 13
#######################################################
```

**Тестовый пример 3.**

```
Equation is
 y=x**4 - 2*x**3 - 7*x**2 + 8*x + 1
######################################################

Intervals with 1 root: [[-3, -2], [-1, 0], [1, 2], [3, 4]]


        Current interval is: [-3, -2]
######################################################
**** Solving with Bisection method ****
ROOT: -2.35003662109375
Iteration: 14
######################################################
**** Solving with Secant method ****
ROOT: -2.350082809843179
Iteration: 14
######################################################
**** Solving with Newton method ****
ROOT: -2.3500847963187756
Iteration: 4

        Current interval is: [-1, 0]
######################################################
**** Solving with Bisection method ****
ROOT: -0.11407470703125
Iteration: 14
######################################################
**** Solving with Secant method ****
ROOT: -0.11401153728200486
Iteration: 7
######################################################
**** Solving with Newton method ****
ROOT: -0.11401681881954807
Iteration: 4
```

```
        Current interval is: [1, 2]
#####################################################
**** Solving with Bisection method ****
ROOT: 1.11407470703125
Iteration: 14
#####################################################
**** Solving with Secant method ****
ROOT: 1.114011537282005
Iteration: 7
#####################################################
**** Solving with Newton method ****
ROOT: 1.114016818819548
Iteration: 4

        Current interval is: [3, 4]
#####################################################
**** Solving with Bisection method ****
ROOT: 3.35003662109375
Iteration: 14
#####################################################
**** Solving with Secant method ****
ROOT: 3.3500828098431787
Iteration: 14
#####################################################
**** Solving with Newton method ****
ROOT: 3.3500847963187756
Iteration: 4
#####################################################
**** Solving with Numpy ****
ROOTS: [15.65447845  8.41683625 -4.2716147 ]
```

## Оценка погрешности

Следующая функция производит проверку на допустимость погрешности.

```
def check_accuracy(w, x):
    for i in range(len(x)):
        if (abs(x[i] - w[i]) / abs(x[i])) >= 0.0001:
            return False
        else:
            print(f"|{x[i]} - {w[i]}| < 0.0001")
    return True
```

```
|-4.271614697123747 - -4.271614697120068| < 0.0001
|8.4168362459457 - 8.416836245945701| < 0.0001
True
```

# Выводы

Таким образом, в ходе выполнения лабораторной работы были изучены методы численного решения нелинейных уравнений (метод простой итерации, метод хорд, метод Биссекции, метод Ньютона), исследована скорость сходимости итерационных процедур, составлена программа численного решения нелинейных уравнений методами бисекции, хорд, Ньютона, проверена правильность работы программы на тестовых примерах, численно решено нелинейное уравнение заданного варианта, сравнены количества итераций, необходимых для достижения заданной точности вычисления разными методами.

Оптимальным способом численного решения нелинейных уравнений является применение метода Ньютона, так скорость сходимости в этом методе почти всегда квадратичная. В ходе работы были рассмотрены 4 функции, имеющие несколько корней на заданном промежутке (-10, 10), и в ходе решения результат был проверен с помощью встроенной функции, что дает понять, что реализованные методы успешно справляются с решением нелинейных уравнений.