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 3
на тему

Численное решение нелинейных уравнений

Выполнил: студент группы 053502
Герчик Артём Вадимович

Проверил: Анисимов Владимир Яковлевич

Минск 2022

Оглавление

Цели выполнения задания

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

2)Исследовать скорость сходимости итерационных процедур

3)Изучить метод Эйткена ускорения сходимости

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

5)Проверить правильность работы программы на тестовых примерах

6)Численно решить нелинейное уравнение заданного варианта

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

Краткие теоретические 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_i 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.

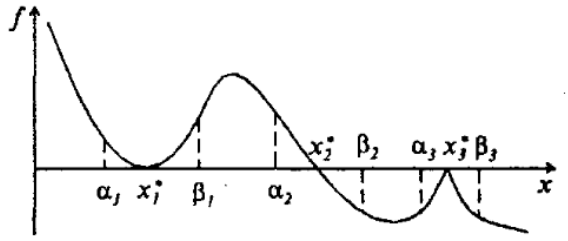

Рис. 5.1

На рис. 3.1 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. 3.1, 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) многочлен и уравнение не имеет кратных корней на промежутке[a, b], то число корней уравнения f (x) = 0, лежащих на промежутке [a, b], совпадает с числом N(a) – N(b), которое определяется из следующей процедуры.

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

$f_0(x) = f(x)$;

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

$f_0(x)$ делим на $f_1(x)$ и в качестве $f_2(x)$ берем остаток от деления, взятый с обратным знаком;

$f_1(x)$ делим на $f_2(x)$ и в качестве $f_3(x)$ берем остаток от деления, взятый с обратным знаком;

и т.д.

Полагаем N(a) – число перемен знака в ряде Штурма, если вместо x подставлена точка a, N(b) – число перемен знака в ряде Штурма, если вместо x подставлена точка b.

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 $n \to \infty$ $|x_{n+1} - x^*| = = O\left(|x_n - x^*|^{\alpha}\right)$. 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

$$x_k=\varphi(x_{k-1}), \quad k=0,1,\ldots$$

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

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

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

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

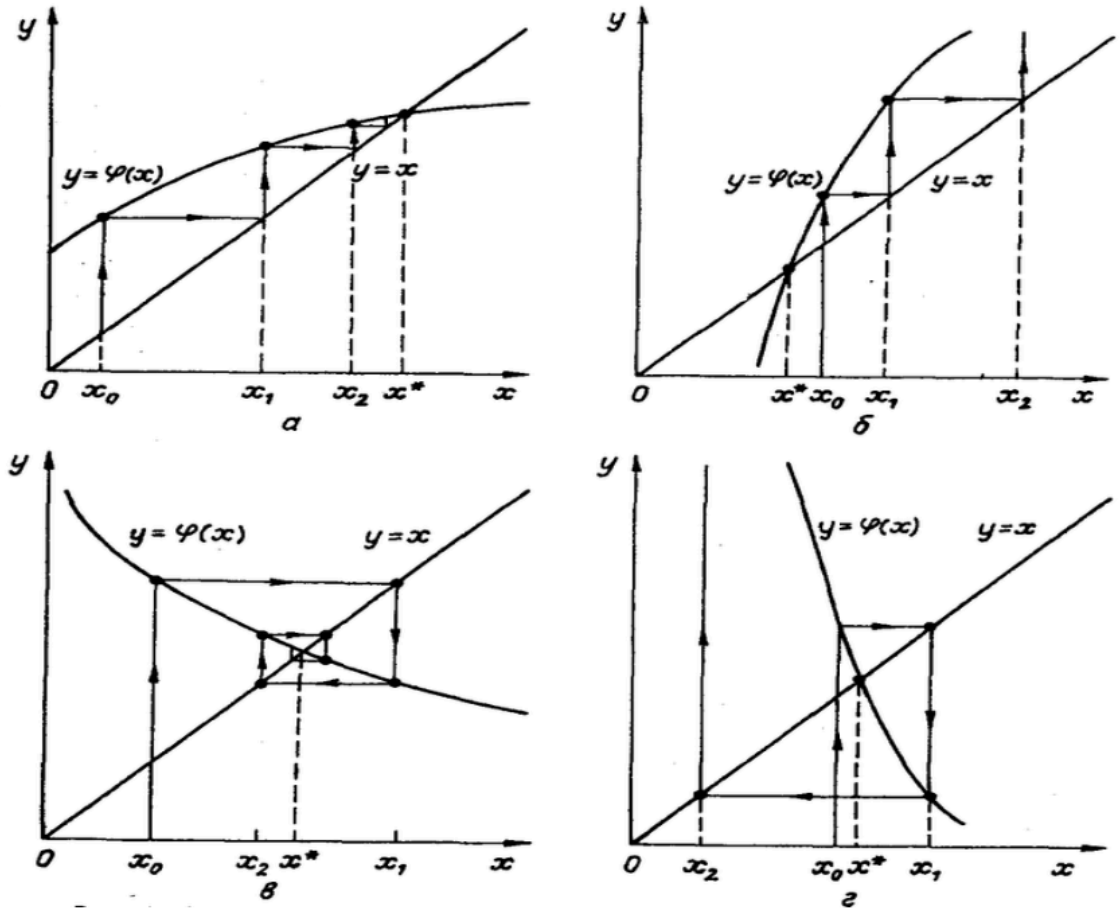

Рис. 3.2. Метод простых итераций: а - односторонний сходящийся процесс; б - односторонний расходящийся процесс; в - двухсторонний сходящийся процесс; *г*- двухсторонний расходящийся процесс

Рассмотрим процесс графически (рис. 3.2). Из графиков видно, что при $\phi'(x) > 0$ и при $\phi'(x) > 0$ возможны как сходящиеся, так и расходящиеся итерационные процессы. Скорость сходимости зависит от абсолютной величины производной $\phi(x)$. Чем меньше $|\phi'(x)|$ 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 $f(a)\cdot f(b)<0$ 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. 3.3).

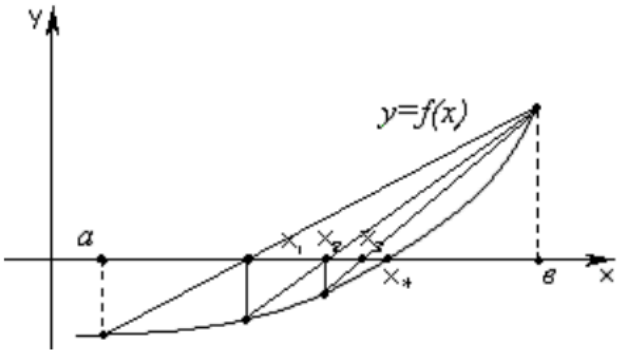

Рис. 3.3

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 Ох.

Полагая $y=0$, получаем из предыдущего уравнения:

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

Теперь возьмем интервал (x_1,b) в качестве исходного и повторим вышеописанную процедуру (см. рис. 6.3). Получим

$$x_2 = x_1 - \frac{f(x_1)}{f(b) - f(x_1)} \cdot (b - x_1) .$$

Продолжим процесс. Каждое последующее приближение вычисляется по рекуррентной формуле

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(b) - f(x_{n-1})} \cdot (b - x_{n-1}) \quad n = 1,2,... \ , \qquad (3.1)$$

$x_0 = a$.

Если же функция вогнута (см. рис. 3.4),

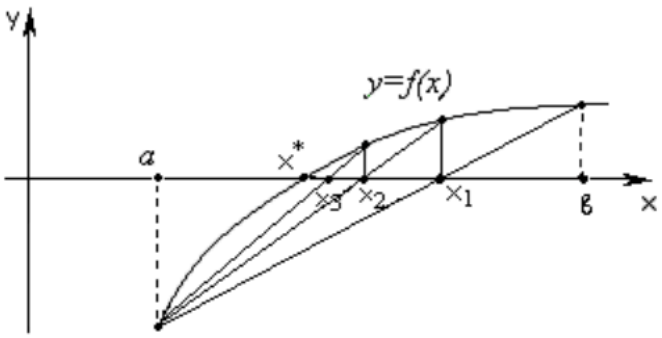

Рис. 3.4

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:

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

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. 3.4). Получим

$$x_2 = x_1 - \frac{f(x_1)}{f(a) - f(x_1)} \cdot (a - x_1).$$

Повторяя данную процедуру, получаем рекуррентную формулу:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f(a) - f(x_{n-1})} \cdot (a - x_{n-1}) \quad n = 1,2,.. \qquad (3.2)$$

$$x_0 = b \ .$$

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

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

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 $f(x_0)f''(x_0) > 0$. Если, кроме этого, для отрезка $[a,b]$, содержащего корень, выполняются условия

$$\left|\frac{f(a)}{f'(a)}\right| < b-a, \quad \left|\frac{f(b)}{f'(b)}\right| < b-a,$$

то метод сходится для любых $a \leq x_0 \leq b$.

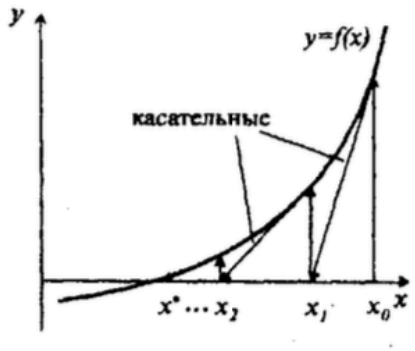

Рис. 5.4

Метод Ньютона получил также второе название *метод касательных* благодаря геометрической иллюстрации его сходимости, представленной на рис. 3.4

Метод Ньютона позволяет находить как простые, так и кратные корни. сновной его недостаток - малая область сходимости и необходимость вычисления производной.

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.

Программная реализация

```
import numpy

numpy.random.seed(42)

EPS = 10.0 ** -4


def input():
    expr = numpy.poly1d([1.0, 38.4621, 364.594, 914.196])
    return expr


(f) = input()


def sturm_method(f):
    arr = []
    arr.append(f)
    arr.append(numpy.polyder(f))

    while True:
        fn = -numpy.polydiv(arr[-2], arr[-1])[1]
        if (fn.order > 0 or abs(fn[0]) > 0.0):
            arr.append(fn)
        else:
            break

    return arr


def n(stseq, x):
    if (abs(f(x)) < EPS):
        raise ValueError("Число в N() является корнем")
```

```
    ans = 0
    for i in range(1, len(stseq)):
        if (stseq[i](x) == 0.0):
            raise ValueError("Элемент ряда Штурма равен
0")
        if (stseq[i - 1](x) * stseq[i](x) < 0):
            ans += 1
    return ans


def get_bounds(f, a, b):
    if ((abs(f(a)) < EPS) or (abs(f(b)) < EPS)):
        raise ValueError("Границы содержат данный
корень")
    if (n(sturm_system, a) - n(sturm_system, b) == 0):
        return []
    if (n(sturm_system, a) - n(sturm_system, b) > 1):
        while True:
            M = a + (b - a) / (1.5 +
numpy.random.random())
            if (abs(f(M)) > EPS):
                break
        return get_bounds(f, a, M) + get_bounds(f, M, b)
    if (b - a < EPS):
        print("Границы маловаты")
    return [(a, b)]


sturm_system = sturm_method(f)

iters = 0


def binary_search(L, R):
    global iters
```

```
    iters += 1
    M = (L + R) / 2
    if (R - L < EPS):
        return M
    # if (abs(f(M)) < EPS):
    #    return M
    if (f(L) * f(M) <= 0):
        return binary_search(L, M)
    elif (f(R) * f(M) <= 0):
        return binary_search(M, R)
    else:
        raise RuntimeError("Ошибка в двоичном поиске")


def chord_method(L, R):
    global iters
    fder2 = numpy.polyder(f, 2)
    if (f(R) * fder2(R) > 0):
        (oldx, x) = (R, L)
    elif (f(L) * fder2(L) > 0):
        (oldx, x) = (L, R)
    else:
        raise ValueError("Границы плоховаты в методе
Хорд")
    t = oldx
    while (abs(x - oldx) > EPS):
        iters += 1
        oldx = x
        x = x - f(x) * (t - x) / (f(t) - f(x))
        if (not (numpy.isfinite(x))):
            raise RuntimeError("Метод Хорд Ошибка")
    if ((x < L) or (R < x)):
        raise RuntimeError("Метод Хорд Ошибка")
    return x
```

```
def newton(L, R):
    global iters
    fder = numpy.polyder(f)
    fder2 = numpy.polyder(f, 2)
    if (f(L) * fder2(L) > 0):
        (oldx, x) = (R, L)
    elif (f(R) * fder2(R) > 0):
        (oldx, x) = (L, R)
    else:
        raise ValueError("Границы плоховаты в методе
Ньютона")
    while (abs(x - oldx) > EPS):
        iters += 1
        oldx = x
        x = x - f(x) / fder(x)
        if (not (numpy.isfinite(x))):
            raise RuntimeError("Метод Ньютона Ошибка")
    if ((x < L) or (R < x)):
        raise RuntimeError("Метод Ньютона Ошибка")
    return x


numpy.set_printoptions(suppress=True, precision=4,
floatmode="fixed")

bounds = get_bounds(f, -10, 10)


def test_method(method):
    global iters
    for i in range(len(bounds)):
        iters = 0
        try:
            str = method(*bounds[i])
```

```
            if (not str is None):
                str = "{:.4f}".format(str)
            print(f"{str} используя {method.__name__}
заняло {iters} итераций)")
            break
        except Exception as ex:
            print("ERROR: {} - in {} method (with {}
iterations)".format(ex, method.__name__, iters))


def main():
    print("Нелинейные уравнения\n")

    print(f"{f}\n")

    print(f"Количество корней на промежутке [-10, 10]:
{n(sturm_system, -10) - n(sturm_system, 10)}")

    bounds = get_bounds(f, -10, 10)
    print("Корни, входящие в промежуток:")
    print(bounds)
    print('\n')

    test_method(binary_search)
    test_method(chord_method)
    test_method(newton)
    print('\n')
    print('Результат, используя встроенные функции')
    print(f.r)


if __name__ == '__main__':
    main()
```

Полученные результаты

Тестовый пример 1

Используя теорему Штурма, определить число корней уравнения: на отрезке [-10, 10]. Вычислить корни с точность до 0.0001 методами половинного деления, хорд и Ньютона. Сравнить число итераций в использованных методах.

x^3 − 6x^2 + 11x − 6 = 0

Ответ:

Количество корней на отрезке: 3		
Метод половинного деления	Метод хорд	Метод Ньютона
x1=1.0000 x2=2.0000 x3=3.0000	x1=1.0000 x2=ошибка x3=3.0000	x1=1.0000 x2=ошибка x3=3.0000
Количество итераций		
18+18+19=55	12+0+43=55	5+0+7=12

Тестовый пример 2

Используя теорему Штурма, определить число корней уравнения: на отрезке [-10, 10]. Вычислить корни с точность до 0.0001 методами половинного деления, хорд и Ньютона. Сравнить число итераций в использованных методах.

x^3 − 3x^2 + 3x − 1 = 0

Ответ:

Количество корней на отрезке: 1		
Метод половинного деления	Метод хорд	Метод Ньютона
x1=1.0000	x1=1.0935	x1=1.0000
Количество итераций		
22	4710	33

Тестовый пример 3

Используя теорему Штурма, определить число корней уравнения: на отрезке [-10, 10]. Вычислить корни с точность до 0.0001 методами половинного деления, хорд и Ньютона. Сравнить число итераций в использованных методах.

x^3 − x^2 = 0

Ответ:

Количество корней на отрезке: 2		
Метод половинного деления	Метод хорд	Метод Ньютона
x1=ошибка x2=1.0000	x1=0.9990 x2=0.9990	x1=−0.0000 x2=1.0000
Количество итераций		
1+21=22	673+663=1336	23+11=34

ЗАДАНИЕ
Вариант 7

Ответ:

Количество корней на отрезке: 2		
Метод половинного деления	Метод хорд	Метод Ньютона
x_min = -8.7648	x_min = -8.7648	x_min = -8.7648
Количество итераций		
17	7	4

Консоль:

```
/Users/lnxd/Desktop/MCHA/LAB3/venv/bin/python /Users/lnxd/Desktop/MCHA/LAB3/main.py
Нелинейные уравнения

   3         2
1 x + 38.46 x + 364.6 x + 914.2

Количество корней на промежутке [-10, 10]: 2
Корни, входящие в промежуток:
[(-10,    -5.730321123990022),    (-5.730321123990022, -1.0394022020647924)]


-8.7648 используя binary_search заняло 17 итераций)
-8.7648 используя chord_method заняло 7 итераций)
-8.7648 используя newton заняло 4 итераций)


Результат, используя встроенные функции
[-25.6273  -8.7648  -4.0700]

Process finished with exit code 0
```

Выводы

Таким образом, в ходе выполнения лабораторной работы были изучены методы численного решения нелинейных уравнений (метод простой итерации, метод хорд, метод Ньютона), исследована скорость сходимости итерационных процедур, изучен метод Эйткена ускорения сходимости, 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, а в случае ошибки вычислений – метода половинного деления.