Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»

Кафедра информатики

Отчет по лабораторной работе №3

Численное решение нелийнейных уравнений

Выполнил: студент гр. 053501

Кононович С. В.

Руководитель: доцент

Анисимов В. Я.

Минск 2022

**Содержание**

1. Цель работы .......................................................................................3
2. Теоретические сведения ...................................................................3
3. Програмная реализация.....................................................................8

Задание 1.............................................................................................8

Задание 2............................................................................................10

Задание 3............................................................................................12

1. Выводы...............................................................................................14
2. **Цель работы**

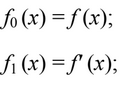
* - изучить методы численного решения нелинейных уравнений – метод хорд, простых итераций, метод Ньютона и его модификацию;
* - исследовать скорость сходимости итерационных процедур;
* - изучить метод Эйткена ускорения сходимости;
* - сравнить число итераций, необходимое для достижения заданной точности вычисления разными методами

1. **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.

Для определения числа корней на заданном промежутке используется Теорема Штурма: Если f (х) многочлен и уравнение не имеет кратных корней на промежутке[а, b], то число корней уравнения f (х) = 0, лежащих на промежутке [а, b], совпадает с числом N(a) - N(b), которое определяется из следующей процедуры.

*Строим ряд Штурма* *, где*



f0(x*) делим на f1(х) и* в *качестве* f2(x*) берем остаток от деления, взятый с обратным знаком;*

f1(x*) делим на f2(х) и* в *качестве* f3(x*) берем остаток от деления, взятый с обратным знаком;*

*и т.д.*

*Полагаем N(a) — число перемен знака в ряде Штурма, если вместо х подставлена точка а, N(b) - число перемен знака в ряде Штурма, если вместо х подставлена точка b.*

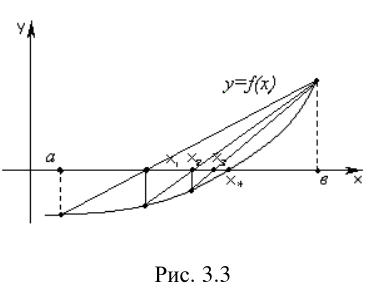
Для отделения корней можно использовать график функции y=f(x). Корнями уравнения являются те значения х, при которых график функции пересекает ось абсцисс. Построение графика функции даже с малой точностью обычно дает представление о расположении и характере корней уравнения (иногда позволяет выявить даже корни четной кратности).

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 е обычно применяют какую-либо итерационную процедуру ***уточнения корня,*** строящую числовую последовательность значений *xn* сходящуюся к искомому корню уравнения. Начальное приближение x0 выбирают на отрезке [а,b], продолжают вычисления, пока не выполнится неравенство , и считают, что хn, - 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.*

**Метод хорд**. Пусть дано уравнение  *,* где *f(x)* - дважды непрерывно дифференцируемая функция.

Пусть выполняется условие *f(а) f(b) <* 0 и проведено отделение корней, то есть на данном интервале *(а, b)* находится один корень уравнения. При этом, не ограничивая общности, можно считать, что f(b) *> 0 .*

Пусть функция *f* выпукла на интервале *(а, b)* (см. рис. 3.3).



Заменим график функции хордой (прямой), проходящей через точки

*М0(а, f(а))* и *М1(b, f(b)) .*

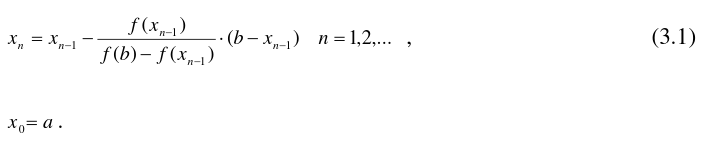
Уравнение прямой, проходящей через две заданные точки, можно записать в виде . В нашем случае получим: 

Найдем точку пересечения хорды с осью Ох. Полагая y = 0, получаем из предыдущего уравнения:

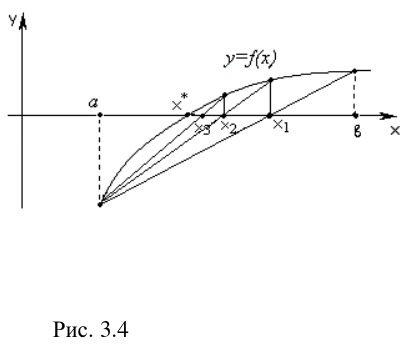


Теперь возьмем интервал *(х1 ,b)* в качестве исходного и повторим вышеописанную процедуру (см. рис. 6.3). Получим 

Продолжим процесс. Каждое последующее приближение вычисляется по рекуррентной формуле

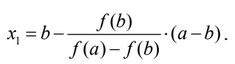


Если же функция вогнута (см. рис. 3.4),



уравнение прямой соединяющей точки M0(а, *f(а))* и *М1(b, f(b))* запишем в виде 

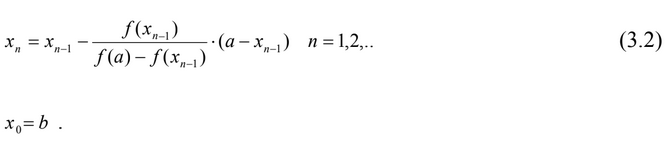
Найдем точку пересечения хорды с осью Ох:



Теперь возьмем интервал *(а, х1)* в качестве исходного и найдем точки пересечения хорды, соединяющей точки *(а, f(a))* и *(х1, f(x1)),* с осью абсцисс (см. рис. 3.4). Получим

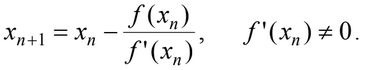


Повторяя данную процедуру, получаем рекуррентную формулу:



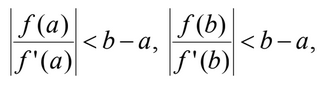
Описанный выше метод построения рекуррентных последовательностей (3.1) и (3.2) называется методом хорд. Для использования метода хорд нужно было бы предварительно найти точки перегиба и выделить участки, на которых функция не меняет характер выпуклости. Однако на практике поступают проще: в случае *f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

**Метод Ньютона (касательных).** Для начала вычислений требуется задание одного начального приближения x0. Последующие приближения вычисляются по формуле



Метод имеет квадратичную скорость сходимости для простого корня, но очень чувствителен к выбору начального приближения. При произвольном начальном приближении итерации сходятся, если всюду *| f (х) f ” (х)| < (f' (х))*2 , в противном случае сходимость будет только при

достаточно близком к корню. Существует несколько достаточных условий сходимости. Если производные *f’(х)* и *f”(x)* сохраняют знак в окрестности корня, рекомендуется выбирать х0 так, чтобы *f* (x 0 ) *f”* (x 0 ) > 0. Если, кроме этого, для отрезка *[а,b],* содержащего корень, выполняются условия



то метод сходится для любых *a <= x0 <= b.*

# **3. Программная реализация**

**Вариант 10.**

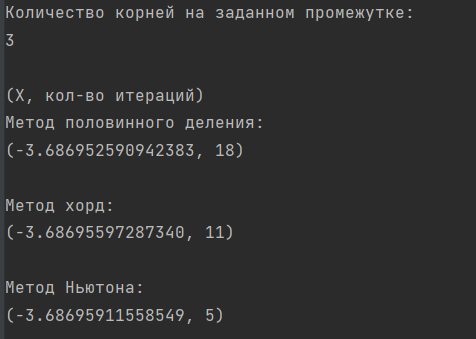
## **Задание 1.**

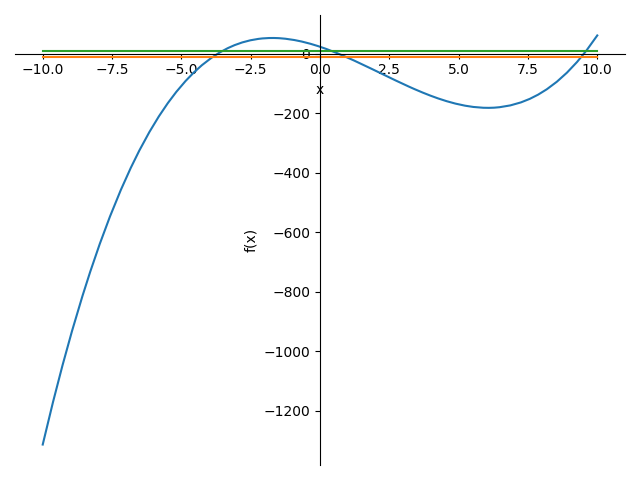
* Используя теорему Штурма определить число корней уравнения:

на отрезке [-10,10]. Значения коэффициентов уравнения взять из таблицы.

* Отделить все корни, лежащие на данном отрезке.
* 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.

Решение задачи:





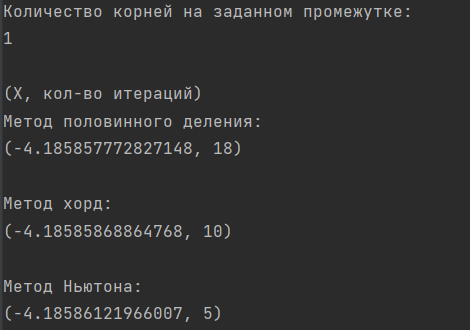
## **Задание 2. Тестовое**

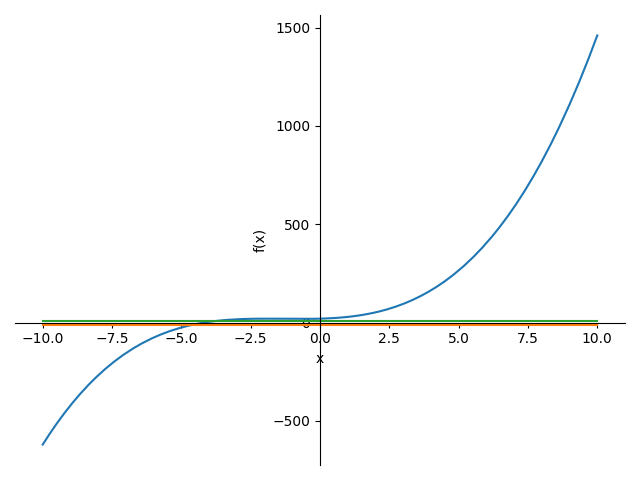
* Используя теорему Штурма определить число корней уравнения:

на отрезке [-10,10]. Значения коэффициентов уравнения взять из таблицы.

* Отделить все корни, лежащие на данном отрезке.
* 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.

Решение задачи:





## **Задание 3. Тестовое**

* Используя теорему Штурма определить число корней уравнения:

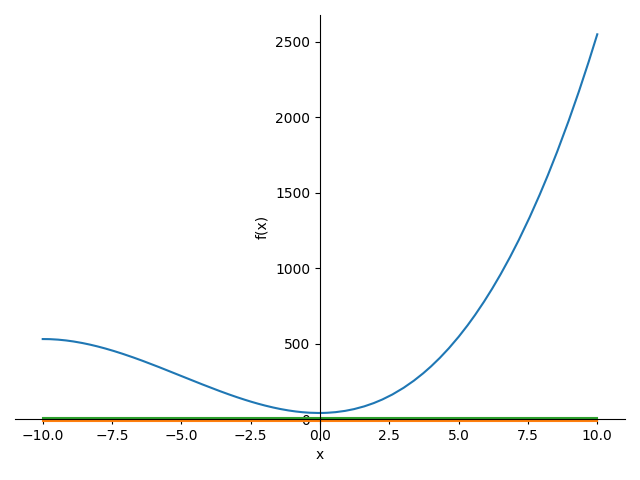
на отрезке [-10,10]. Значения коэффициентов уравнения взять из таблицы.

* Отделить все корни, лежащие на данном отрезке.
* 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.

Решение задачи:



График не имеет пересечения с Ox, т.к. корни отсутствуют.



import sympy as sp  
from sympy.abc import x  
from sympy.plotting import plot  
  
a = sp.Float(15)  
b = sp.Float(1)  
c = sp.Float(40)  
base\_poly = sp.poly(x \*\* 3 + a \* x \*\* 2 + b \* x + c)  
  
  
def shturm\_amount\_of\_roots(poly=None, left\_border=None, right\_border=None):  
 def shturm\_range\_value(\_poly\_range\_, value):  
 counter = 0  
 size = len(\_poly\_range\_)  
 current\_sgn = \_poly\_range\_[0](value) > 0  
 for i in range(size - 1):  
 new\_sgn = \_poly\_range\_[i + 1](value) > 0  
 if new\_sgn != current\_sgn:  
 counter += 1  
 current\_sgn = new\_sgn  
 return counter  
  
 shturm\_sequence = [poly, sp.diff(poly)]  
 sequence\_range = sp.degree(poly, gen=x)  
 for i in range(sequence\_range - 1):  
 shturm\_sequence.append(-sp.div(shturm\_sequence[i], shturm\_sequence[i + 1])[1])  
 return shturm\_range\_value(shturm\_sequence, left\_border) - shturm\_range\_value(shturm\_sequence, right\_border)  
  
  
def printPlot(left\_border=None, right\_border=None):  
 plot(x \*\* 3 + a \* x \*\* 2 + b \* x + c, left\_border, right\_border)  
 return  
  
  
def bisection(poly, left\_border, right\_border, accuracy):  
 global average  
 counter = 0  
 while abs((left\_border - right\_border)) > accuracy:  
 average = (left\_border + right\_border) / 2  
 if poly(left\_border) \* poly(average) <= 0:  
 right\_border = average  
 else:  
 left\_border = average  
 counter += 1  
 return average, counter  
  
  
def chord(poly, left\_border, right\_border, accuracy):  
 temp = 0  
 average = 10 \* accuracy  
 counter = 0  
 while abs(temp - average) > accuracy:  
 temp = average  
 average = left\_border - (poly(left\_border) / (poly(right\_border) - poly(left\_border))) \* (  
 right\_border - left\_border)  
 if poly(left\_border) \* poly(average) <= 0:  
 right\_border = average  
 else:  
 left\_border = average  
 counter += 1  
 return average, counter  
  
  
def newton(poly, entrypoint, accuracy):  
 derivative\_func = sp.diff(poly)  
 temp = entrypoint + accuracy \* 10  
 counter = 0  
 while abs(entrypoint - temp) > accuracy:  
 temp = entrypoint  
 entrypoint = entrypoint - poly(entrypoint) / derivative\_func(entrypoint)  
 counter += 1  
 return entrypoint, counter  
  
  
print("Количество корней на заданном промежутке:")  
print(shturm\_amount\_of\_roots(base\_poly, -10, 10))  
printPlot(left\_border=-10, right\_border=10)  
print()  
print("(Х, кол-во итераций)")  
print("Метод половинного деления:")  
print(bisection(base\_poly, -5, -2.5, 1e-5))  
print()  
print("Метод хорд:")  
print(chord(base\_poly, -5, -2.5, 1e-5))  
print()  
print("Метод Ньютона:")  
print(newton(base\_poly, -5, 1e-5))

# **4. Выводы**

Таким образом, в ходе выполнения лабораторной работы я изучил методы численного решения нелинейных уравнений (методы хорд, простых итераций, метод Ньютона и его модификацию).

Исследовал данные методы на сходимость и практическим путем проверил и убедился в том, что метод половинного деления сходится за большее число (в большинстве случаев) итераций, чем метод хорд, который в свою очередь сходится медленнее(в большинстве случаев), чем метод Ньютона, который ко всему прочему тоже можно ускорить.