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 053506

Слуцкий Никита Сергеевич

Проверил: Анисимов Владимир Яковлевич

Минск 2022

**Оглавление**

[**Цели выполнения задания**](#_Toc65424824)

[**Краткие теоретические сведения**](#_Toc65424825)

[**Задание**](#_Toc65424826)

[**Программная реализация**](#_Toc65424827)

[**Полученные результаты**](#_Toc65424828)

[**Тестовые задания**](#_Toc65424828)

[**Выводы**](#_Toc65424830)

**Вариант 9 (Номер в журнале – 21)**

**Цели выполнения задания:**

* Изучить методы численного решения нелинейных уравнений
  + Метод половинного деления
  + Метод Ньютона (метод касательных)
  + Метод хорд
* Сравнить количества итераций, необходимых для достижения требуемой точности, в разных методах
* Изучить метод нахождения количества корней уравнения на отрезке

**Краткие теоретические сведения:**

В общем виде не существует решения любых типов уравнений. В частных случаях существует прямой способ нахождения корней в линейном, квадратном, кубическом уравнениях. Остальные типы уравнений решаются с помощью некоторых теорем, а также приближёнными методами решений.

Численное решение нелинейного уравнения f(x) = 0 заключается в вычислении с заданной точностью значения корней уравнения. Разбивается на несколько этапов. Первый – это определение количества корней на отрезке, а также их тип (действительные/комплексные, простые/кратные). Второй – для каждого корня найти приблизительное расположение. Расположение здесь подразумевает отрезок, на котором лежит только один этот рассматриваемый корень. Процесс нахождения называется отделением корней. Например, пусть для уравнения f(x) = 0 существует корень x1 = 2.5. Тогда для него вышеописанным расположением могут быть интервалы, например, [0..3], [2..3], [-100..100], ПРИ УСЛОВИИ, что это будет ЕДИНСТВЕННЫЙ КОРЕНЬ на отрезке. Существует ещё одно условие наличия по крайней мере одного корня уравнения на отрезке. Это условие следует из теоремы Больцано-Коши. Теорема эта гласит, что если функция непрерывна на отрезке [a..b] и f(a) = A, f(b) = B, то существует некоторая точка c (c принадлежит [a..b]), для которой f(c) = C, где уже C принадлежит отрезку от A до B. Тут, кстати, уже была потребована непрерывность. Так что стоит это также учитывать на этапе подсчёта и отделения корней. Ну а после отделения корней идёт какой-то итерационный процесс (три из них будут описаны ниже), который и “подберётся” к искомому корню настолько близко, насколько точным требуется решение.   
Рассмотрю первый этап нахождения корней уравнения (для случая с полиномом n-й степени). Для этого можно использовать метод Штурма. Соответствующая теорема гласит: Если p(x) – многочлен и уравнение НЕ ИМЕЕТ КРАТНЫХ КОРНЕЙ на промежутке [a..b], то число корней уравнения f(x) = 0 на данном отрезке совпадает с числом Т=| N(a) – N(b) |.

N(a), N(b) – это числа перемены знака при подстановке a, b в так называемый ряд (или систему) Штурма. Построю этот ряд. Ряд этот имеет вид p0(x), p1(x), ... , pm(x). Правила построение следующие:

p0(x) = p(x) // исходный многочлен

p1(x) = p’(x) // его производная

p2(x) = p0(x) % p1(x) // что означает остаток от деления p0(x) на p1(x)

...

pm(x) = pm-2(x) % pm-1(x)

Так происходит, пока многочлен не превратится в “нулевой”. Система Штурма получена. Чтобы вычислить вышеупомянутое N(a), нужно подставить a в многочлены системы 0 – m, найти значения и посмотреть, сколько раз происходит смена знака. Например, для переменной t получается ситуация “+ + - + + - -” — знак меняется 3 раза, то есть N(t) = 3. Таким образом | N(a) – N(b) | покажет количество корней на отрезке [a..b]. В этом и заключается метод (теорема) Штурма.

Далее стоит задача отделить корни. Сделать это можно графически, например, воспользовавшись системой компьютерной алгебры Maple. Выделить отрезки с одним корнем, для которых f(a) \* f(b) < 0 (разного знака значения функции на концах) и внутри которых ОДИН корень.

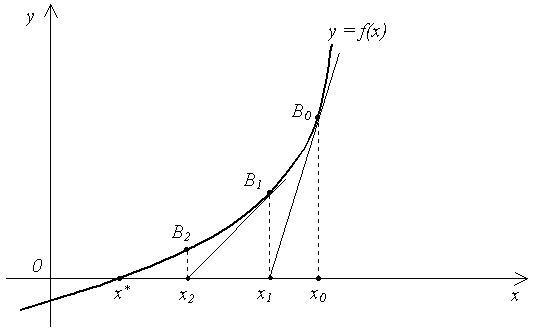
Также можно пытаться (при отсутствии графика) использовать теорему Ролля. С помощью неё можно найти стационарные точки, отрезки которых можно пытаться интерпретировать как отрезки, содержащие один корень. Но решение уравнения f’(x) = 0 – задача также трудоёмкая. Поэтому ограничусь графическим методом.

А теперь настало время непосредственно найти приближённое значение корней. Выбираю отрезок из второго пункта, чтобы найти на нём уточнённое приближённое значение корня.

Метод Дихотомии (половинного деления, бинарного поиска) заключается в следующем: есть отрезок [a..b], f(a) \* f(b) < 0. Нахожу середину отрезка c = (a + b) / 2. Смотрю значение функции в этой функции f(c). В зависимости от того, больше оно нуля или меньше, заменяю левую или правую границу отрезка поиска на c. Например, [a..c]. Теперь отрезок поиска уменьшился в два раза. Таким образом на каждой итерации отрезок поиска уменьшается в два раза, я получаю в каком-то смысле систему вложенных отрезков. Они стягиваются в какую-то точку. Пока разность | a’ – b’ | больше заданной точности, я буду сужать диапазон. При достижении точности я в очередной раз посчитаю среднее значение ( am + bm ) / 2 и верну его как значение найденного корня. Значение f от полученного x будет достаточно близко к нулю. Почему ? Потому что функция непрерывна. Малому приращению аргумента соответствует малое приращение функции. Поэтому когда разность между границами меньше заданного малого числа точности, а функция на концах этого отрезка с этими границами имеет разный знак, то и от нуля она будет отстоять относительно недалеко.

Метод Ньютона

Классический метод Ньютона или касательных заключается в том, что если xn — некоторое приближение к корню уравнения f(x) = 0, то следующее приближение определяется как корень касательной к функции f(x), проведенной в точке xn.



Метод хорд

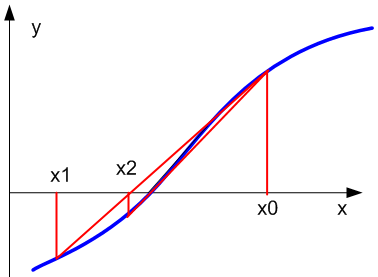
Если **x0**, **x1** - приближенные значения корня уравнения **f(x) = 0** и выполняется условие f(a) \* f(b) < 0, то последующие приближения находят по формуле:

Метод хорд

Методом хорд называют также метод, при котором один из концов отрезка закреплен, т.е. вычисление приближения корня уравнения **f(x) = 0** производят по формулам:

Метод секущих

Геометрическая интерпретация метода хорд:



**Задание**

* Используя теорему Штурма определить число корней уравнения:  
  x3 + ax2 + bx + c = 0 на отрезке [-10..10]. Значения взять из таблицы. Для моего варианта:  
  
* Отделить корни, лежащие на данном отрезке
* Вычислить наименьший из корней:
  + Методом половинного деления
  + Методом хорд
  + Методом Ньютона

Сравнить число итераций. Точность – до 0.0001

**Программная реализация:**

Ниже можно ознакомиться с программной реализацией главных функций по триангуляции, обратному ходу, а также вызов этого всего из главного файла. Вспомогательные функции и перегруженные операторы (реализации) опущены. Реализация на языке программирования Python с использованием библиотеки NumPy.

Метод Штурма:

**def get\_sturm\_system(polynom: list) -> list:  
 sturm\_system: list = [np.poly1d(polynom), np.poly1d(polynom).deriv()]  
  
 counter: int = 1  
  
 while sturm\_system[counter][0] != 0.00:  
 counter += 1  
 sturm\_system.append(np.polydiv(sturm\_system[counter - 2], sturm\_system[counter - 1])[1] \* -1)  
  
 return sturm\_system  
  
  
def get\_number\_of\_sign\_changes(sturm\_series: list, value\_to\_insert: float) -> int:  
 response: int = 0  
  
 for counter in range(1, len(sturm\_series)):  
 if np.sign(sturm\_series[counter](value\_to\_insert)) != np.sign(sturm\_series[counter - 1](value\_to\_insert)):  
 response += 1  
  
 return response  
  
  
def get\_number\_of\_roots\_by\_sturm\_theorem(source: list, range\_from: float, range\_to: float) -> int:  
 sturm\_system = get\_sturm\_system(source)  
  
 left\_range\_side\_sign\_changes\_number: int = get\_number\_of\_sign\_changes(sturm\_system, range\_from)  
 right\_range\_side\_sign\_changes\_number: int = get\_number\_of\_sign\_changes(sturm\_system, range\_to)  
  
 return abs(left\_range\_side\_sign\_changes\_number - right\_range\_side\_sign\_changes\_number)**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Метод половинного деления:

**def get\_root\_by\_half\_division(source: list[float], search\_from: float, search\_to: float, error: float) -> (float, int):  
 polynom = np.poly1d(source)  
  
 left\_border: float = search\_from  
 right\_border: float = search\_to  
  
 if polynom(left\_border) \* polynom(right\_border) > 0:  
 raise Exception('Condition f(a) \* f(b) is not met')  
  
 middle\_of\_interval: float = (left\_border + right\_border) / 2  
 iterations\_count: int = 0  
  
 while abs(polynom(middle\_of\_interval)) >= error:  
 middle\_of\_interval = (left\_border + right\_border) / 2  
  
 if polynom(left\_border) \* polynom(middle\_of\_interval) < 0:  
 right\_border = middle\_of\_interval  
 elif polynom(middle\_of\_interval) \* polynom(right\_border) < 0:  
 left\_border = middle\_of\_interval  
  
 iterations\_count += 1  
  
 return (left\_border + right\_border) / 2, iterations\_count**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Метод хорд:

***# Для выпуклой (f'' > 0)*def for\_convex\_function(polynom, left\_border: float, right\_border: float, error: float) -> (float, int):  
 root: float = left\_border  
 iterations\_count: int = 0  
  
 while abs(polynom(root)) > error:  
 root = root - (polynom(root) / (polynom(right\_border) - polynom(root))) \* (right\_border - root)  
 iterations\_count += 1  
  
 return root, iterations\_count  
  
  
*# Для вогнутой (f'' < 0)*def for\_concave\_function(polynom, left\_border: float, right\_border: float, error: float) -> (float, int):  
 root: float = right\_border  
 iterations\_count: int = 0  
  
 while abs(polynom(root)) > error:  
 root = root - (polynom(root) / (polynom(left\_border) - polynom(root))) \* (left\_border - root)  
 iterations\_count += 1  
  
 return root, iterations\_count  
  
  
def get\_root\_by\_chord\_method(source: list[float], search\_from: float, search\_to: float, error: float) -> (float, int):  
 polynom = np.poly1d(source)  
  
 left\_border: float = search\_from  
 right\_border: float = search\_to  
  
 if polynom(left\_border) \* polynom(right\_border) > 0:  
 raise Exception('Condition f(a) \* f(b) is not met')  
  
 second\_derivative = np.polyder(polynom, 2)  
  
 if polynom(right\_border) \* second\_derivative(right\_border) > 0:  
 return for\_convex\_function(polynom, left\_border, right\_border, error)  
 elif polynom(left\_border) \* second\_derivative(left\_border) > 0:  
 return for\_concave\_function(polynom, left\_border, right\_border, error)**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Метод касательных (Ньютона):

**def get\_root\_by\_newton\_method(source: list[float], search\_from: float, search\_to: float, error: float) -> (float, int):  
 polynom = np.poly1d(source)  
  
 left\_border: float = search\_from  
 right\_border: float = search\_to  
  
 first\_derivative = np.polyder(polynom)  
 second\_derivative = np.polyder(first\_derivative)  
  
 root: float = left\_border  
 iterations\_count: float = 0  
  
 if second\_derivative(right\_border) \* polynom(right\_border) > 0:  
 root = right\_border  
 elif second\_derivative(left\_border) \* polynom(left\_border) > 0:  
 root = left\_border  
  
 while abs(polynom(root)) > error:  
 root = root - polynom(root) / first\_derivative(root)  
 iterations\_count += 1  
  
 return root, iterations\_count**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

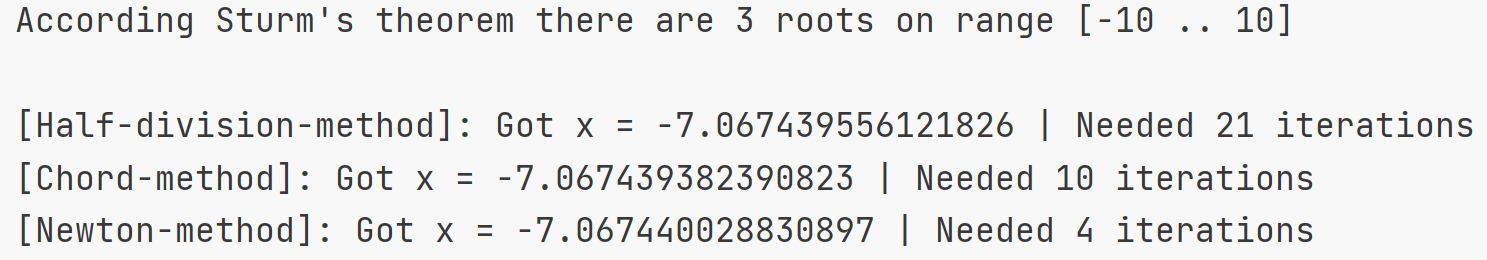
Вызов всех функций для проверки:

**def test\_run() -> None:  
 number\_of\_roots: int = get\_number\_of\_roots\_by\_sturm\_theorem(data.TEST\_EQUATION, data.RANGE\_FROM, data.RANGE\_TO)  
  
 print(f'According Sturm\'s theorem there are {number\_of\_roots} roots on range [{data.RANGE\_FROM} .. {data.RANGE\_TO}]\n')  
  
 *# The following values are taken from chart (Maple)* look\_for\_from: float = 0  
 look\_for\_to: float = 2  
  
 root, iterations\_count = get\_root\_by\_half\_division(data.TEST\_EQUATION, look\_for\_from, look\_for\_to, data.ERROR)  
 print(f'[Half-division-method]: Got x = {root} | Needed {iterations\_count} iterations')  
  
 root, iterations\_count = get\_root\_by\_chord\_method(data.TEST\_EQUATION, look\_for\_from, look\_for\_to, data.ERROR)  
 print(f'[Chord-method]: Got x = {root} | Needed {iterations\_count} iterations')  
  
 root, iterations\_count = get\_root\_by\_newton\_method(data.TEST\_EQUATION, look\_for\_from, look\_for\_to, data.ERROR)  
 print(f'[Newton-method]: Got x = {root} | Needed {iterations\_count} iterations')**

**Полученные результаты:**

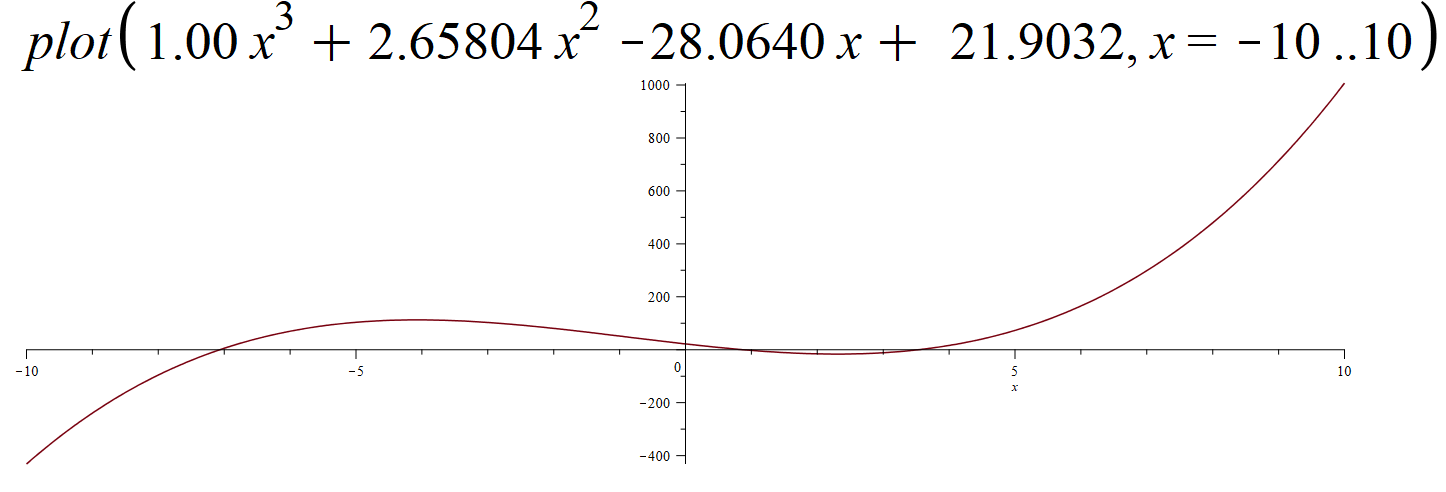
В условии 9-го варианта мне дан многочлен:   
x3 + 2.65804 \* x2 - 28.0640 \* x + 21.9032.

Результат отработки программного продукта:



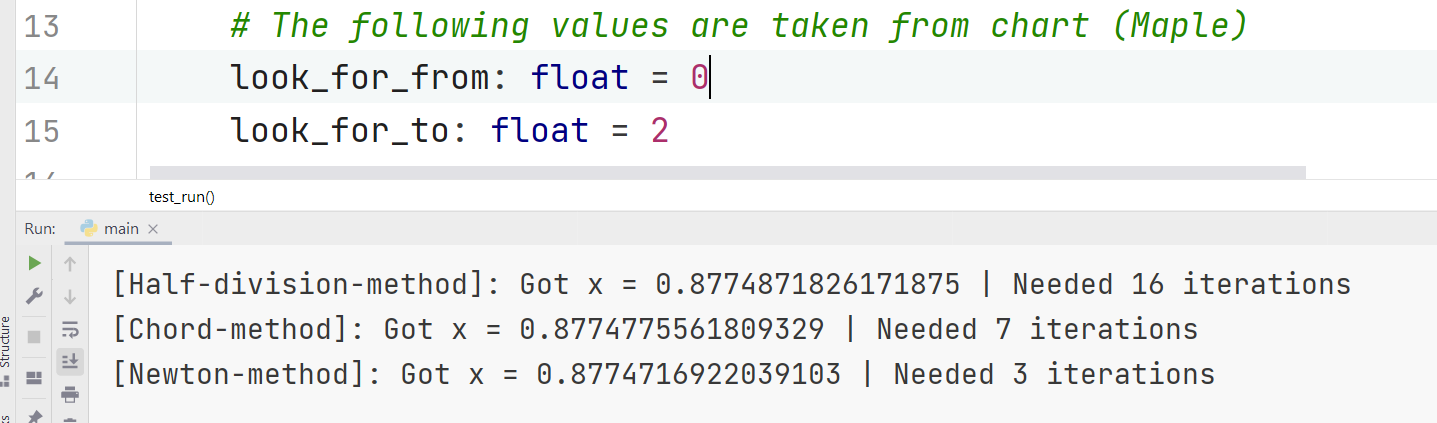
Отрезком поиска был задан отрезок, содержащий один корень, - [-8 .. -2]. Методу дихотомии понадобилась 21 итерация. Методу хорд – 10. А методу касательных – всего 4.

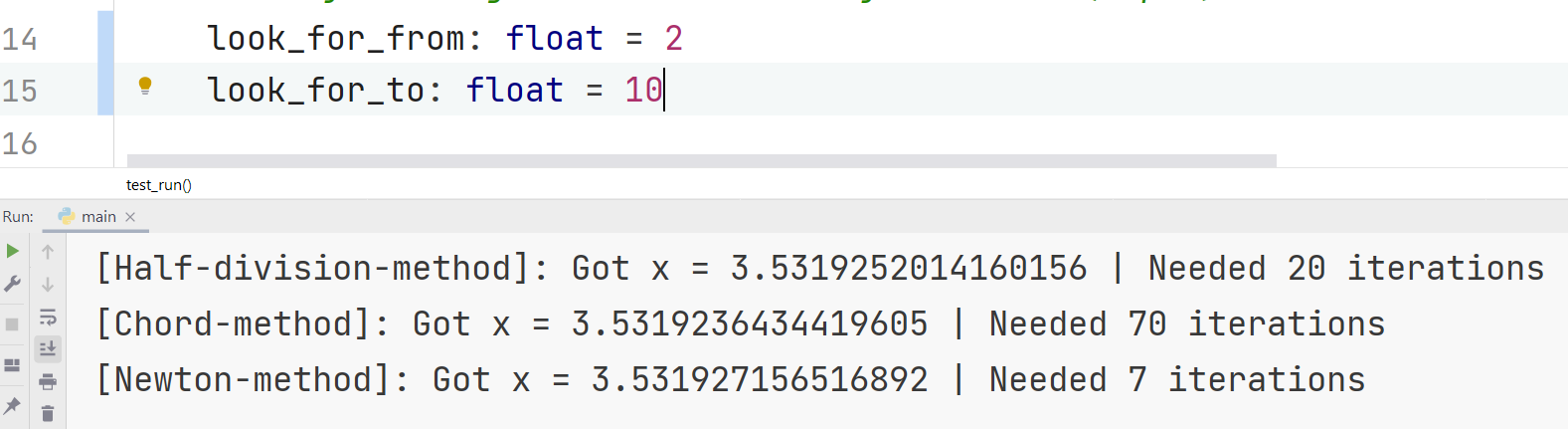
Отрезок был выбран по графику после построения в СКА Maple.



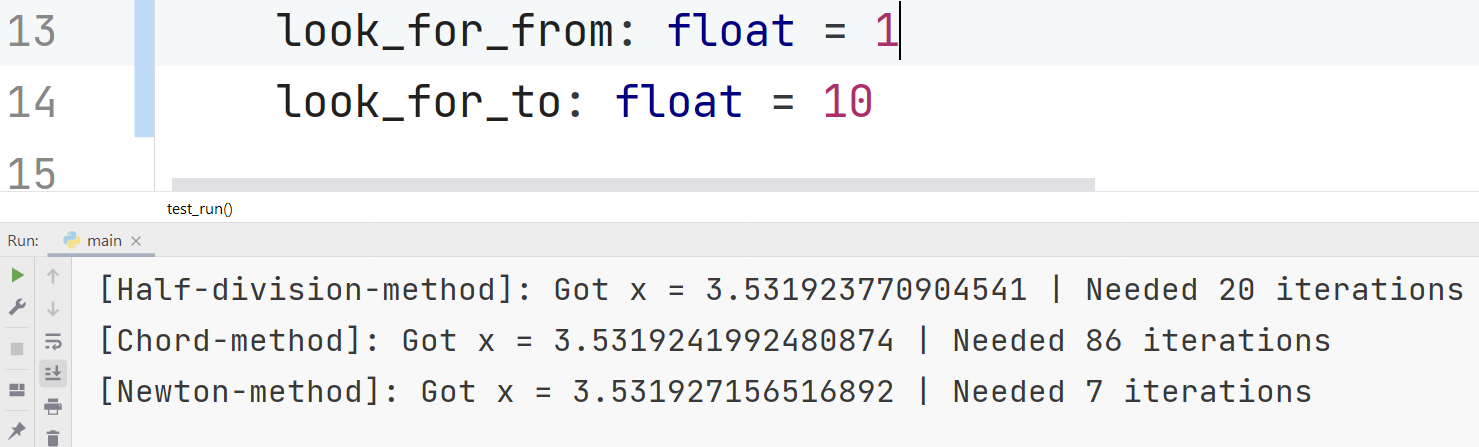
# **Тестовые задания**

Во-первых предлагаю найти все отделённые корни (для моего же варианта). Для этого прогоню программу с интервалом поиска [0..2] и [2..10].

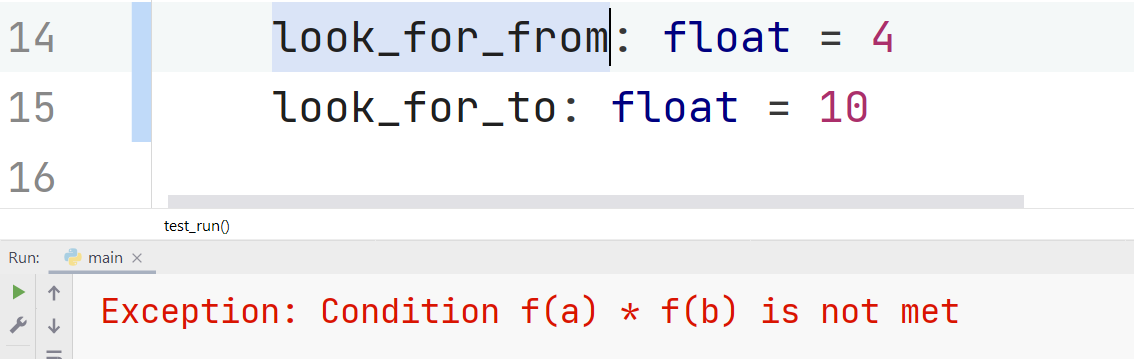




В общем случае в рассматриваемых примерах метод касательных демонстрирует более быструю сходимость. Причём даже при расширении интервала как-то сохраняет устойчивое число итераций:

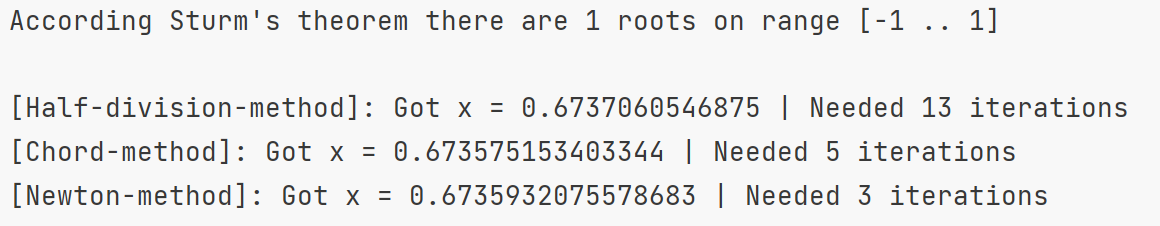
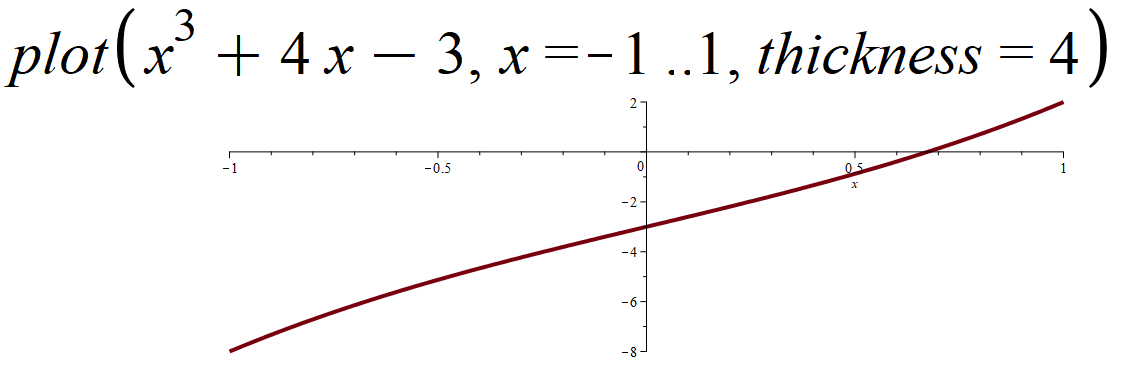


При попытке указать отрезок, для которого f(a) \* f(b) > 0, выкидывается соответствующее исключение:

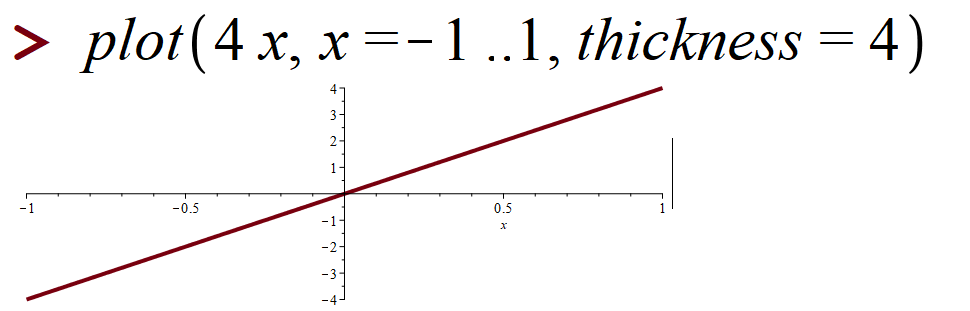


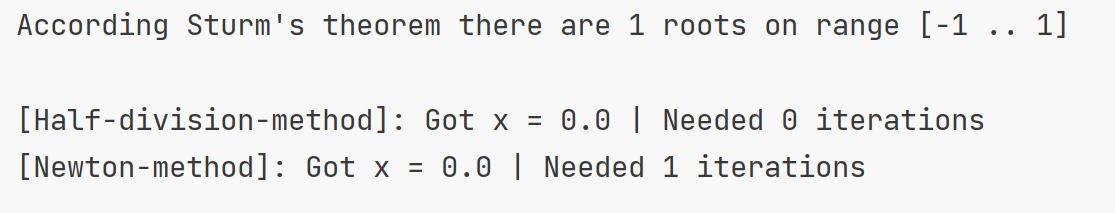
Ещё несколько примеров для разных уравнений, чтобы посмотреть динамику отношений количества итераций в разных методах.

1. **x3 + 4x -3 = 0, отрезок [-1..1]:**



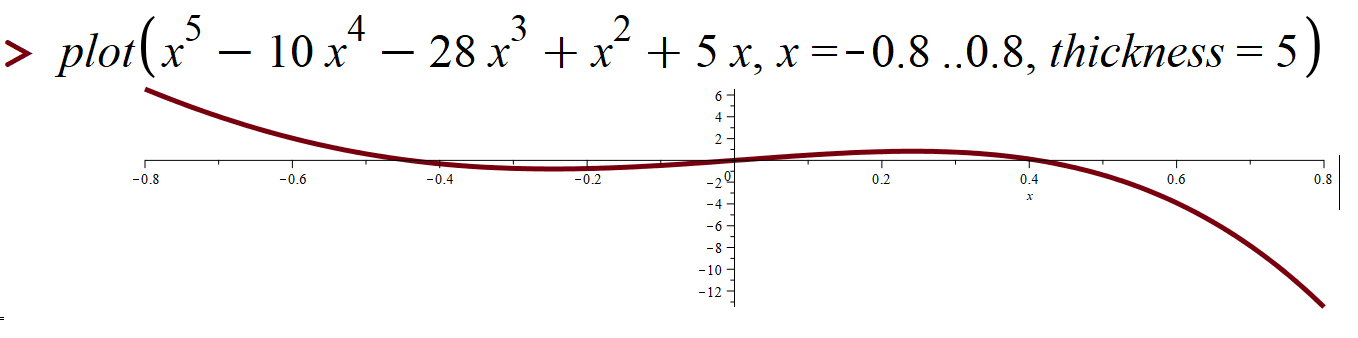
1. **4x = 0, отрезок [-1..1]:**





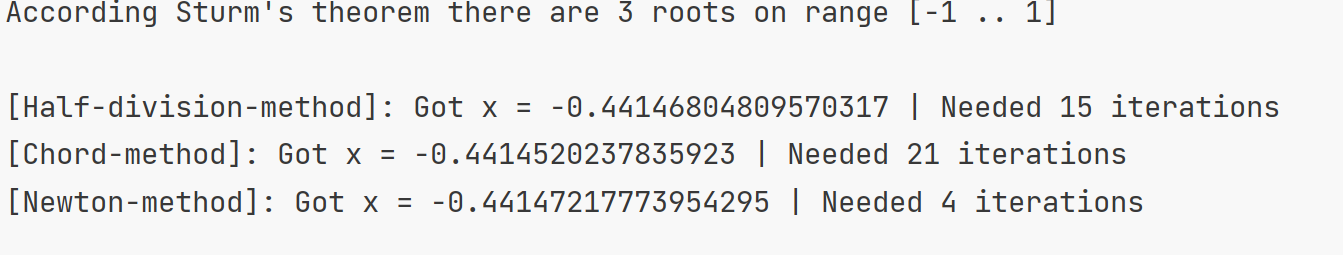
Метод дихотомии, метод касательных и теорема Штурма отрабатывают верно. Метод хорд выкидывает исключение.

1. **x5 - 10 x4 - 28 x3 + x2 + 5 x, отрезок [-0.8..0..8]:**

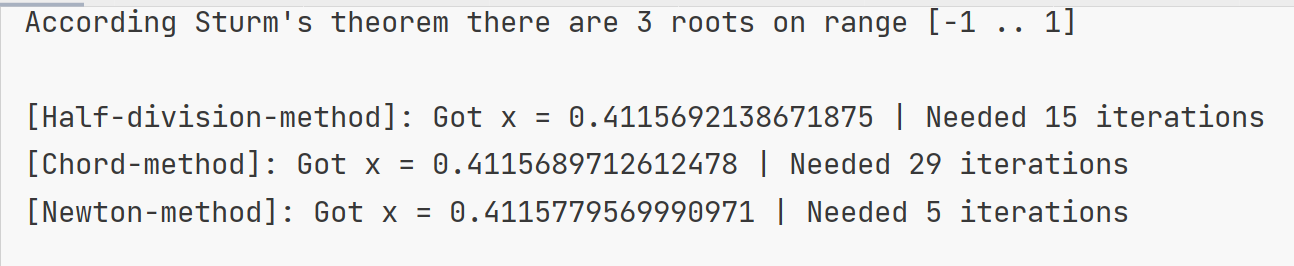


Выбранный отрезок для вычисления корня:

**3.1)** [-0.8..-0.05]



**3.2)** [0.2..0.8]



# **Выводы**

В результате прогона тестовых примеров через мой программный продукт, я пробую сделать предположение, что метод Ньютона является наиболее эффективным методом в общем случае для нахождения корней как минимум обычных многочленов. По количеству итераций в тестовых примерах он порой отрабатывал за в десятки раз меньшее число итераций, чем метод хорд или метод дихотомии. Да и это относительно малое количество итераций достигается не таким уж и трудоёмким путём. От меня лишь требуется один раз посчитать производную, что в случае с многочленом одной переменной не составляет большой вычислительной трудности, а далее лишь “строить прямые” и “находить точки пересечения прямых с осью Ox”, что сводится к относительно простым формулам с классическими арифметическими действиями. Правда, для выбора начального значения нужно будет ещё одну производную (II-го порядка) посчитать, но это также несложная операция для многочлена.