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

Шишко Виктор Викторович

Проверил: Анисимов Владимир Яковлевич

Минск 2023

Оглавление

[Цели выполнения задания 3](file:///C:\Users\Asus\AppData\Local\Temp\6bfaaed3-db69-40d2-bc61-cd03dc68b120_1%20подгруппа-20230831T163939Z-001.zip.120\1%20подгруппа\Дайнович%20О.%20Л\Лабораторная_работа_3_Дайнович.docx#_Toc116778377)

[Краткие теоретические сведения 4](file:///C:\Users\Asus\AppData\Local\Temp\6bfaaed3-db69-40d2-bc61-cd03dc68b120_1%20подгруппа-20230831T163939Z-001.zip.120\1%20подгруппа\Дайнович%20О.%20Л\Лабораторная_работа_3_Дайнович.docx#_Toc116778378)

[Задание 12](file:///C:\Users\Asus\AppData\Local\Temp\6bfaaed3-db69-40d2-bc61-cd03dc68b120_1%20подгруппа-20230831T163939Z-001.zip.120\1%20подгруппа\Дайнович%20О.%20Л\Лабораторная_работа_3_Дайнович.docx#_Toc116778379)

[Программная реализация 13](file:///C:\Users\Asus\AppData\Local\Temp\6bfaaed3-db69-40d2-bc61-cd03dc68b120_1%20подгруппа-20230831T163939Z-001.zip.120\1%20подгруппа\Дайнович%20О.%20Л\Лабораторная_работа_3_Дайнович.docx#_Toc116778380)

[Полученные результаты 17](file:///C:\Users\Asus\AppData\Local\Temp\6bfaaed3-db69-40d2-bc61-cd03dc68b120_1%20подгруппа-20230831T163939Z-001.zip.120\1%20подгруппа\Дайнович%20О.%20Л\Лабораторная_работа_3_Дайнович.docx#_Toc116778381)

[Выводы 22](file:///C:\Users\Asus\AppData\Local\Temp\6bfaaed3-db69-40d2-bc61-cd03dc68b120_1%20подгруппа-20230831T163939Z-001.zip.120\1%20подгруппа\Дайнович%20О.%20Л\Лабораторная_работа_3_Дайнович.docx#_Toc116778382)

Вариант 27

Цели выполнения задания

1) Изучить методы численного решения нелинейных уравнений (метод половинного деления (биссекции), метод хорд, метод Ньютона)

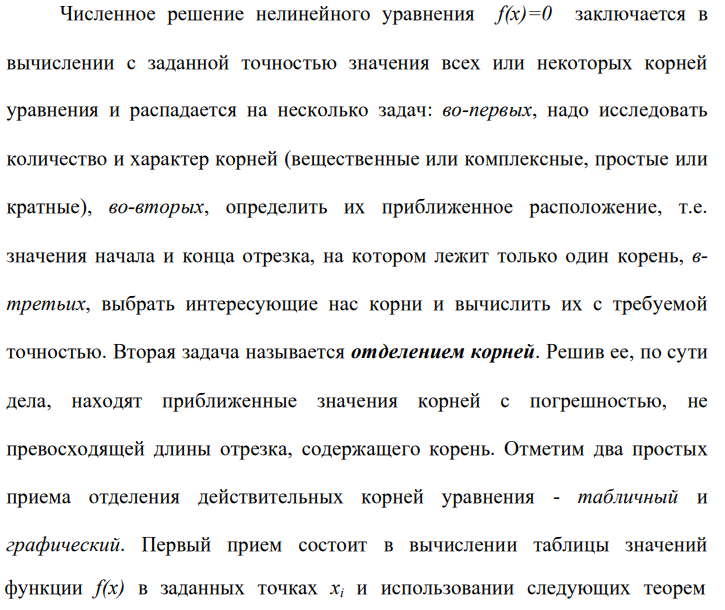
2) Исследовать скорость сходимости итерационных процедур

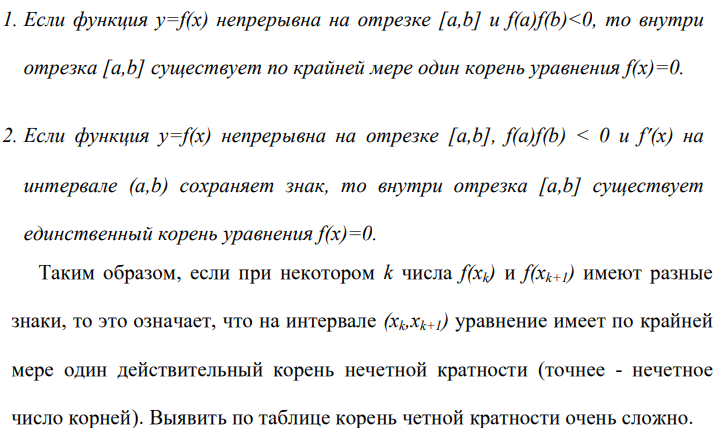
3) Составить программу численного решения нелинейных уравнений методами половинного деления (биссекции), хорд, Ньютона

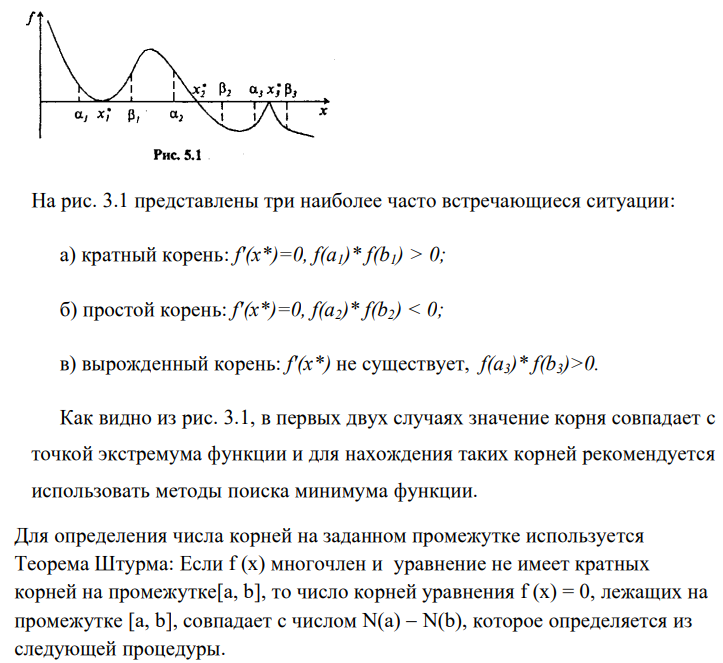
4) Проверить правильность работы программы на тестовых примерах

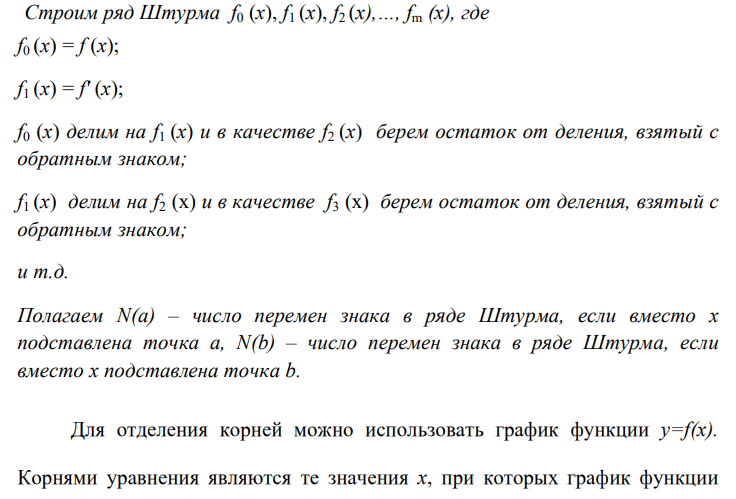
5) Численно решить нелинейное уравнение заданного варианта

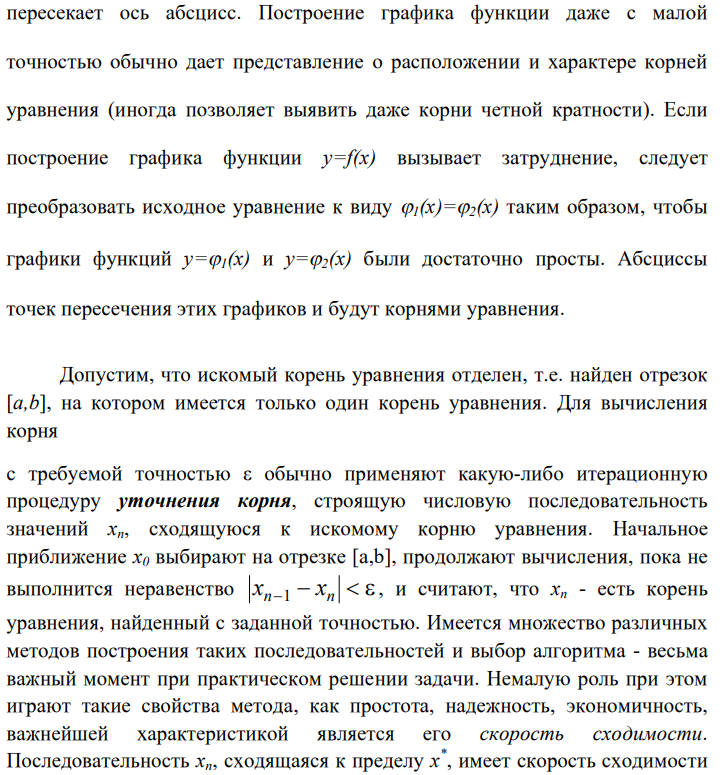
Краткие теоретические сведения

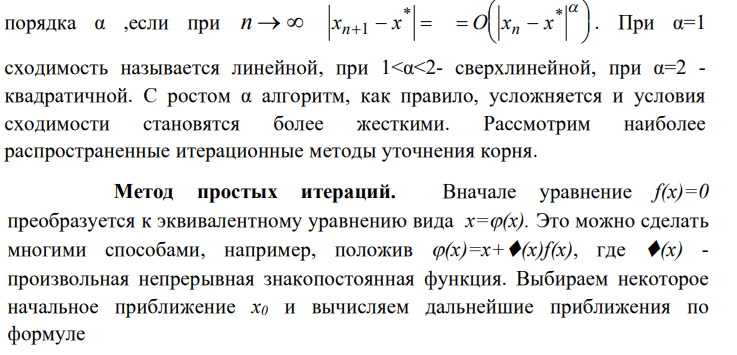


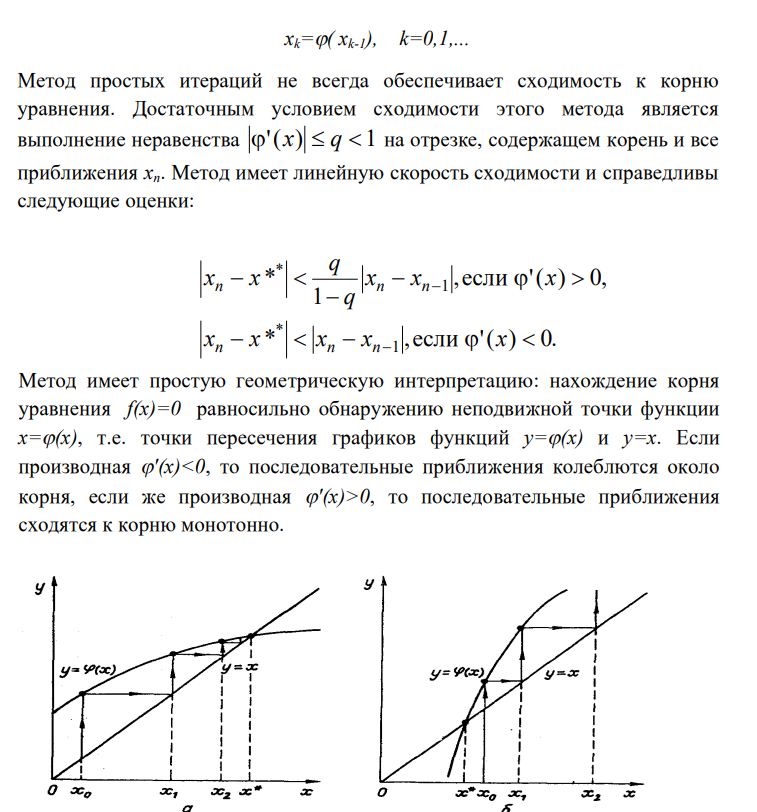


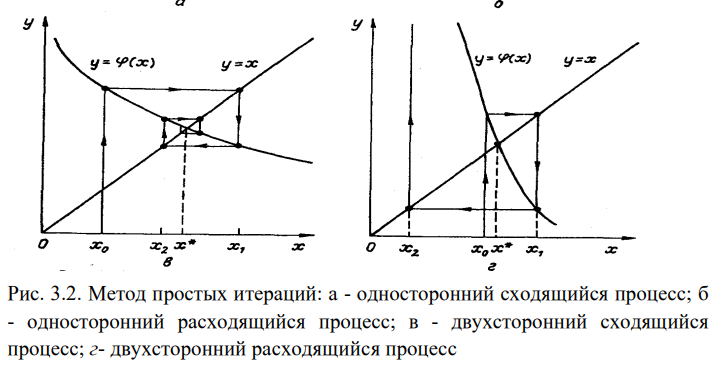


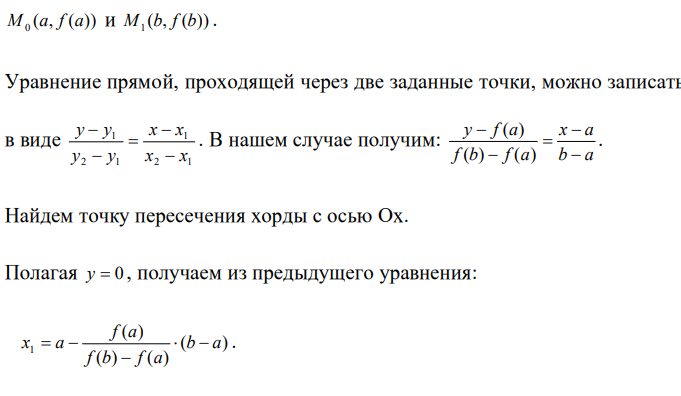
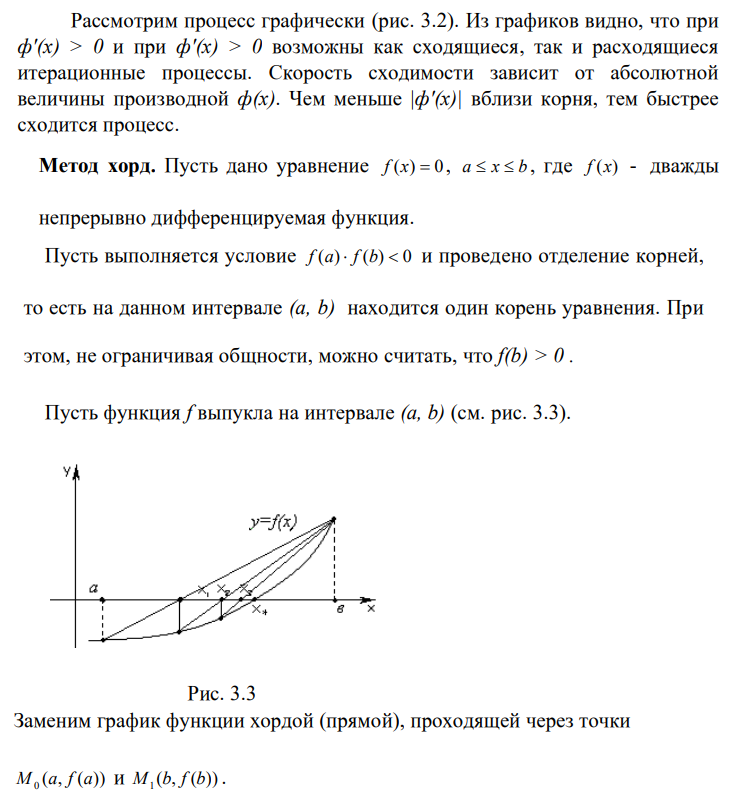


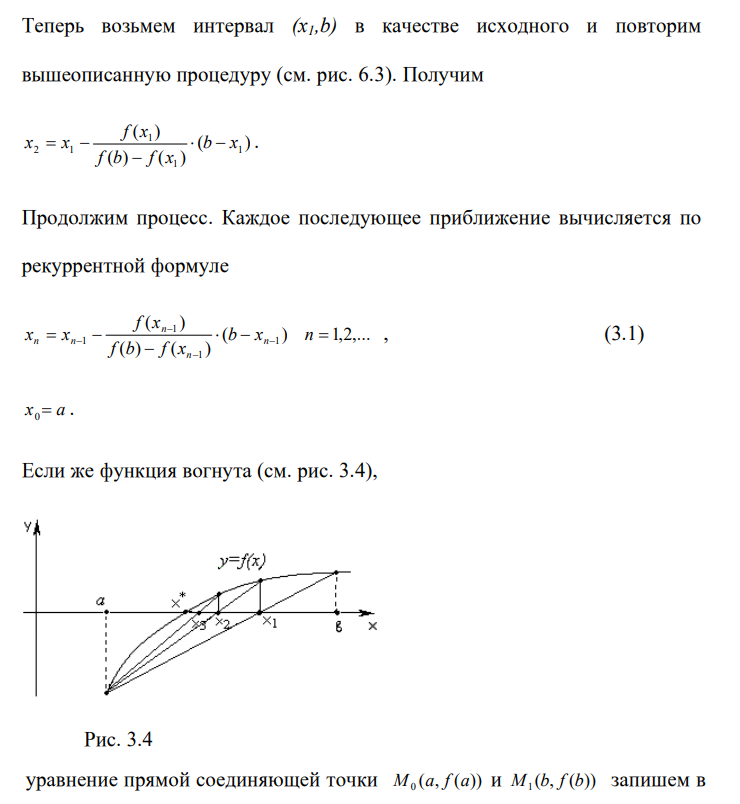


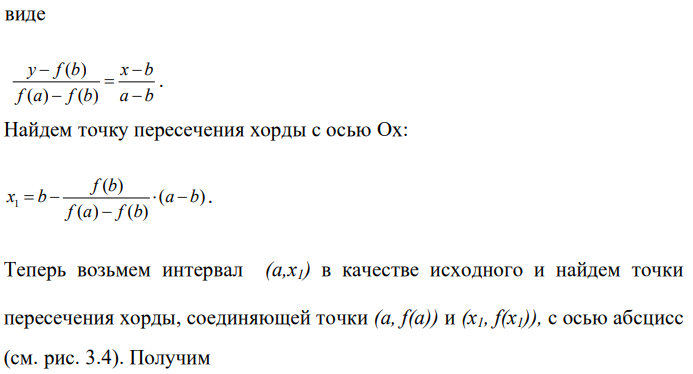


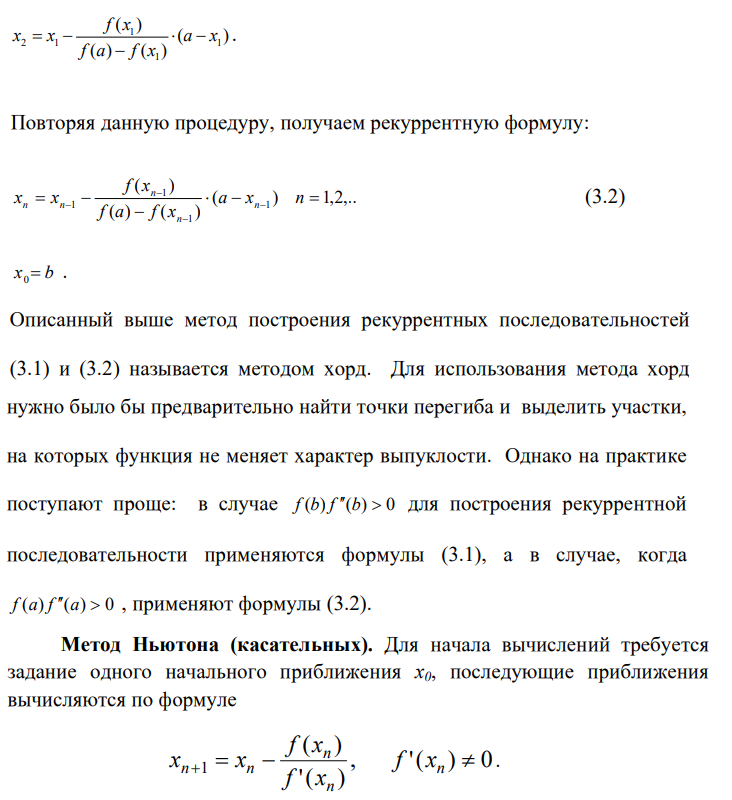


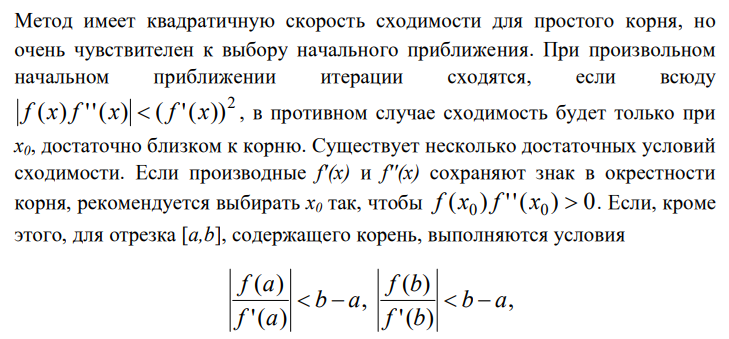


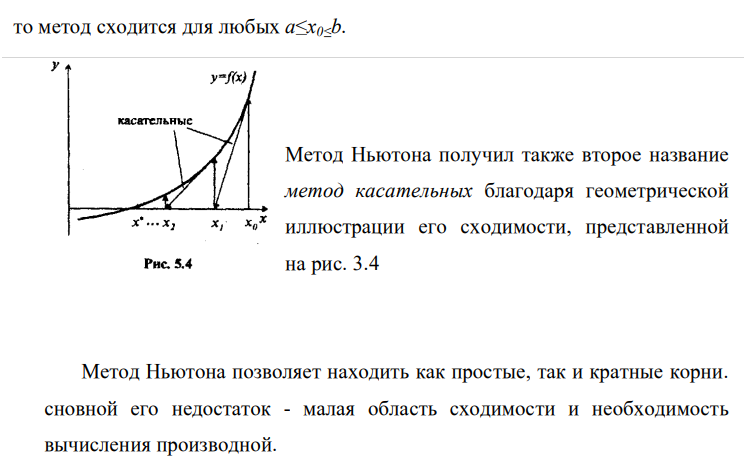




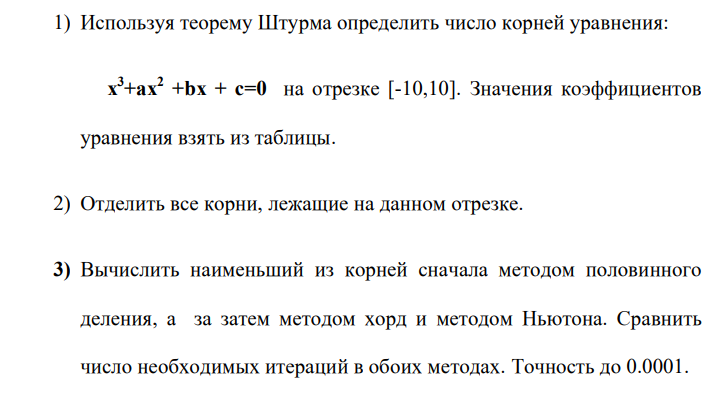




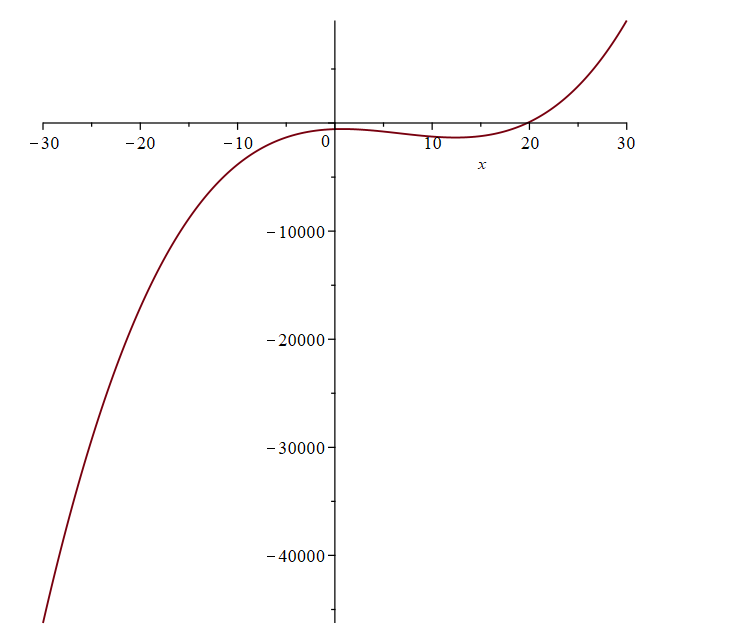




Задание







**Программная реализация**

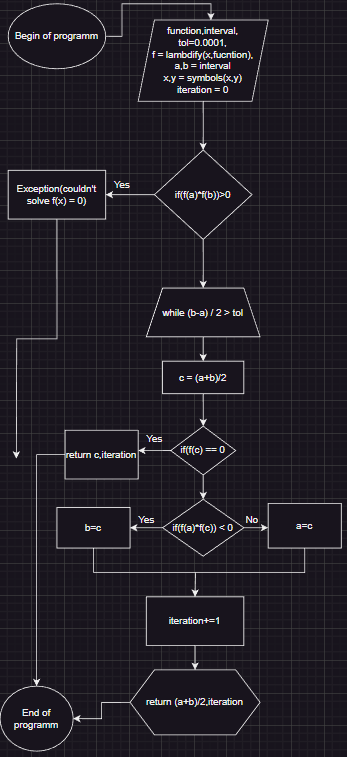


Рисунок 1. Алгоритм половинного деления

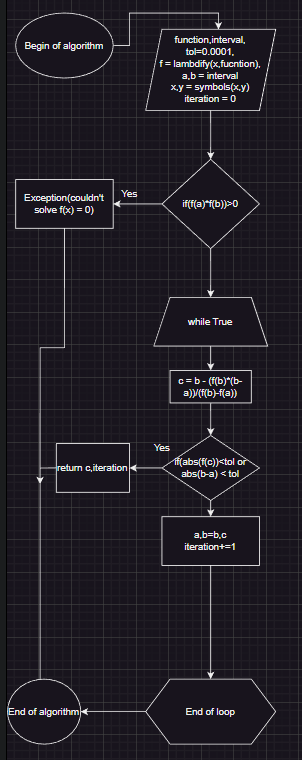


Рисунок 2. Алгоритм метода хорд

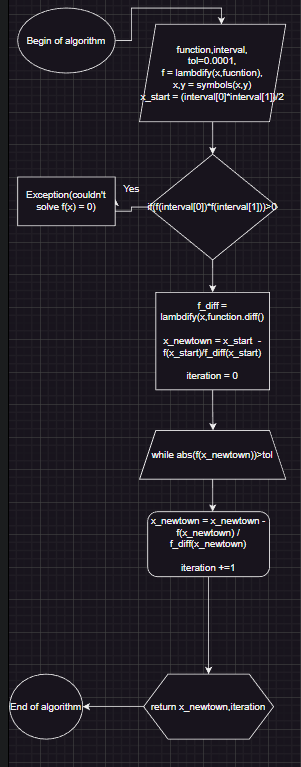


Рисунок 3. Алгоритм метода Ньютона

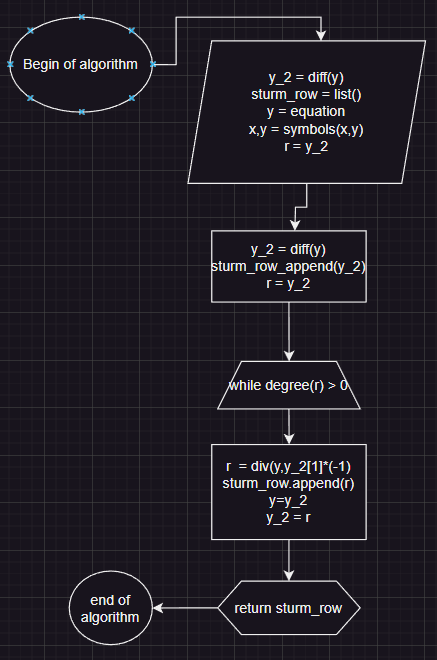


Рисунок 4. Алгоритм получения последовательности Штурма

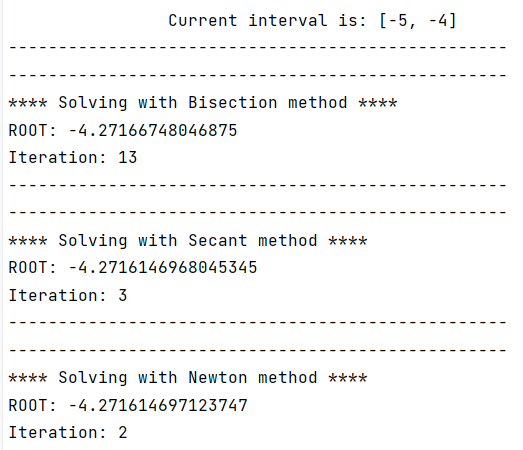
1. nonlinear methods

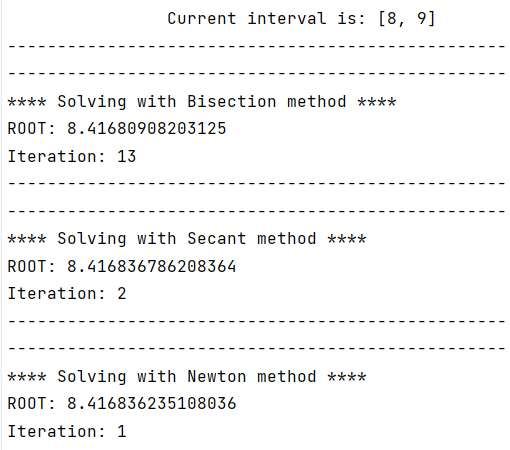
**def bisection(function: Any, interval: tuple, tol=0.0001) -> tuple[float | Any, int]:  
 x, y = symbols('x y')  
 a, b = interval  
 f = lambdify(x, function)  
  
 if f(a) \* f(b) > 0:  
 raise Exception("Bisection method couldn't solve f(x) = 0, because f(a) \* f(b) >= 0\n"  
 "--> more than 1 root or no roots in the interval")  
 iteration = 0  
 while (b - a) / 2 > tol:  
 c = (a + b) / 2  
 if f(c) == 0:  
 return c, iteration  
 if f(a) \* f(c) < 0:  
 b = c  
 else:  
 a = c  
 iteration += 1  
 return (a + b) / 2, iteration**  
  
**def secant(function: Any, interval: tuple, tol=0.0001) -> tuple[float | Any, int]:  
 x, y = symbols('x y')  
 a, b = interval  
 f = lambdify(x, function)  
 if f(a) \* f(b) > 0:  
 raise Exception(f"Secant method couldn't solve f(x) = 0, because f(a) \* f(b) = {f(a) \* f(b)} >= 0\n"  
 f"--> more than 1 root on ({(a, b)}) or no roots)"  
 iteration = 0  
 while True:  
 c = b - (f(b) \* (b - a)) / (f(b) - f(a))  
 if abs(f(c)) < tol or abs(b - a) < tol:  
 return c, iteration  
 a, b = b, c  
 iteration += 1  
  
  
def newton(function: Any, interval: tuple, tol=0.0001) -> tuple[float | Any, int]:  
 x, y = symbols('x y')  
 x\_start = (interval[0] + interval[1]) / 2  
 f = lambdify(x, function)  
 if f(interval[0]) \* f(interval[1]) > 0:  
 raise Exception(  
 f"Newton method couldn't solve f(x) = 0, because f(a) \* f(b) = {f(interval[0]) \* f(interval[1])} >= 0\n"  
 f"--> more than 1 root on ({(interval[0], interval[1])}) or no roots)")  
 f\_diff = lambdify(x, function.diff())  
 x\_newton = x\_start - f(x\_start) / f\_diff(x\_start)  
 iteration = 0  
 while abs(f(x\_newton)) > tol:  
 x\_newton = x\_newton - f(x\_newton) / f\_diff(x\_newton)  
 iteration += 1  
 return x\_newton, iteration**

1. sturms\_row

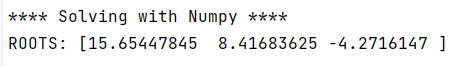
**def traverse\_list(ls, a):  
 if type(ls) == tuple:  
 if ls[0] != 0 or ls[1] != 0:  
 a.append(ls)  
 return  
 for i in range(len(ls)):  
 traverse\_list(ls[i], a)  
  
  
def get\_clean\_intervals(ls, a):  
 traverse\_list(ls, a)  
 return a  
  
def get\_sturm\_row(equation):  
 x, y = symbols('x y')  
 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  
  
def get\_1root\_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\_1root\_small\_interval(sturm\_row, (interval[0], interval[0] + middle)))  
 result\_intervals.append(get\_1root\_small\_interval(sturm\_row, (interval[0] + middle, interval[1])))  
  
 return result\_intervals  
  
def get\_1root\_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\_1root\_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**

**Результаты работы программной реализации**





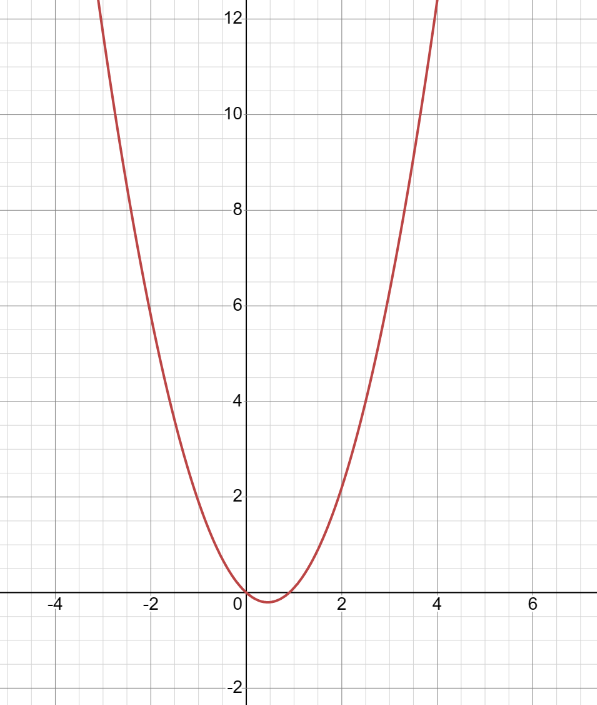
Проверка решения через встроенную функцию

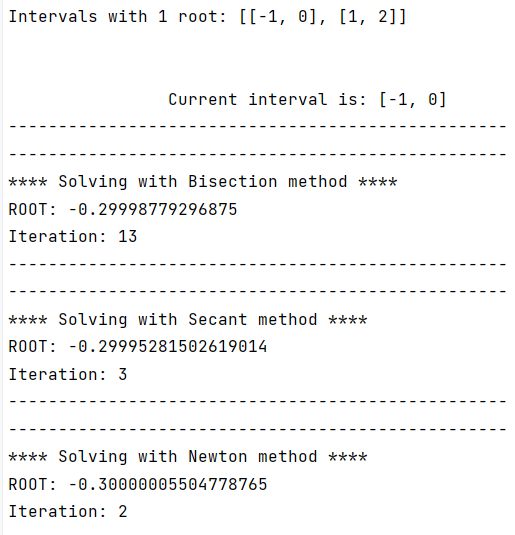


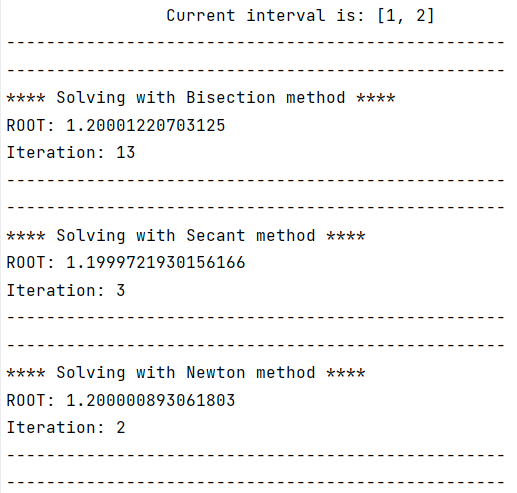
**Тестирование**

1. Тестирование решения квадратного уравнения

****

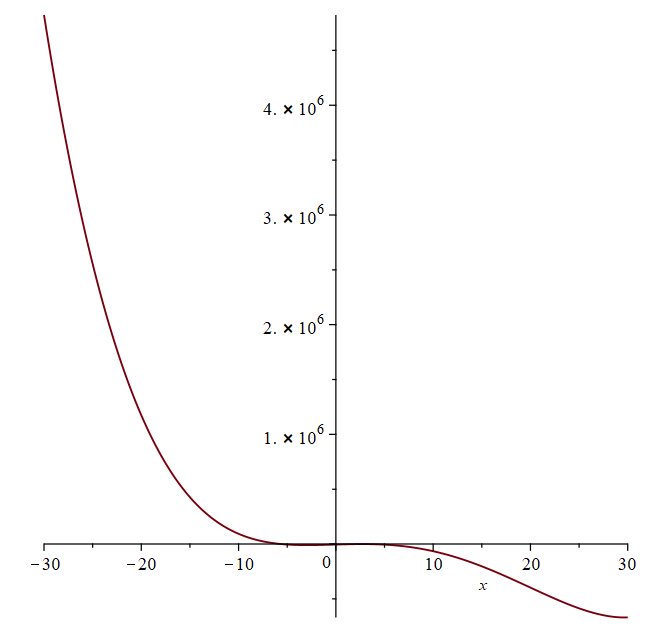
****

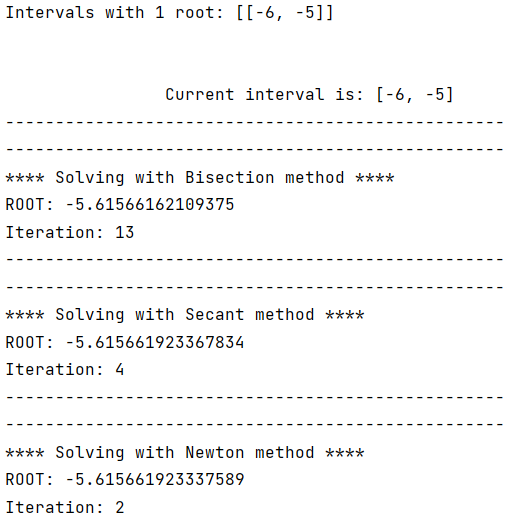
****

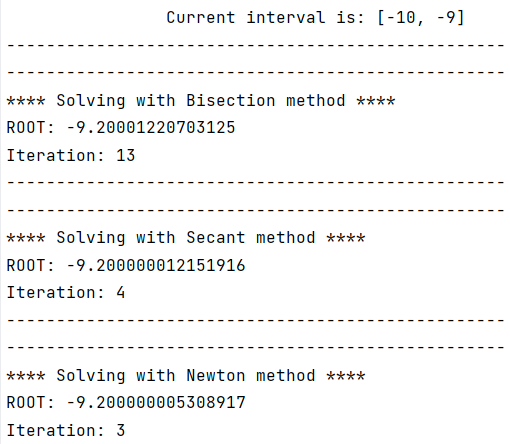
****

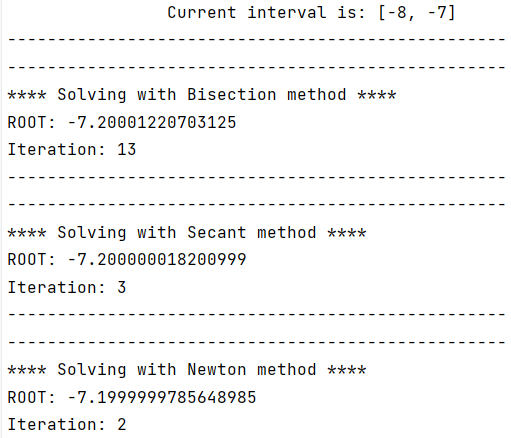
1. Тестирование решения уравнения 4-й степени

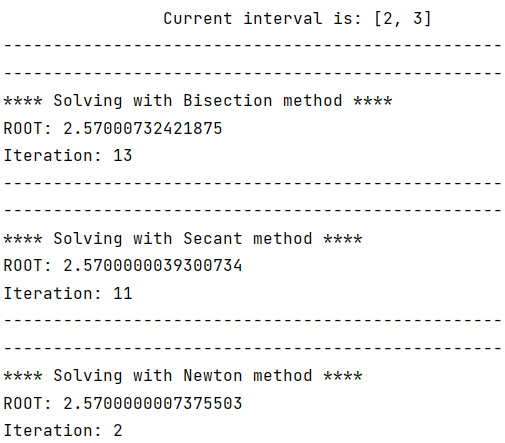




****

****

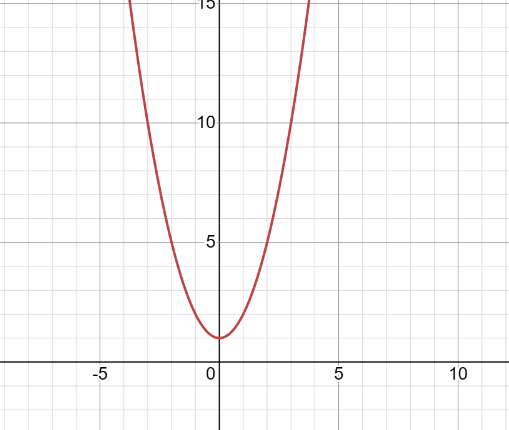
****

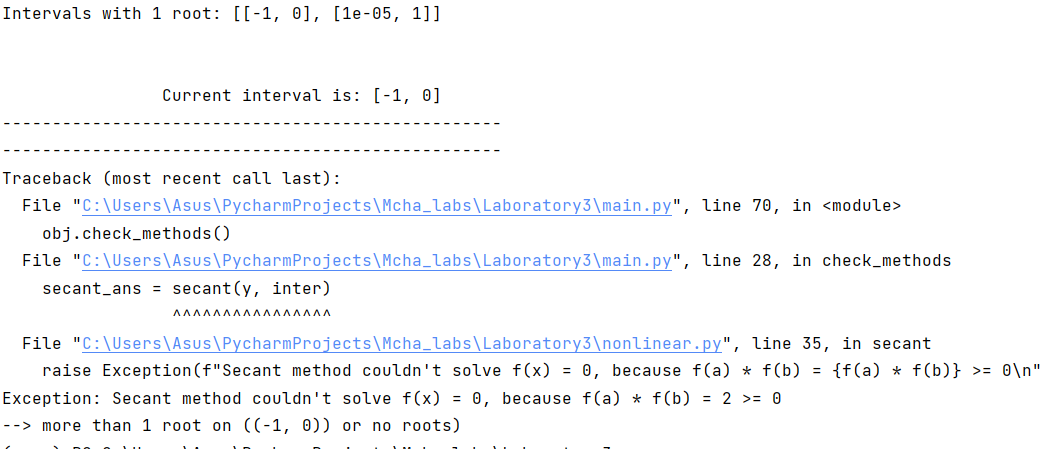
****

Проверка решения через встроенную функцию

****

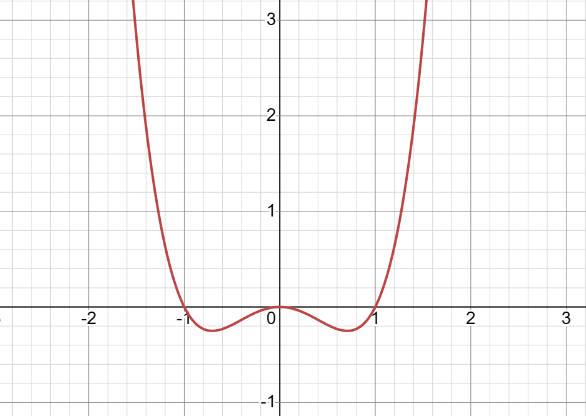
1. Тест на проверку корней на интервале





1. Тест на проверку нескольких корней на интервале



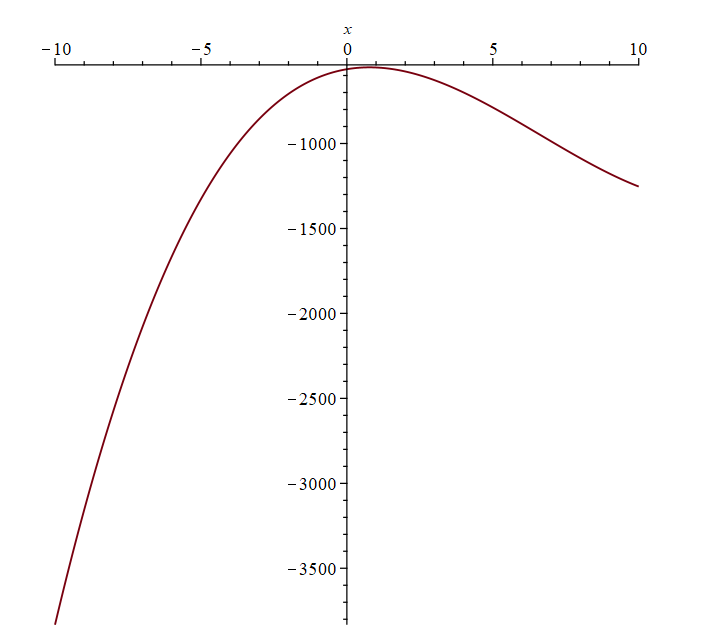




**Оценка скорости сходимости**

Пусть задана функция





|  |  |  |
| --- | --- | --- |
| Метод половинного деления | Метод хорд | Метод Ньютона |
| x1 = -4.27161469680453 | x1 = -4.27166748046875 | x1 = -4.27161469712374 |
| Количество итераций | | |
| 13 | 3 | 2 |

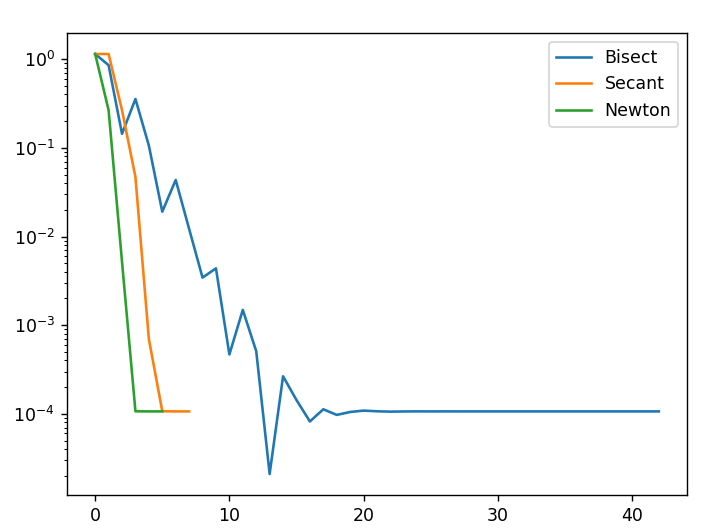
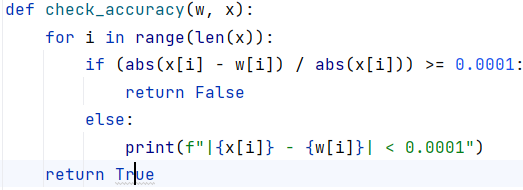


Рисунок 5. График скорости сходимости методов

**Оценка полученных результатов**

****

****

**Выводы**

В ходе выполнения лабораторной работы были изучены методы численного решения нелинейных уравнений (метод простой итерации, метод хорд, метод Ньютона), исследована скорость сходимости итерационных процедур, составлены алгоритмы работы программ, 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 – 21 метода половинного деления.

Метод хорд бывает не всегда оптимален, так же может не найти результата. В ходе работы были рассмотрены 3 функции, имеющих несколько корней на заданном промежутке (-10, 10), и в ходе решения результат был проверен с помощью графика, что дает понять, что реализованные методы успешно справляются с решением нелинейных уравнений