МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра вычислительной математики

Жуковский Павел Сергеевич

Отчёт по лабораторной работе №4

(«Методы вычислений»)

Студента 3 курса 12 группы

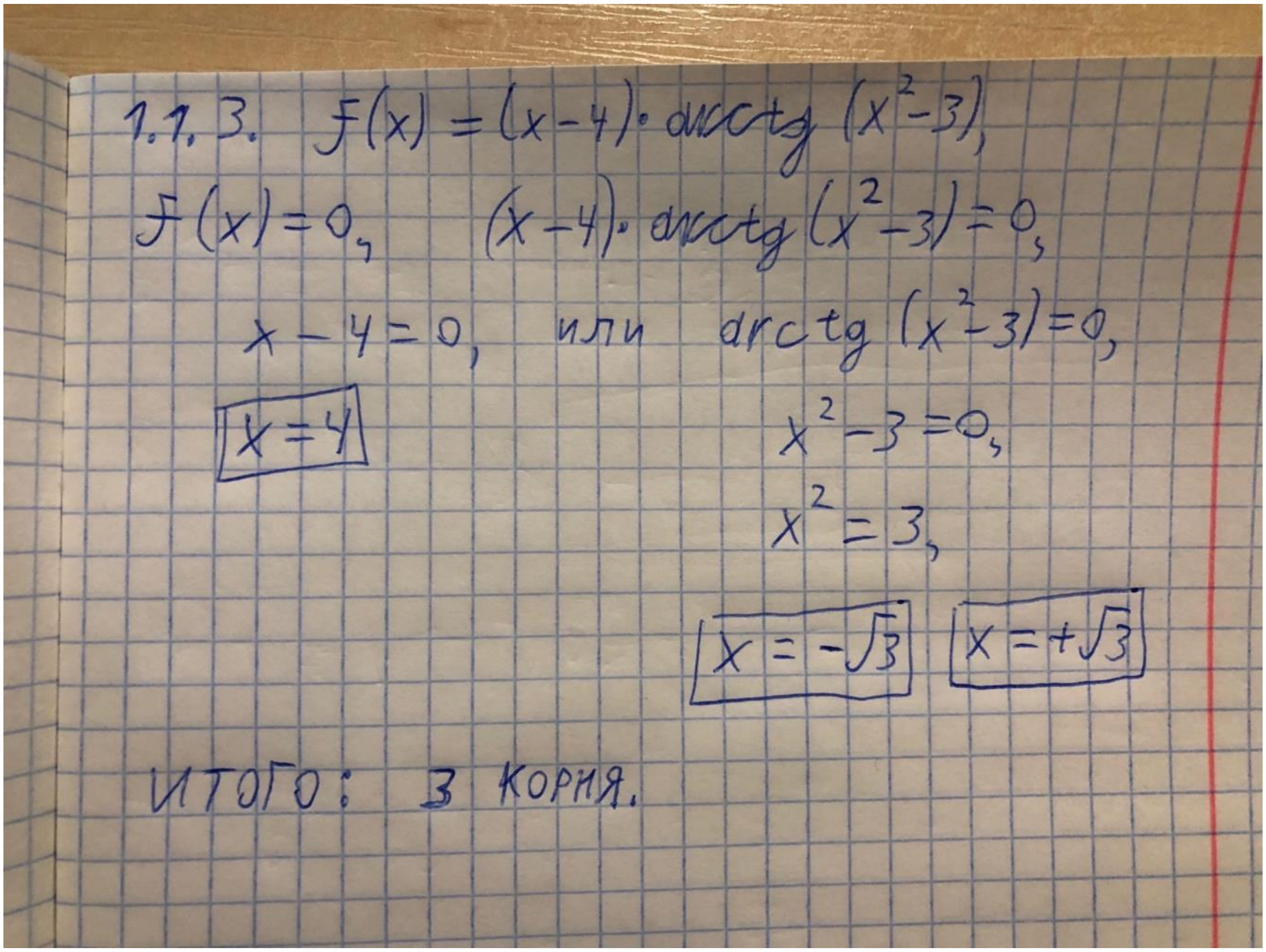
Преподаватель

Бондарь Иван Васильевич

Минск 2020

Задание 1.1.3. $f(x) = (x – 4)*arctg(x^2 – 3)$, метод хорд

После недолгих вычислений на листике, я пришёл к выводу, что данная функция имеет 3 корня:


1.1.3. $f(x) = (x-4)\cdot arctg(x^2-3)$,

$f(x)=0$, $(x-4)\cdot arctg(x^2-3)=0$,

$x-4=0$, или $arctg(x^2-3)=0$,

$\boxed{x=4}$ $x^2-3=0$,

$x^2=3$,

$\boxed{x=-\sqrt{3}}$ $\boxed{x=+\sqrt{3}}$

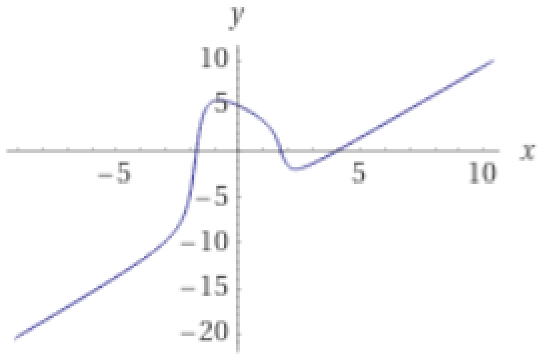
ИТОГО: 3 КОРНЯ.

А именно, это корни: x = -√3, x = √3, x = 4.

На всякий случай я проверил себя с помощью WolframAlpha:

Input:

$f(x) = (x - 4)\tan^{-1}(x^2 - 3)$



(x from −9.1 to 10.3)

Roots:

$x = 4$

$x = -\sqrt{3}$

$x = \sqrt{3}$

Теперь, когда мы знаем корни, вычисленные теоретически и точно на листике выше, перейдём к просмотру кода, который я написал с целью реализации методов Бисекции и Хорд для решения моей задачи. На всякий случай, я оставил в коде комментарии чуть ли ни к каждой строчке. Код я написал на языке программирования Python, где сделал 2 функции для двух моих методов, а потом с помощью этих функций нашёл обоими методами 3 корня на разных отрезках [a, b] (но об этом чуть позже):

```
from math import atan
import numpy as np
import matplotlib.pyplot as plt

# f(x) = (x - 4)*arctg(x^2 - 3)
# Корни: -sqrt(3), sqrt(3), 4.
X_1 = -3**(1/2)
X_2 = 3**(1/2)
X_3 = 4

# Концы отрезка [a; b] для 1-ого корня
a1 = -3.0
b1 = -1.5

# Концы отрезка [a; b] для 2-ого корня
a2 = 1.5
b2 = 3.0

# Концы отрезка [a; b] для 3-его корня
a3 = 3.0
b3 = 6.0

# Точность
Epsilon = 1e-12

# Наша функция
def Foo(X):
    return (X - 4.0)*atan(X**2 - 3.0)
```

```
# 2-ая производная от исходной функции (нужна, чтобы выбрать нужное начальное приближение в методе хорд)
def Foo2(X):
    return 4*X/((3 - X**2)**2 + 1) - (X - 4)/(((-8)*(3 - X**2)*(X**2))/(((3 - X**2)**2 + 1)**2) + (-2)/((3 - X**2)**2 + 1))

# Метод Бисекции
def BisectionMethod(f, a, b, Eps, ResRateArr, needed_X):
    a0 = a  # Запомним, каким a было изначально
    b0 = b  # Запомним, каким b было изначально
    Xk = 0  # Искомое решение
    print("Расчёты итераций для Xk...")
    Count = 0  # Счётчик итераций
    while abs(b - a) > 2.0*Eps:
        Xk = (a + b) / 2.0
        if f(Xk)*f(a) < 0:
            b = Xk
        else:
            a = Xk
        Count += 1  # Делаем инкремент итерации на счётчике
        nevyazka = abs(needed_X - Xk)
        ResRateArr.append(nevyazka)  # Запоминаем невязку на каждой итерации
        print(Count, ") Xk =", Xk, ", |X* - Xk| =", nevyazka)
    print("Корень функции, полученный с помощью метода бисекции на отрезке [", a0, ";", b0, "]:", Xk)
    return Count

# Метод Хорд
def ChordMethod(f, f2, a, b, Eps, ResRateArr, needed_X):
    a0 = a  # Запомним, каким a было изначально
    b0 = b  # Запомним, каким b было изначально
    if np.sign(f(a)) == np.sign(f2(a)):  # Если знак f(a) == f''(a), то берем X0 = a и X1 = b, иначе - берём X0 = b и X1 = a
        X0 = a
        X1 = b
    else:
        X0 = b
        X1 = a
    Xk = X1 - f(X1)*((X1 - X0)/(f(X1) - f(X0)))  #Xk = X2
    Xk_1 = f(4.0) + 1  #Xk+1, просто инициализируем чем-нибудь, отличным от 0
    print("Расчёты итераций для Xk...")
    Count = 0  # Счётчик итераций
    while abs(f(Xk_1)) >= Eps:
        Xk_1 = Xk - f(Xk) * ((Xk - X0) / (f(Xk) - f(X0)))
        Xk = Xk_1
        Count += 1  # Делаем инкремент итерации на счётчике
        nevyazka = abs(needed_X - Xk)
        ResRateArr.append(nevyazka)  # Запоминаем невязку на каждой итерации
        print(Count, ") Xk =", Xk, ", |X* - Xk| =", nevyazka)
    print("Корень функции, полученный с помощью метода хорд на отрезке [", a0, ";", b0, "]:", Xk)
    return Count  # Возвращаем количество итераций, которое нам понадобилось

print("\nФункция: f(x) = (x - 4)*arctg(x^2 - 3)")
print("Корни этой функции, расчитанные теоретически: -sqrt(3), sqrt(3), 4")

print("\nПоиск первого корня методом бисекции на отрезке [", a1, ";", b1, "]")
BM_ResRateArr_1 = []  # Массив ординат (норм невязки на разных итерациях) для графика Метода Бисекции для 1-ого корня
BM_IterAmount_1 = BisectionMethod(Foo, a1, b1, Epsilon, BM_ResRateArr_1, X_1)  # Делаем расчёты и запоминаем количество итераций
BM_IterArr_1 = np.arange(1, BM_IterAmount_1 + 1)  # Массив абсцисс (количества итераций) для графика Метода Бисекции для 1-ого корня

print("\nПоиск второго корня методом бисекции на отрезке [", a2, ";", b2, "]")
BM_ResRateArr_2 = []  # Массив ординат (норм невязки на разных итерациях) для графика Метода Бисекции для 2-ого корня
BM_IterAmount_2 = BisectionMethod(Foo, a2, b2, Epsilon, BM_ResRateArr_2, X_2)
BM_IterArr_2 = np.arange(1, BM_IterAmount_2 + 1)  # Массив абсцисс (количества итераций) для графика Метода Бисекции для 2-ого корня

print("\nПоиск третьего корня методом бисекции на отрезке [", a3, ";", b3, "]")
BM_ResRateArr_3 = []  # Массив ординат (норм невязки на разных итерациях) для графика Метода Бисекции для 3-ого корня
```

```
BM_IterAmount_3 = BisectionMethod(Foo, a3, b3, Epsilon, BM_ResRateArr_3, X_3)
BM_IterArr_3 = np.arange(1, BM_IterAmount_3 + 1)  # Массив абсцисс (количества итераций) для графика Метода Бисекции для 3-ого корня

print("\nПоиск первого корня методом хорд на отрезке [", a1, ";", b1, "]")
CM_ResRateArr_1 = []  # Массив ординат (норм невязки на разных итерациях) для графика Метода Хорд для 1-ого корня
CM_IterAmount_1 = ChordMethod(Foo, Foo2, a1, b1, Epsilon, CM_ResRateArr_1, X_1)
CM_IterArr_1 = np.arange(1, CM_IterAmount_1 + 1)  # Массив абсцисс (количества итераций) для графика Метода Хорд для 1-ого корня

print("\nПоиск второго корня методом хорд на отрезке [", a2, ";", b2, "]")
CM_ResRateArr_2 = []  # Массив ординат (норм невязки на разных итерациях) для графика Метода Хорд для 2-ого корня
CM_IterAmount_2 = ChordMethod(Foo, Foo2, a2, b2, Epsilon, CM_ResRateArr_2, X_2)
CM_IterArr_2 = np.arange(1, CM_IterAmount_2 + 1)  # Массив абсцисс (количества итераций) для графика Метода Хорд для 2-ого корня

print("\nПоиск третьего корня методом хорд на отрезке [", a3, ";", b3, "]")
CM_ResRateArr_3 = []  # Массив ординат (норм невязки на разных итерациях) для графика Метода Хорд для 3-ого корня
CM_IterAmount_3 = ChordMethod(Foo, Foo2, a3, b3, Epsilon, CM_ResRateArr_3, X_3)
CM_IterArr_3 = np.arange(1, CM_IterAmount_3 + 1)  # Массив абсцисс (количества итераций) для графика Метода Хорд для 3-ого корня

# Строим графики сходимостей
plt.semilogy(BM_IterArr_1, BM_ResRateArr_1, label="BM1, [-3.0; -1.5]")
plt.semilogy(BM_IterArr_2, BM_ResRateArr_2, label="BM2, [1.5; 3.0]")
plt.semilogy(BM_IterArr_3, BM_ResRateArr_3, label="BM3, [3.0; 6.0]")
plt.semilogy(CM_IterArr_1, CM_ResRateArr_1, label="CM1, [-3.0; -1.5]")
plt.semilogy(CM_IterArr_2, CM_ResRateArr_2, label="CM2, [1.5; 3.0]")
plt.semilogy(CM_IterArr_3, CM_ResRateArr_3, label="CM3, [3.0; 6.0]")
plt.title("Диаграммы сходимости")
plt.xlabel("Номер итерации")
plt.ylabel("Норма невязки на этой итерации")
plt.legend()
plt.grid(True)
plt.show()
```

Для начала отмечу, что т.к. в моей функции 3 корня, то я подобрал 3 отрезка [a; b], при которых сходятся оба метода (я постарался найти такие 3 отрезка, чтобы оба метода сходились на каждом корне в этих отрезках, что позволит нам более-менее справедливо сравнить их диаграммы сходимости, но о них позже).

Мои отрезки были следующие:

Для корня x = $-\sqrt{3}$, a1 = -3.0, b1 = -1.5, т.е. отрезок [-3.0; -1.5]

Для корня x = $\sqrt{3}$, a2 = 1.5, b2 = 3.0, т.е. отрезок [1.5; 3.0]

Для корня x = 4, a3 = 3.0, b3 = 6.0, т.е. отрезок [3.0; 6.0]

Самое главное, чтобы наши предполагаемые корни были внутри подобранных отрезков [a; b].

Стоит также отметить один нюанс, который касается сходимости в методе Хорд, о котором я нашёл в книжке своего лектора:

Метод хорд

У метода секущих есть недостаток: члены последовательности x_k могут выходить за пределы отрезка локализации $[a, b]$, что может привести к расходимости итерационного процесса.

$\triangleright_5$ Приведите пример, когда при $x_0 = a$ и $x_1 = b$ приближение x_3, построенное по методу секущих, не лежит в $[a, b]$.

Избежать этой проблемы можно, зафиксировав один из концов секущей линии:

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_0}{f(x_k) - f(x_0)}, \qquad (4.17)$$

причем в качестве x_0 нужно брать тот конец отрезка $[a, b]$, в котором знаки f и f'' совпадают.

Т.е. нам важно, чтобы в качестве начального приближения был взято именно тот конец отрезка, в котором совпадают знаки функции и её 2-ой производной. Именно для этого я написал также 2-ую производную моей функцию:

```
# 2-ая производная от исходной функции (нужна, чтобы выбрать нужное начальное приближение в методе хорд)
def Foo2(X):
    return 4*X/((3 - X**2)**2 + 1) - (X - 4)/(((-8)*(3 - X**2)*(X**2))/(((3 - X**2)**2 + 1)**2) + (-2)/((3 - X**2)**2 + 1))
```

Корректность вычисления второй производной я проверил с помощью WolframAlpha:

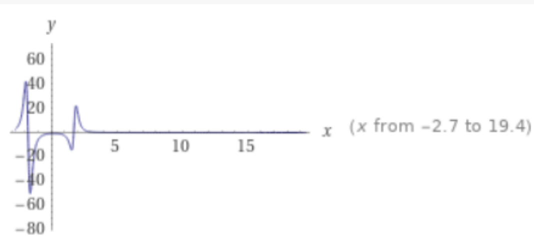
((x-4)*arctg(x^2-3))''

Extended Keyboard   Upload

Derivative:

$$\frac{d^2}{dx^2}((x-4)\tan^{-1}(x^2-3)) = \frac{4x}{(3-x^2)^2+1} - (x-4)\left(-\frac{8(3-x^2)x^2}{((3-x^2)^2+1)^2} - \frac{2}{(3-x^2)^2+1}\right)$$

Plots:



И именно для этого внутри метода Хорда я сделал вот такую проверку на знаки функции и её 2-ой производной в концах отрезка:

```
if np.sign(f(a)) == np.sign(f2(a)):  # Если знак f(a) == f''(a), то берем X0 = a и X1 = b, иначе - берём X0 = b и X1 = a
    X0 = a
    X1 = b
else:
    X0 = b
    X1 = a
```

Кстати говоря, после реализации в коде вышеуказанного нюанса некоторые мои экспериментальные отрезки, которые не сходились до этого, начали сходиться.

Что ж, поговорим о результатах вычислений.

Вот, как сходился X_k в методе Бисекции на отрезке [-3.0; -1.5]:

```
Поиск первого корня методом бисекции на отрезке [ -3.0 ; -1.5 ]
Расчёты итераций для Xk...
1 ) Xk = -2.25 , |X* - Xk| = 0.5179491924311228
2 ) Xk = -1.875 , |X* - Xk| = 0.1429491924311228
3 ) Xk = -1.6875 , |X* - Xk| = 0.044550807568877719
4 ) Xk = -1.78125 , |X* - Xk| = 0.04919919243112281
5 ) Xk = -1.734375 , |X* - Xk| = 0.002324192431122807
6 ) Xk = -1.7109375 , |X* - Xk| = 0.021113307568877193
7 ) Xk = -1.72265625 , |X* - Xk| = 0.009394557568877193
8 ) Xk = -1.728515625 , |X* - Xk| = 0.003535182568877193
9 ) Xk = -1.7314453125 , |X* - Xk| = 0.0006054950688771932
10 ) Xk = -1.73291015625 , |X* - Xk| = 0.0008593486811228068
```

Последние итерации:

```
31 ) Xk = -1.7320508079137653 , |X* - Xk| = 3.448881180645458e-10
32 ) Xk = -1.7320508075645193 , |X* - Xk| = 4.3578474162586645e-12
33 ) Xk = -1.7320508077391423 , |X* - Xk| = 1.7026513532414356e-10
34 ) Xk = -1.7320508076518308 , |X* - Xk| = 8.295364395394245e-11
35 ) Xk = -1.732050807608175 , |X* - Xk| = 3.929789826884189e-11
36 ) Xk = -1.7320508075863472 , |X* - Xk| = 1.7470025426291613e-11
37 ) Xk = -1.7320508075754333 , |X* - Xk| = 6.556089005016474e-12
38 ) Xk = -1.7320508075699763 , |X* - Xk| = 1.099120794378905e-12
39 ) Xk = -1.7320508075672478 , |X* - Xk| = 1.6293633109398797e-12
40 ) Xk = -1.732050807568612 , |X* - Xk| = 2.651212582804874e-13
Корень функции, полученный с помощью метода бисекции на отрезке [ -3.0 ; -1.5 ]: -1.732050807568612
```

Вот, как сходился X_k в методе Бисекции на отрезке [1.5; 3.0]:

```
Поиск второго корня методом бисекции на отрезке [ 1.5 ; 3.0 ]
Расчёты итераций для Xk...
1 ) Xk = 2.25 , |X* - Xk| = 0.5179491924311228
2 ) Xk = 1.875 , |X* - Xk| = 0.1429491924311228
3 ) Xk = 1.6875 , |X* - Xk| = 0.04455080756887719
4 ) Xk = 1.78125 , |X* - Xk| = 0.04919919243112281
5 ) Xk = 1.734375 , |X* - Xk| = 0.002324192431122807
6 ) Xk = 1.7109375 , |X* - Xk| = 0.021113307568877193
7 ) Xk = 1.72265625 , |X* - Xk| = 0.009394557568877193
8 ) Xk = 1.728515625 , |X* - Xk| = 0.003535182568877193
9 ) Xk = 1.7314453125 , |X* - Xk| = 0.0006054950688771932
10 ) Xk = 1.73291015625 , |X* - Xk| = 0.0008593486811228068
```

Последние итерации:

```
31 ) Xk = 1.7320508079137653 , |X* - Xk| = 3.448881180645458e-10
32 ) Xk = 1.7320508075645193 , |X* - Xk| = 4.357847416258645e-12
33 ) Xk = 1.7320508077391423 , |X* - Xk| = 1.7026513532414356e-10
34 ) Xk = 1.7320508076518308 , |X* - Xk| = 8.295364395394245e-11
35 ) Xk = 1.732050807608175 , |X* - Xk| = 3.929789826884189e-11
36 ) Xk = 1.7320508075863472 , |X* - Xk| = 1.7470025426291613e-11
37 ) Xk = 1.7320508075754333 , |X* - Xk| = 6.556089005016474e-12
38 ) Xk = 1.7320508075699763 , |X* - Xk| = 1.099120794378905e-12
39 ) Xk = 1.7320508075672478 , |X* - Xk| = 1.6293633109398797e-12
40 ) Xk = 1.732050807568612 , |X* - Xk| = 2.651212582804874e-13
Корень функции, полученный с помощью метода бисекции на отрезке [ 1.5 ; 3.0 ]: 1.732050807568612
```

Вот, как сходился X_k в методе Бисекции на отрезке [3.0; 6.0]:

```
Поиск третьего корня методом бисекции на отрезке [ 3.0 ; 6.0 ]
Расчёты итераций для Xk...
1 ) Xk = 4.5 , |X* - Xk| = 0.5
2 ) Xk = 3.75 , |X* - Xk| = 0.25
3 ) Xk = 4.125 , |X* - Xk| = 0.125
4 ) Xk = 3.9375 , |X* - Xk| = 0.0625
5 ) Xk = 4.03125 , |X* - Xk| = 0.03125
6 ) Xk = 3.984375 , |X* - Xk| = 0.015625
7 ) Xk = 4.0078125 , |X* - Xk| = 0.0078125
8 ) Xk = 3.99609375 , |X* - Xk| = 0.00390625
9 ) Xk = 4.001953125 , |X* - Xk| = 0.001953125
10 ) Xk = 3.9990234375 , |X* - Xk| = 0.0009765625
```

Последние итерации:

```
30 ) Xk = 3.9999999990686774 , |X* - Xk| = 9.313225746154785e-10
31 ) Xk = 4.000000000465661 , |X* - Xk| = 4.656612873077393e-10
32 ) Xk = 3.9999999997671694 , |X* - Xk| = 2.3283064365386963e-10
33 ) Xk = 4.000000000116415 , |X* - Xk| = 1.1641532182693481e-10
34 ) Xk = 3.9999999999417923 , |X* - Xk| = 5.820766091346741e-11
35 ) Xk = 4.000000000029104 , |X* - Xk| = 2.9103830456733704e-11
36 ) Xk = 3.999999999985448 , |X* - Xk| = 1.4551915228366852e-11
37 ) Xk = 4.000000000007276 , |X* - Xk| = 7.275957614183426e-12
38 ) Xk = 3.999999999996362 , |X* - Xk| = 3.637978807091713e-12
39 ) Xk = 4.000000000001819 , |X* - Xk| = 1.8189894035458565e-12
40 ) Xk = 3.9999999999990905 , |X* - Xk| = 9.094947017729282e-13
41 ) Xk = 4.000000000000455 , |X* - Xk| = 4.547473508864641e-13
Корень функции, полученный с помощью метода бисекции на отрезке [ 3.0 ; 6.0 ]: 4.000000000000455
```

Вот, как сходился X_k в методе Хорд на отрезке [-3.0; -1.5]:

```
Поиск первого корня методом хорд на отрезке [ -3.0 ; -1.5 ]
Расчёты итераций для Xk...
1 ) Xk = -1.7090881602863155 , |X* - Xk| = 0.0229626472825617
2 ) Xk = -1.7395594462139599 , |X* - Xk| = 0.007508638645082666
3 ) Xk = -1.7298454847762852 , |X* - Xk| = 0.0022053227925920016
4 ) Xk = -1.732721862304251 , |X* - Xk| = 0.0006710547353738772
5 ) Xk = -1.7318487190743566 , |X* - Xk| = 0.00020208849452063582
6 ) Xk = -1.7321118590708604 , |X* - Xk| = 6.105150198321141e-05
7 ) Xk = -1.7320323812614926 , |X* - Xk| = 1.8426307384578067e-05
8 ) Xk = -1.7320563705167455 , |X* - Xk| = 5.5629478683538736e-06
9 ) Xk = -1.7320491282466859 , |X* - Xk| = 1.6793221913058431e-06
10 ) Xk = -1.732051314529734 , |X* - Xk| = 5.069608568497586e-07
```

Последние итерации:

```
15 ) Xk = -1.7320508062978273 , |X* - Xk| = 1.2710499319723567e-09
16 ) Xk = -1.732050807952585 , |X* - Xk| = 3.8370773225437915e-10
17 ) Xk = -1.7320508074530427 , |X* - Xk| = 1.1583445314045093e-10
18 ) Xk = -1.7320508076038457 , |X* - Xk| = 3.496847256201363e-11
19 ) Xk = -1.732050807558321 , |X* - Xk| = 1.0556222562740913e-11
20 ) Xk = -1.732050807572064 , |X* - Xk| = 3.1867841698840493e-12
21 ) Xk = -1.7320508075679153 , |X* - Xk| = 9.618972285352356e-13
22 ) Xk = -1.7320508075691676 , |X* - Xk| = 2.9043434324194095e-13
23 ) Xk = -1.7320508075687897 , |X* - Xk| = 8.748557434046234e-14
24 ) Xk = -1.7320508075689036 , |X* - Xk| = 2.6423307986078726e-14
Корень функции, полученный с помощью метода хорд на отрезке [ -3.0 ; -1.5 ]: -1.7320508075689036
```

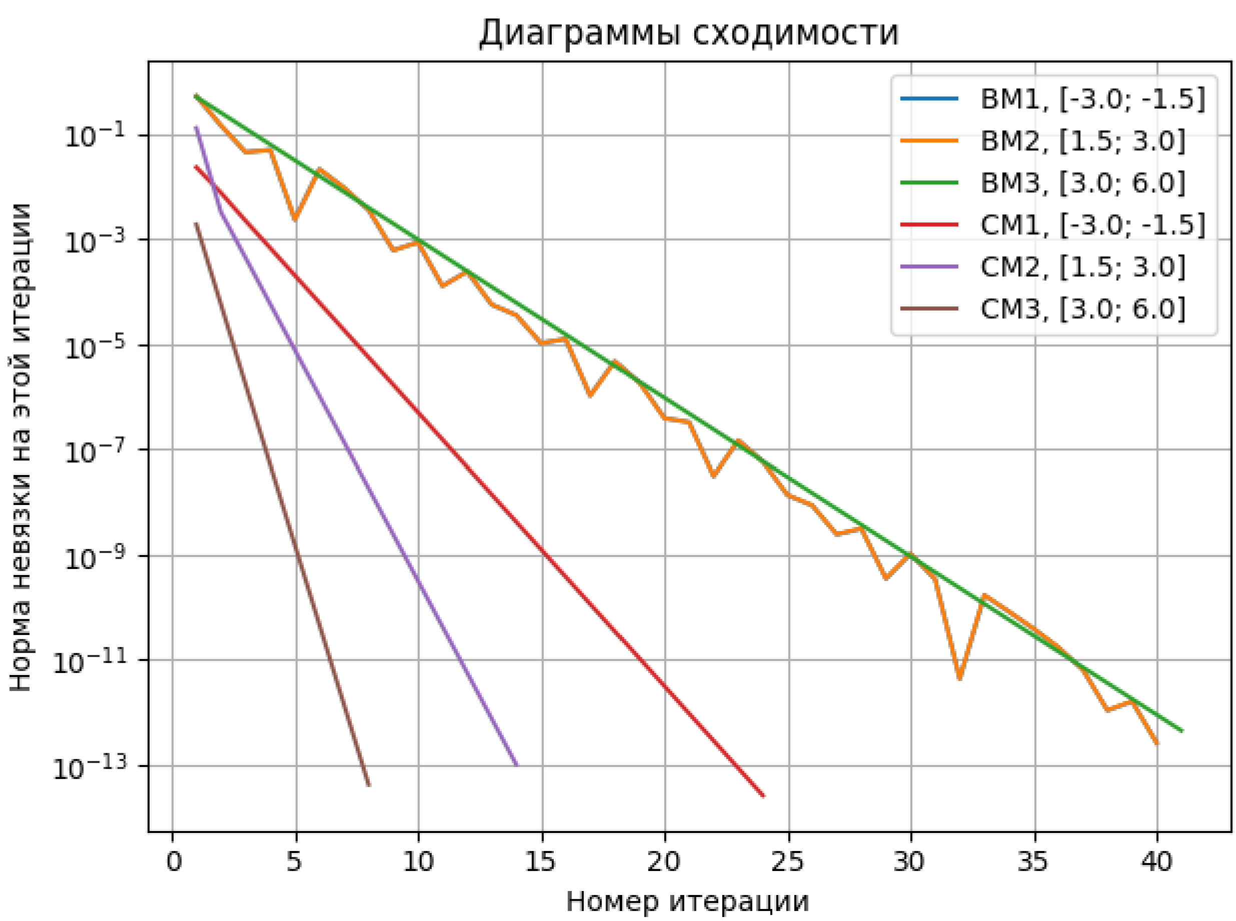
Вот, как сходился X_k в методе Хорд на отрезке [1.5; 3.0]:

```
Поиск второго корня методом хорд на отрезке [ 1.5 ; 3.0 ]
Расчёты итераций для Xk...
1 ) Xk = 1.859815404071829 , |X* - Xk| = 0.1277645965029519
2 ) Xk = 1.7288368534363898 , |X* - Xk| = 0.0032139541324873733
3 ) Xk = 1.7324874838576845 , |X* - Xk| = 0.0004366762888072806
4 ) Xk = 1.7319927865904978 , |X* - Xk| = 5.802097837936948e-05
5 ) Xk = 1.7320585404229147 , |X* - Xk| = 7.732854037545422e-06
6 ) Xk = 1.7320497773768788 , |X* - Xk| = 1.0301919983746188e-06
7 ) Xk = 1.7320509448213088 , |X* - Xk| = 1.372524316423096e-07
8 ) Xk = 1.7320507892828743 , |X* - Xk| = 1.8286002934075896e-08
9 ) Xk = 1.7320508100051053 , |X* - Xk| = 2.436228108138039e-09
10 ) Xk = 1.7320508072443008 , |X* - Xk| = 3.245763657844236e-10
11 ) Xk = 1.7320508076121204 , |X* - Xk| = 4.324318680914985e-11
12 ) Xk = 1.732050807563116 , |X* - Xk| = 5.761169319384862e-12
13 ) Xk = 1.732050807569645 , |X* - Xk| = 7.678302438307583e-13
14 ) Xk = 1.732050807568775 , |X* - Xk| = 1.021405182655144e-13
Корень функции, полученный с помощью метода хорд на отрезке [ 1.5 ; 3.0 ]: 1.732050807568775
```

Вот, как сходился X_k в методе Хорд на отрезке [3.0; 6.0]:

```
Поиск третьего корня методом хорд на отрезке [ 3.0 ; 6.0 ]
Расчёты итераций для Xk...
1 ) Xk = 3.9981270992663966 , |X* - Xk| = 0.0018729007336033554
2 ) Xk = 3.999943437279083 , |X* - Xk| = 5.656272091680847e-05
3 ) Xk = 3.9999982934088645 , |X* - Xk| = 1.7065911355196306e-06
4 ) Xk = 3.999999948510804 , |X* - Xk| = 5.1489196017939776e-08
5 ) Xk = 3.999999998446532 , |X* - Xk| = 1.5534680208872942e-09
6 ) Xk = 3.999999999953131 , |X* - Xk| = 4.686917520757561e-11
7 ) Xk = 3.999999999998586 , |X* - Xk| = 1.4139800441625994e-12
8 ) Xk = 3.9999999999999574 , |X* - Xk| = 4.263256414560601e-14
Корень функции, полученный с помощью метода хорд на отрезке [ 3.0 ; 6.0 ]: 3.9999999999999574
```

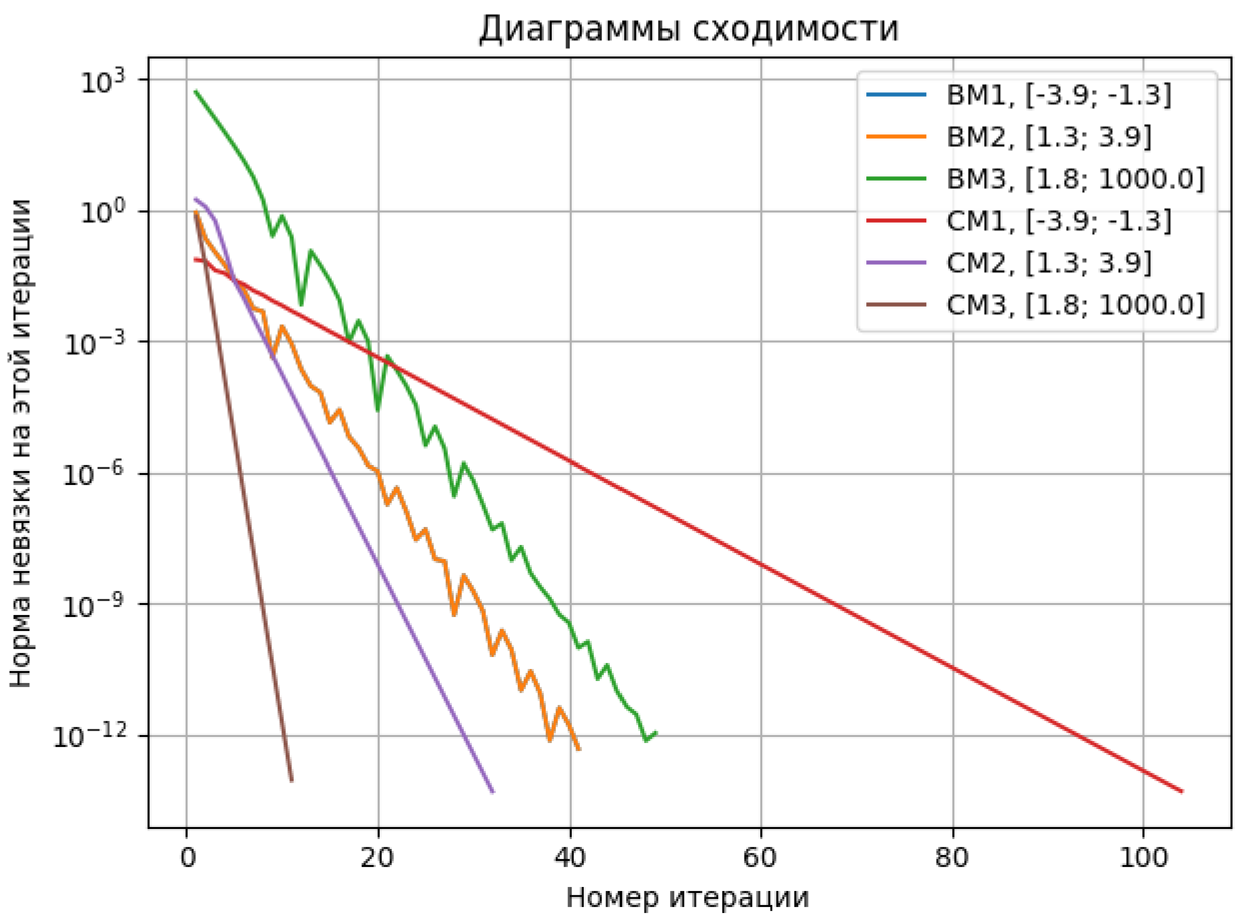
Также посмотрим на графики сходимостей, где отображена невязка каждого метода для каждого корня (на каждом отрезке) относительно текущего номера итерации (здесь BM – означает метод Бисекции, CM – метод Хорд, а цифры после них означают номер корня (отрезка), на котором этот метод выполнялся):



На этом графике отображены отношения невязок к номеру итерации для всех методов на всех отрезках (кстати, здесь совсем не видно голубой график для

ВМ1, но это не удивительно, т.к. там невязки точно такие же, как и в ВМ2, потому что эти методы искали два противоположных корня, а именно -$\sqrt{3}$ и $\sqrt{3}$). Этот график я прикреплю в письме под именем Task_1_1_3_Графики_Сходимостей.jpg.

Можно сказать, что оба метода нашли корни с нужной точностью (1e-12), однако на графиках видно, что метод Хорд сошёлся быстрее на заданных отрезках. Я также делал эксперименты и на других отрезках, и вот, например, были и такие ситуации:



А если брать b1 ⊂ [-1.2; 0.0] и a1 ⊂ [0.0; 1.2], то метод Хорд вообще расходится:

```
88602 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88603 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.05950319642570600 6
88604 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88605 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88606 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88607 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88608 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88609 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88610 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88611 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88612 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88613 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88614 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88615 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88616 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88617 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88618 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88619 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88620 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88621 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88622 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88623 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88624 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88625 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88626 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88627 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88628 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88629 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88630 ) Xk = -1.804187089382421 , |X* - Xk| = 0.07213628181354381
88631 ) Xk = -1.6725476111431712 , |X* - Xk| = 0.059503196425706006
88632
```

Так что, хоть метод Хорд и сходится немного быстрее, но для него нужно ещё подобрать хорошие приближения, иначе он либо будет сходиться медленно, либо вообще будет расходиться.

Файл task_1_1_3.py с исходным кодом я прикреплю также в письме.

Задание 2.3. Рассмотрим систему уравнений:

$$f(x,y) = \begin{pmatrix} (x - c)^2 + y^2 - 1 \\ x^2 + (y-c)^2 - 100 \end{pmatrix} = 0$$

Итак, я подробно изучил вышеупомянутые функции и то, как они изменяются в зависимости от параметра с. Обе эти функции являются окружностями, при чём первая с радиусом 1, а вторая – с радиусом 10. Более того, при увеличении параметра с первая окружность уходит вправо на графике (ровно по оси O_x), а вторая окружность уходит вверх на графике (ровно по оси O_y), а при уменьшении – первая окружность уходит уже влево, а вторая окружность уходит уже вниз.

Чтобы наглядно представить это, я построил эту ситуацию в программе Geogebra, где построил обе окружности по уравнениям и где добавил ползунок для параметра с, результаты моих экспериментов можно кратно изобразить на следующем gif-файле (этот файлик я также оставлю в письме с именем Наглядное_движение_окружностей.gif):

https://s8.gifyu.com/images/Task_2_3_KORNI_FUNKTII_PRI_RAZNYK_c.gif

На графиках чётко видно, что:

1) **Есть всего 3 диапазона для параметра с, при которых нету корней**: маленькая окружность слева сверху от большой, маленькая окружность внутри большой, маленькая окружность справа снизу от большой;

2) **Есть 4 значения для параметра с, при которых есть только один корень**, а именно это случаи, когда окружности касаются: маленькая окружность касается большой внешне слева сверху, маленькая окружность касается большой внутренне слева сверху, маленькая окружность касается большой внутренне справа снизу, маленькая окружность касается большой внешне справа снизу.

3) **Есть 2 диапазона для параметра с, при которых есть два корня**, а именно: когда маленькая окружность пересекается с большой слева сверху, когда маленькая окружность пересекается с большой справа снизу.

В своей лабораторной работе я решил рассмотреть по одному значению параметра с для каждого из 3-ёх вышеупомянутых случаев: 1) одно значение с = 7, при котором наша система имеет 2 корня; 2) одно значение с = $\frac{11\sqrt{2}}{2}$, при котором наша система имеет ровно один корень; 3) одно значение с = 10, при котором наша система вообще не имеет корней. Для каждого из этих случаев я предоставлю письменные доказательства своих вычислений, а также сверю их с показаниями WolframAlpha и с результатами вывода моей программы.

Для начала покажу код моей второй программы. Её я также сделал на языке Python, а также подписал комментарии чуть ли не к каждой строчке (на всякий случай). Метод Ньютона я реализовал в отдельном методе. Код программы (будет прикреплен в письме под именем task_2_3.py):

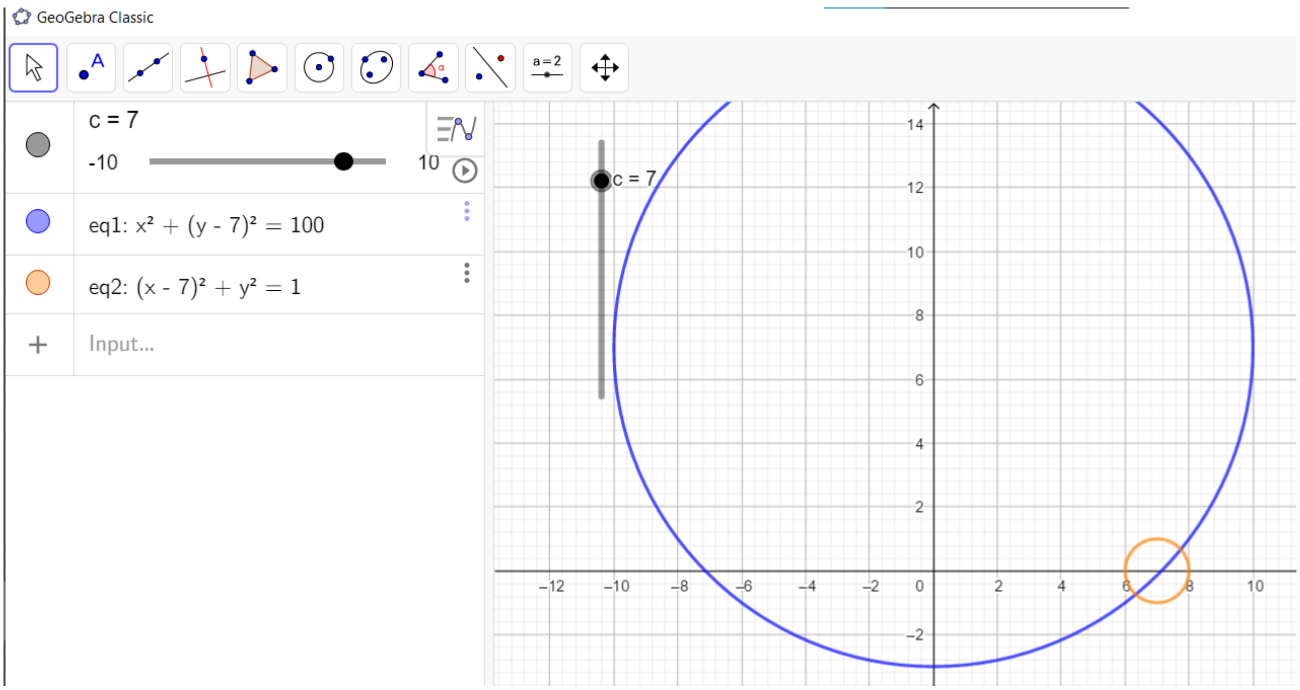
```
from math import sqrt
import numpy as np
import matplotlib.pyplot as plt

# Точность
Epsilon = 1e-12

# Пусть c = 7.0, в этом случае, согласно теории, есть 2 корня
c = 7.0  # Параметр c, при котором есть 2 корня
X1 = (1379.0 - sqrt(19159.0))/196.0  # Первый корень (расчитан теоретически на листочке)
X2 = (1379.0 + sqrt(19159.0))/196.0  # Второй корень (расчитан теоретически на листочке)

# Наша функция, зависящая от X, после того, как мы подставили параметр c в систему, и подстановкой выразили Y через X
def Foo1(X):
    return 392.0*X**2 - 5516.0*X + 19209.0

# Первая производная от вышеуказанной функции
def Foo2(X):
    return 784.0*X - 5516.0

# Метод Ньютона
def NyutonMethod(X0, f, f1, Eps, ResRateArr, needed_X, solutions_exist):
    Xk = X0  # Искомое решние
    Xk_1 = Xk + 1.0  # Xk+1
    Count = 0  # Счётчик итераций
    nevyazka = abs(needed_X - X0)  # Невязка
    first_iteration = True  # Флажок для первой итерации
    print("Расчёты итераций для Xk...")
    while abs(Xk_1 - Xk) >= Eps:
        if first_iteration:
            first_iteration = False
        else:
            Xk = Xk_1
        Xk_1 = Xk - f(Xk)/f1(Xk)  # Xk+1 = Xk - f(Xk) / f'(Xk)
        Count += 1  # Делаем инкремент итерации на счётчике
        nevyazka = abs(needed_X - Xk_1)
        ResRateArr.append(nevyazka)  # Запоминаем невязку на каждой итерации
        print(Count, ") Xk =", Xk_1, ", |X* - Xk| =", nevyazka)
        if Count == 30 and not solutions_exist:
            print("За 30 итераций метод Ньютона так и не смог найти никаких корней...")
            return Count
    print("Корень функции, полученный с помощью метода Ньютона с начальным приближением X0 =", X0, ", равен", Xk)
    return Count

print("\nПараметр c =", c, "(один из множества случаев, когда система имеет 2 корня)")
print("Вид функции при c =", c, ": f(x) = 392x^2 - 5516x + 19209, f'(x) = 784x - 5516")
print("Корни этой функции, расчитанные теоретически:", X1, ",", X2)

X_0 = 4.0
print("\nПоиск первого корня методом Ньютона с начальным приближением X0 =", X_0)
NM_ResRateArr_1 = []  # Массив ординат (норм невязки на разных итерациях) для графика Метода Ньютона для 1-ого корня
NM_IterAmount_1 = NyutonMethod(X_0, Foo1, Foo2, Epsilon, NM_ResRateArr_1, X1, True)  # Делаем расчёты и запоминаем количество итераций
NM_IterArr_1 = np.arange(1, NM_IterAmount_1 + 1)  # Массив абсцисс (количества итераций) для графика Метода Ньютона для 1-ого корня

X_0 = 10.0
print("\nПоиск второго корня методом Ньютона с начальным приближением X0 =", X_0)
```

```
NM_ResRateArr_2 = []  # Массив ординат (норм невязки на разных итерациях) для графика Метода Ньютона для 2-ого корня
NM_IterAmount_2 = NyutonMethod(X_0, Foo1, Foo2, Epsilon, NM_ResRateArr_2, X2, True)  # Делаем расчёты и запоминаем количество итераций
NM_IterArr_2 = np.arange(1, NM_IterAmount_2 + 1)  # Массив абсцисс (количества итераций) для графика Метода Ньютона для 2-ого корня

# Пусть c = 11sqrt(2)/2, в этом случае, согласно теории, есть толькой один корень (это один из четырёх таких случаев)
c = sqrt(121.0 / 2.0)  # Параметр c, при котором есть только один корень
X1 = 5*sqrt(2)  # Первый и единственый корень (расчитан теоретически на листочке)

# Наша функция, зависящая от X, после того, как мы подставили параметр c в систему, и подстановкой выразили Y через X
def Foo3(X):
    return X**2 - 10*sqrt(2)*X + 50

# Первая производная от вышеуказанной функции
def Foo4(X):
    return 2*X - 10*sqrt(2)

print("\nПараметр c =", c, "(один из четырёх случаев, когда система имеет только один корень)")
print("Вид функции при c =", c, ": f(x) = x^2 - 10sqrt(2) + 50, f'(x) = 2x - 10sqrt(2)")
print("Корень этой функции, расчитанный теоретически:", X1)

X_0 = 2.0
print("\nПоиск первого и единственного корня методом Ньютона с начальным приближением X0 =", X_0)
NM_ResRateArr_3 = []  # Массив ординат (норм невязки на разных итерациях) для графика Метода Ньютона для 1-ого корня
NM_IterAmount_3 = NyutonMethod(X_0, Foo3, Foo4, Epsilon, NM_ResRateArr_3, X1, True)  # Делаем расчёты и запоминаем количество итераций
NM_IterArr_3 = np.arange(1, NM_IterAmount_3 + 1)  # Массив абсцисс (количества итераций) для графика Метода Ньютона для 1-ого корня

# Пусть c = 10, в этом случае, согласно теории, корней вообщей нету (дискриминант отрицательный, пересечений нету)
c = 10.0  # Параметр c, при котором нет корней

# Наша функция, зависящая от X, после того, как мы подставили параметр c в систему, и подстановкой выразили Y через X
def Foo5(X):
    return 800*X**2 - 11960*X + 49401

# Первая производная от вышеуказанной функции
def Foo6(X):
    return 1600*X - 11960

print("\nПараметр c =", c, "(один из множества случаев, когда система вообще не имеет корней)")
print("Вид функции при c =", c, ": f(x) = 800x^2 - 11960x + 49401, f'(x) = 1600x - 11960")

X_0 = 5.0
print("\nПопытки найти какие-то корни Методом Ньютона с начальным приближением X0 =", X_0)
NM_ResRateArr_4 = []  # Массив ординат (норм невязки на разных итерациях) для графика Метода Ньютона
NM_IterAmount_4 = NyutonMethod(X_0, Foo5, Foo6, Epsilon, NM_ResRateArr_4, X1, False)  # Делаем расчёты и запоминаем количество итераций
NM_IterArr_4 = np.arange(1, NM_IterAmount_4 + 1)  # Массив абсцисс (количества итераций) для графика Метода Ньютона

# Строим графики сходимостей
plt.semilogy(NM_IterArr_1, NM_ResRateArr_1, label="NM1, c = 7.0 (1-ый корень из двух")
plt.semilogy(NM_IterArr_2, NM_ResRateArr_2, label="NM2, c = 7.0 (2-ой корень из двух)")
plt.semilogy(NM_IterArr_3, NM_ResRateArr_3, label="NM3, c = 11sqrt(2)/2 (единственный корень)")
plt.semilogy(NM_IterArr_4, NM_ResRateArr_4, label="NM4, c = 10 (нет корней)")
plt.title("Диаграммы сходимости")
plt.xlabel("Номер итерации")
plt.ylabel("Норма невязки на этой итерации")
plt.legend()
plt.grid(True)
plt.show()
```

Итак, теперь рассмотрим, что же было в каждом из трёх моих случаев, сходились ли корни, были ли погрешности и как себя показали графики.

Случай 1 – система имеет 2 корня

В качестве параметра c, при котором система имеет 2 корня, я просто взял какое-нибудь целое число, а именно 7, при котором графики окружностей (изображенных в программе GeoGebra) пересекаются:



Таким образом, при параметре c = 7 наша система, по идее, должна иметь 2 корня (т.к. на графике 2 пересечения окружностей). Кстати, можно было взять и какое-нибудь другое расположение окружностей с двумя пересечениями (например, слева сверху), но я для удобства взял целочисленное значение параметра c.

Чтобы доказать мою гипотезу, я письменно в тетради подставил c = 7 в оба уравнения моей системы, затем выразил во втором уравнении y через x и подставил в первое уравнение, после чего привёл его к привычному виду (к уравнению параболы) и затем, посчитав дискриминант (который оказался больше нуля), я нашёл 2 корня (хотя, они получились довольно «страшенькими»), а ещё нашёл производную моей функции (она пригодится в методе Ньютона):

$$\begin{cases} x^2+(y-c)^2=100, \\ (x-c)^2+y^2=1, \end{cases}$$

$\boxed{\text{СЛУЧАЙ С ДВУМЯ КОРНЯМИ}}$

$c=7,$ $\begin{cases} x^2+(y-7)^2=100, \\ (x-7)^2+y^2=1, \end{cases}$ $\begin{cases} x^2+y^2-14y=51, \\ y^2=1-(x-7)^2, \end{cases}$

$\begin{cases} x^2+y^2-14y=51, \\ y=\pm\sqrt{1-(x-7)^2}, \end{cases}$ $\begin{cases} x^2+(1-(x-7)^2)-14y=51, \\ y=\pm\sqrt{1-(x-7)^2}, \end{cases}$

ДВА СЛУЧАЯ:

1) $x^2+1-x^2+14x-49-14\sqrt{1-(x-7)^2}=51,$

$14x-14\sqrt{1-(x-7)^2}=99,$

$-14\sqrt{1-(x-7)^2}=99-14x,$

2) $x^2+1-x^2+14x-49+14\sqrt{1-(x-7)^2}=51,$

$14x+14\sqrt{1-(x-7)^2}=99,$

$14\sqrt{1-(x-7)^2}=99-14x,$

$$\left(-14\sqrt{1-(x-7)^2}\right)^2=(99-14x)^2 \quad \Bigg| \quad \left(14\sqrt{1-(x-7)^2}\right)^2=(99-14x)^2$$

$$196\cdot\left(1-(x-7)^2\right)=9801-2772x+196x^2,$$

$$196-196x^2+2744x-9604-9801+2772x-196x^2=0,$$

$$-392x^2+5516x-19209=0,$$

$$f(x)=\boxed{392x^2-5516x+19209=0,}$$

$$D=\boxed{(5516)^2-4\cdot 392\cdot 19209}=$$

$$=30426256-30119712=306544>0,$$

$$x_1=\frac{5516-\sqrt{306544}}{2\cdot 392}=\boxed{\frac{1379-\sqrt{19159}}{196}},$$

$$x_2=\frac{5516+\sqrt{306544}}{2\cdot 392}=\boxed{\frac{1379+\sqrt{19159}}{196}}$$

$$f(x)=392x^2-5516x+19209;$$

$$f'(x)=2\cdot 392x-5516,$$

$$\boxed{f'(x)=784x-5516,}$$

Таким образом, я теоретически нашёл корни, к которым должен сойтись мой метод Ньютона для посчитанной относительно с = 7 функции и её производной.

Кстати, что касается начальных приближений, то Метод Ньютона довольно гибок в этом плане. Я пробовал брать то близкие к корням числа, то далёкие, но все равно метод сходился достаточно быстро.

Итак, для начала я считал 1-ый корень ($X_1 = \frac{1379 - \sqrt{19159}}{196}$), а в качестве приближения взял число $X_0 = 4$:

```
Параметр c = 7.0 (один из множества случаев, когда система имеет 2 корня)
Вид функции при c = 7.0 : f(x) = 392x^2 - 5516x + 19209, f'(x) = 784x - 5516
Корни этой функции, расчитанные теоретически: 6.329510002382672 , 7.7419185690459

Поиск первого корня методом Ньютона с начальным приближением X0 = 4.0
Расчёты итераций для Xk...
1 ) Xk = 5.435714285714286 , |X* - Xk| = 0.8937957166683859
2 ) Xk = 6.079862882653062 , |X* - Xk| = 0.24964711972960973
3 ) Xk = 6.296908865746957 , |X* - Xk| = 0.032601136635714845
4 ) Xk = 6.328790710075645 , |X* - Xk| = 0.0007192923070267199
5 ) Xk = 6.329509636443943 , |X* - Xk| = 3.659387290966265e-07
6 ) Xk = 6.329510002382577 , |X* - Xk| = 9.50350909079134e-14
7 ) Xk = 6.329510002382669 , |X* - Xk| = 2.6645352591003757e-15
Корень функции, полученный с помощью метода Ньютона с начальным приближением X0 = 4.0 , равен 6.329510002382577
```

Метод на данном корне сошёлся хорошо.

Но я также пробовал его и на более отдалённых приближениях (X_0 = -9999):

```
Поиск первого корня методом Ньютона с начальным приближением X0 = -9999.0
Расчёты итераций для Xk...
1 ) Xk = -4995.982167778326 , |X* - Xk| = 5002.311677780708
2 ) Xk = -2494.473276588671 , |X* - Xk| = 2500.8027865910535
3 ) Xk = -1243.7188808362068 , |X* - Xk| = 1250.0483908385895
4 ) Xk = -618.3417826446876 , |X* - Xk| = 624.6712926470702
5 ) Xk = -305.65343291824223 , |X* - Xk| = 311.9829429206249
6 ) Xk = -149.3096567927582 , |X* - Xk| = 155.63916679514088
7 ) Xk = -71.138566198375 , |X* - Xk| = 77.46807620075768
8 ) Xk = -32.05461578095495 , |X* - Xk| = 38.384125783337616
9 ) Xk = -12.515829876281376 , |X* - Xk| = 18.84533987866405
10 ) Xk = -2.752811889938773 , |X* - Xk| = 9.082321892321445
11 ) Xk = 2.1159762448229937 , |X* - Xk| = 4.213533757559678
12 ) Xk = 4.525159183443685 , |X* - Xk| = 1.8043508189389872
13 ) Xk = 5.681111193119077 , |X* - Xk| = 0.648398809263595
14 ) Xk = 6.174327640053576 , |X* - Xk| = 0.15518236232909555
15 ) Xk = 6.315531630591472 , |X* - Xk| = 0.013978371791199429
16 ) Xk = 6.329374345909594 , |X* - Xk| = 0.0001356564730778942
17 ) Xk = 6.329509989355881 , |X* - Xk| = 1.3026791201298238e-08
18 ) Xk = 6.329510002382672 , |X* - Xk| = 0.0
19 ) Xk = 6.329510002382672 , |X* - Xk| = 0.0
Корень функции, полученный с помощью метода Ньютона с начальным приближением X0 = -9999.0 , равен 6.329510002382672
```

Затем я также считал и 2-ой корень ($X_2 = \frac{1379 + \sqrt{19159}}{196}$), а в качестве приближения взял число $X_0 = 10$:

```
Поиск второго корня методом Ньютона с начальным приближением X0 = 10.0
Расчёты итераций для Xk...
1 ) Xk = 8.60197934595525 , |X* - Xk| = 0.86006077690935
2 ) Xk = 7.978055018346543 , |X* - Xk| = 0.23613644930064304
3 ) Xk = 7.771504694720718 , |X* - Xk| = 0.029586125674817332
4 ) Xk = 7.742513397968464 , |X* - Xk| = 0.0005948289225639058
5 ) Xk = 7.741918819344358 , |X* - Xk| = 2.502984575158962e-07
6 ) Xk = 7.741918569045946 , |X* - Xk| = 4.618527782440651e-14
7 ) Xk = 7.741918569045894 , |X* - Xk| = 6.217248937900877e-15
Корень функции, полученный с помощью метода Ньютона с начальным приближением X0 = 10.0 , равен 7.741918569045946
```

Для этого корня я также попробовал приближения, которые подальше (X_0 = 9999):

```
Поиск второго корня методом Ньютона с начальным приближением X0 = 9999.0
Расчёты итераций для Xk...
1 ) Xk = 5003.017882099136 , |X* - Xk| = 4995.27596353009
2 ) Xk = 2505.0268481049825 , |X* - Xk| = 2497.2849295359365
3 ) Xk = 1256.0313810204605 , |X* - Xk| = 1248.2894624514145
4 ) Xk = 631.5337473032955 , |X* - Xk| = 623.7918287342496
5 ) Xk = 319.2851300947936 , |X* - Xk| = 311.5432115257477
6 ) Xk = 163.16122078981016 , |X* - Xk| = 155.41930222076425
7 ) Xk = 85.10006472870481 , |X* - Xk| = 77.3581461596589
8 ) Xk = 46.07108382373793 , |X* - Xk| = 38.32916525469203
9 ) Xk = 26.5597871649249 , |X* - Xk| = 18.81786859594659
10 ) Xk = 16.810522765615673 , |X* - Xk| = 9.068604196569773
11 ) Xk = 11.948629229573983 , |X* - Xk| = 4.206710660528083
12 ) Xk = 9.54292823272002 , |X* - Xk| = 1.8010096636741206
13 ) Xk = 8.38877916355586 , |X* - Xk| = 0.6468605945099597
14 ) Xk = 7.896541098823352 , |X* - Xk| = 0.15462252977745194
15 ) Xk = 7.755805291874866 , |X* - Xk| = 0.013886722828965858
16 ) Xk = 7.742052469531584 , |X* - Xk| = 0.00013390048568417967
17 ) Xk = 7.741918581737648 , |X* - Xk| = 1.2691748096926858e-08
18 ) Xk = 7.7419185690458985 , |X* - Xk| = 1.7763568394002505e-15
19 ) Xk = 7.7419185690458985 , |X* - Xk| = 1.7763568394002505e-15
Корень функции, полученный с помощью метода Ньютона с начальным приближением X0 = 9999.0 , равен 7.7419185690458985
```

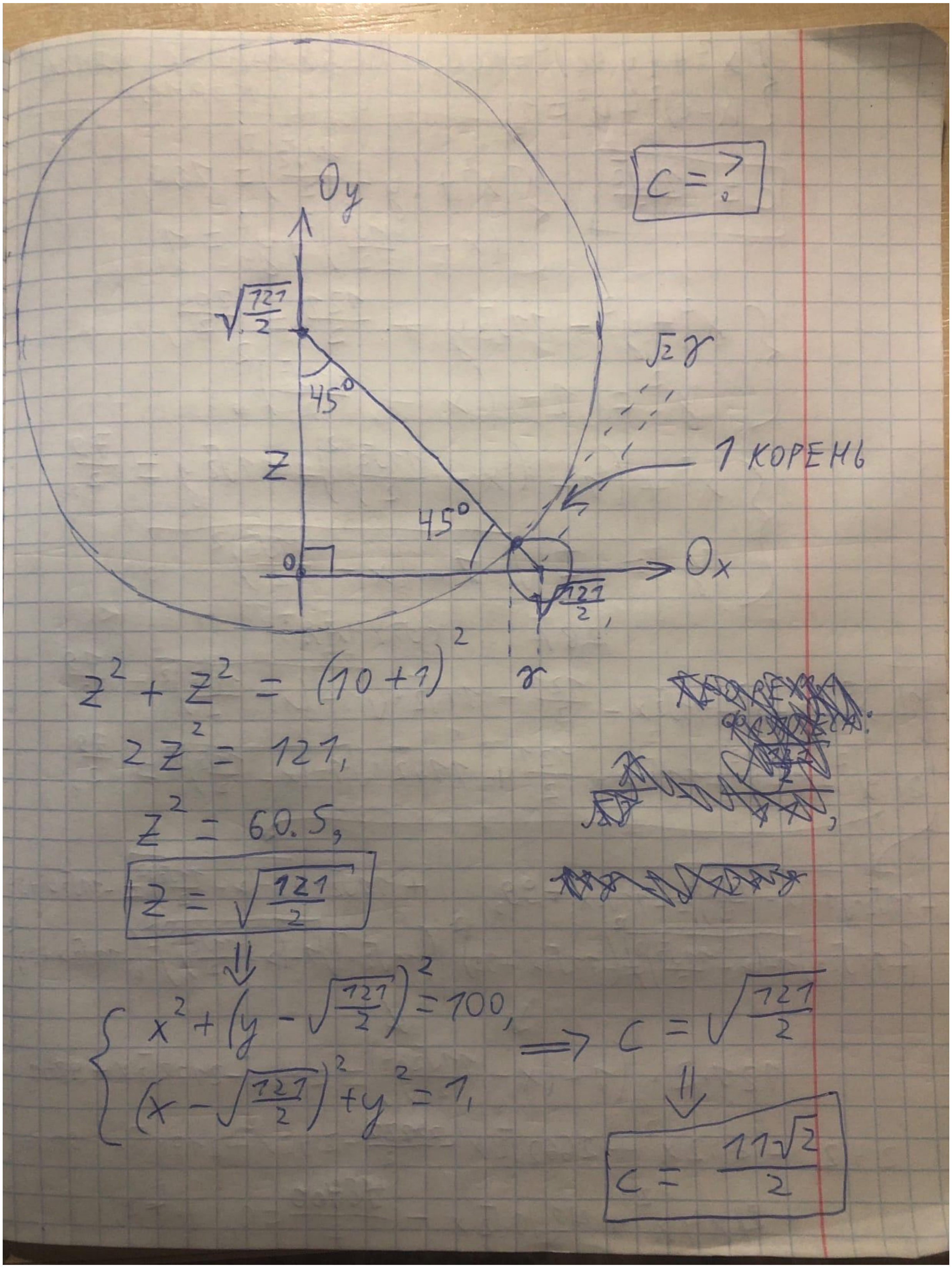
Таким образом, для случая с двумя корнями Метод Ньютона работает очень хорошо. Какое бы начальное приближение мы не взяли, оно сведётся к одному из двух корней (и насколько я понял, к тому из корней, который поближе).

Случай 1 – система имеет ровно 1 корень

Вообще, этот случай возможен лишь в 4-ёх точках (лишь 4-мя способами окружности могут касаться, я надеюсь, очевидно, какими, ~~пожалуйста, не заставляйте меня это доказывать~~).

Для своего случая я взял точку, в которой окружности касаются внешним образом справа снизу.

К сожалению, в GeoGebra нельзя узнать точное значение параметра c, при котором окружности касались бы, поэтому мне пришлось пойти здесь другим путём. Чтобы подсчитать параметр c, который позволяет окружностям коснуться в одной точке, я решил несложную геометрическую задачку в тетради:



Из теоремы Пифагора я нашёл оба равных катета из суммы радиусов моих окружностей, а значения этих катетов по совместительству и являются параметром с, потому что чётко видно, что большая окружность сместилась по оси O_y ровно на значение этого катета вверх, а малая окружность сместилась по оси O_x ровно на значение этого катета вправо. Так, я пришёл к выводу, что для этого случая параметр с = $\frac{11\sqrt{2}}{2}$.

Далее я, как и в первом случае, подставил параметр с в уравнение, выразил во втором уравнении у через х и вывел уравнение функции, зависящей только от х. Я не стал считать дискриминант (хотя он, по идее, получился бы у меня равным нулю), а просто выделил полный квадрат, откуда сразу и был виден корень:

$$\begin{cases} x^2+(y-c)^2=100, \\ (x-c)^2+y^2=1, \end{cases}$$

СЛУЧАЙ С ОДНИМ КОРНЕМ

$$\boxed{c=\frac{11\sqrt{2}}{2}}\qquad \begin{cases} x^2+\left(y-\frac{11\sqrt{2}}{2}\right)^2=100, \\ \left(x-\frac{11\sqrt{2}}{2}\right)^2+y^2=1, \end{cases}$$

$$\begin{cases} x^2+y^2-11\sqrt{2}y+\frac{121}{2}=\frac{200}{2}, \\ y^2=1-\left(x-\frac{11\sqrt{2}}{2}\right)^2, \end{cases}$$

$$\begin{cases} x^2+\left(1-\left(x-\frac{11\sqrt{2}}{2}\right)^2\right)-11\sqrt{2}y=\frac{79}{2}, \\ y=\pm\sqrt{1-\left(x-\frac{11\sqrt{2}}{2}\right)^2}, \end{cases}$$

ДВА СЛУЧАЯ:

$$\left[\begin{array}{l} 1)\ x^2+1-x^2+11\sqrt{2}x-\frac{121}{2}-11\sqrt{2}\cdot\sqrt{1-\left(x-\frac{11\sqrt{2}}{2}\right)^2}=\frac{79}{2}, \\ 2)\ x^2+1-x^2+11\sqrt{2}x-\frac{121}{2}+11\sqrt{2}\cdot\sqrt{1-\left(x-\frac{11\sqrt{2}}{2}\right)^2}=\frac{79}{2}, \end{array}\right.$$

$$-11\sqrt{2}\cdot\sqrt{1-\left(x-\frac{11\sqrt{2}}{2}\right)^2}=-11\sqrt{2}x+99, \qquad +11\sqrt{2}\cdot\sqrt{1-\left(x-\frac{11\sqrt{2}}{2}\right)^2}=-11\sqrt{2}x+99,$$

$$\left(-11\sqrt{2}\cdot\sqrt{1-\left(x-\frac{11\sqrt{2}}{2}\right)^2}\right)^2=\left(-11\sqrt{2}x+99\right)^2 \qquad \left(11\sqrt{2}\cdot\sqrt{1-\left(x-\frac{11\sqrt{2}}{2}\right)^2}\right)^2=\left(-11\sqrt{2}x+99\right)^2$$

$$121\cdot 2\cdot\left(1-\left(x-\frac{11\sqrt{2}}{2}\right)^2\right)=242x^2-2178\sqrt{2}x+9801,$$

$$242-242x^2+2662\sqrt{2}x-\frac{29282}{2}-242x^2+2178\sqrt{2}x-9801=0,$$

$$-484x^2+4840\sqrt{2}x-\frac{48400}{2}=0, \quad |\cdot(-2)$$

$$968x^2 - 9680\sqrt{2}x + 48400 = 0,$$

$$(22\sqrt{2}x)^2 - 2\cdot(22\sqrt{2}x)\cdot 220 + (220)^2 = 0,$$

$$(22\sqrt{2}x - 220)^2 = 0,$$

$$22\sqrt{2}x = 220,$$

$$\sqrt{2}x = 10,$$

$$\boxed{x = 5\sqrt{2}.}$$

$$\boxed{f(x) = x^2 - 10\sqrt{2}x + 50}$$

$$\boxed{f'(x) = 2x - 10\sqrt{2}}$$

Таким образом, я теоретически нашёл тот единственный корень, к которому должен сойтись мой метод Ньютона для посчитанной относительно $c = \frac{11\sqrt{2}}{2}$ функции и её производной.

Итак, я посчитал этот единственный корень ($X_1 = 5\sqrt{2}$), а в качестве приближения взял число $X_0 = 5$:

```
Поиск первого и единственного корня методом Ньютона с начальным приближением X0 = 5.0
Расчёты итераций для Xk...
1 ) Xk = 6.035533905932737 , |X* - Xk| = 1.0355339059327386
2 ) Xk = 6.553300858899108 , |X* - Xk| = 0.5177669529663671
3 ) Xk = 6.812184335382294 , |X* - Xk| = 0.2588834764831818
4 ) Xk = 6.941626073623883 , |X* - Xk| = 0.12944173824159222
5 ) Xk = 7.00634694274466 4 , |X* - Xk| = 0.06472086912081121
6 ) Xk = 7.038707377305075 , |X* - Xk| = 0.03236043456040072
7 ) Xk = 7.0548875945851455 , |X* - Xk| = 0.016180217280330034
8 ) Xk = 7.062977703225006 , |X* - Xk| = 0.008090108640469218
9 ) Xk = 7.067022757545224 , |X* - Xk| = 0.004045054320251928 5
10 ) Xk = 7.069045284704006 , |X* - Xk| = 0.002022527161469334
11 ) Xk = 7.070056548282286 , |X* - Xk| = 0.001011263583189148
12 ) Xk = 7.070562180072834 , |X* - Xk| = 0.0005056317926417364
13 ) Xk = 7.070814995967584 , |X* - Xk| = 0.00025281589789116765
14 ) Xk = 7.070941403900122 , |X* - Xk| = 0.00012640796535379195
15 ) Xk = 7.071004607893318 , |X* - Xk| = 6.320397215731077e-05
16 ) Xk = 7.0710362098811235 , |X* - Xk| = 3.160198435203654e-05
17 ) Xk = 7.071052010819679 , |X* - Xk| = 1.5801045796237645e-05
18 ) Xk = 7.0710599112621475 , |X* - Xk| = 7.900603328003797e-06
19 ) Xk = 7.071063861218329 , |X* - Xk| = 3.950647146311326e-06
20 ) Xk = 7.071065836922983 , |X* - Xk| = 1.9749424922110848e-06
21 ) Xk = 7.071066822717306 , |X* - Xk| = 9.891481695945004e-07
22 ) Xk = 7.0710673147788485 , |X* - Xk| = 4.970866269715657e-07
23 ) Xk = 7.071067550632209 , |X* - Xk| = 2.612332661300343e-07
24 ) Xk = 7.071067673030186 , |X* - Xk| = 1.3883528993119398e-07
25 ) Xk = 7.071067724209013 , |X* - Xk| = 8.76564625329479e-08
26 ) Xk = 7.071067764738982 , |X* - Xk| = 4.712649381843903e-08
27 ) Xk = 7.071067764738982 , |X* - Xk| = 4.712649381843903e-08
Корень функции, полученный с помощью метода Ньютона с начальным приближением X0 = 5.0 , равен 7.071067764738982
```

Метод на данном корне сошёлся хорошо.

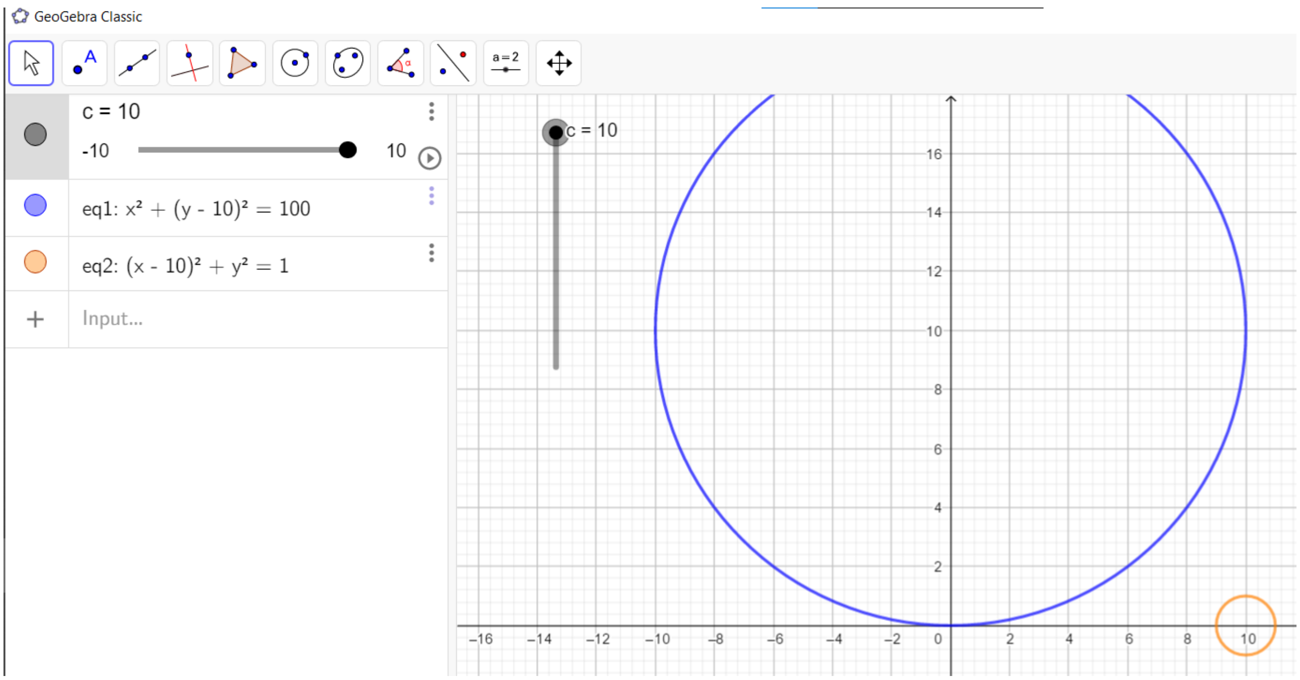
Но я также пробовал его и на более отдалённых приближениях (X_0 = 9999)

```
36 ) Xk = 7.071067959025961 , |X* - Xk| = 1.471604855751707e-07
37 ) Xk = 7.071067910742434 , |X* - Xk| = 9.887695817667463e-08
38 ) Xk = 7.071067910742434 , |X* - Xk| = 9.887695817667463e-08
Корень функции, полученный с помощью метода Ньютона с начальным приближением X0 = 9999.0 , равен 7.071067910742434
```

Так, для случая с одним единственным корнем Метод Ньютона также работает неплохо. Какое бы начальное приближение мы не взяли, оно сведётся к одному из двух корней (и насколько я понял, к тому из корней, который поближе). Только стоит отметить, что здесь слегка повыше погрешность вычислений (1e-8), что связано, скорее всего, с появлением корней внутри функции.

Случай 3 – система не имеет корней

Итак, последний случай, который я всё же решил разобрать – это случай, в котором система не имеет корней. Для этого я выбрал какой-нибудь параметр c = 10, при котором окружности не буту пересекаться (рисунок из GeoGebra):



Однако, прежде чем действовать в программе, я все же посчитал это функцию и её производную (для метода Ньютона), а также убедился в том, что дискриминант отрицательный:

$$\begin{cases} x^2+(y-c)^2=100, \\ (x-c)^2+y^2=1, \end{cases}$$

СЛУЧАЙ, КОГДА КОРНЕЙ НЕТ

$\boxed{c=10}$

$$\begin{cases} x^2+(y-10)^2=100, \\ (x-10)^2+y^2=1, \end{cases} \quad \begin{cases} x^2+y^2-20y=0, \\ y^2=1-(x-10)^2, \end{cases}$$

$$\begin{cases} x^2+(1-(x-10)^2)-20y=0, \\ y^2=1-(x-10)^2, \end{cases} \quad \begin{cases} x^2+(1-(x-10)^2)-20y=0, \\ y=\pm\sqrt{1-(x-10)^2}, \end{cases}$$

ДВА СЛУЧАЯ:

1) $x^2+1-x^2+20x-100-20\sqrt{1-(x-10)^2}=0$ | 2) $x^2+1-x^2+20x-100+20\sqrt{1-(x-10)^2}=$

$$20x-99=20\sqrt{1-(x-10)^2}, \qquad 20x-99=-20\sqrt{1-(x-10)^2},$$

$$(20x-99)^2=\left(20\sqrt{1-(x-10)^2}\right)^2 \qquad (20x-99)^2=\left(-20\sqrt{1-(x-10)^2}\right)^2,$$

$$400x^2-3960x+9801=400\cdot\left(1-(x-10)^2\right),$$

$$400x^2-3960x+9801=400-400x^2+8000x-40000,$$

$$400x^2+400x^2-3960x-8000x+9801+39600=0,$$

$$800x^2-11960x+49401=0,$$

$$D=(-11960)^2-4\cdot800\cdot49401=$$

$$=143041600-158083200=-15041600<0$$

НЕТ РЕШЕНИЙ.

$$\boxed{f(x)=800x^2-11960x+49401}$$

$$\boxed{f'(x)=1600x-11960}$$

Так, я теоретически доказал, что корней нет. Я попробовал поискать корни методом Ньютона в программе. Изначально, я дал ему 1000 итераций (начальное приближение выбрал наугад: $X_0 = 5$), чтобы он смог что-то найти, но ни одно значение для корня так и не сошлось:

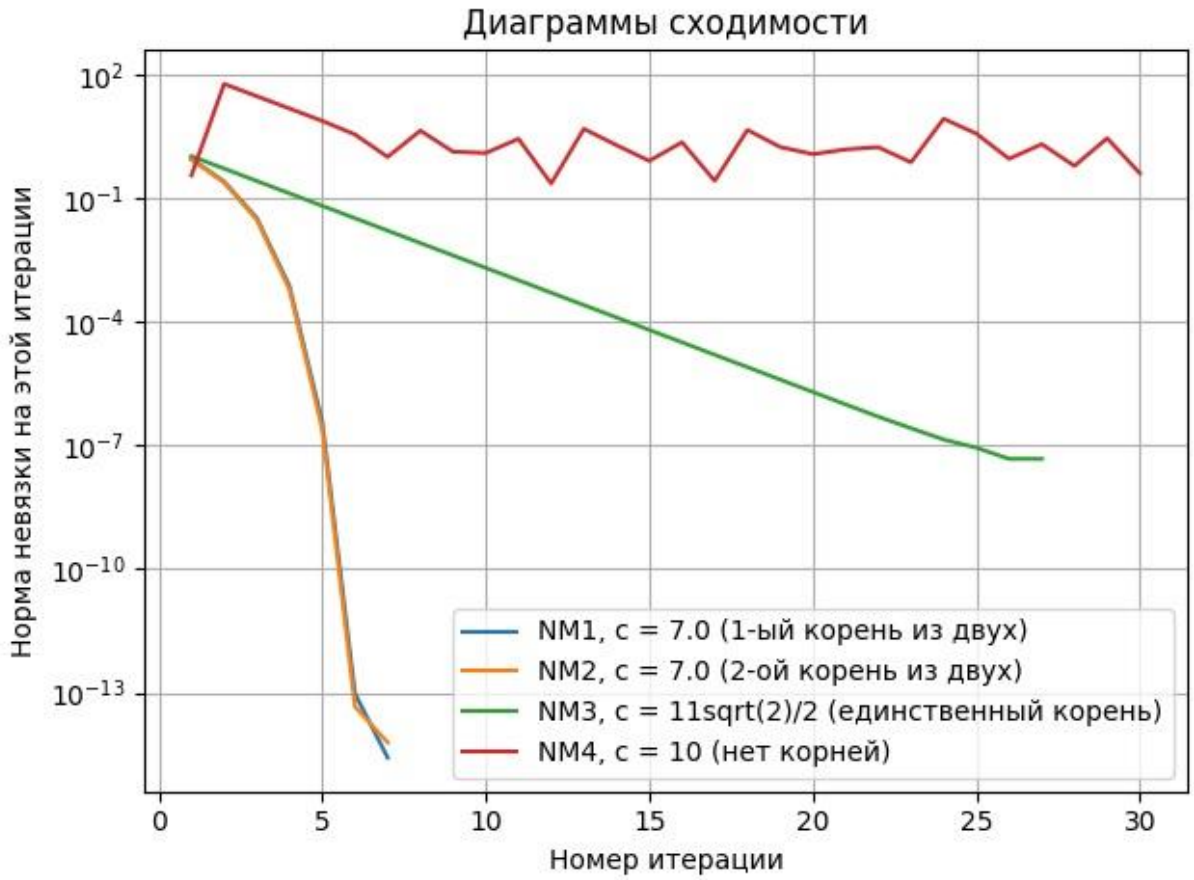
```
Параметр c = 10.0 (один из множества случаев, когда система вообще не имеет корней)
Вид функции при c = 10.0 : f(x) = 800x^2 - 11960x + 49401, f'(x) = 1600x - 11960

Попытки найти какие-то корни Методом Ньютона с начальным приближением X0 = 5.0
Расчёты итераций для Xk...
1 ) Xk = 7.424494949494495 , |X* - Xk| = 0.353427137629474
2 ) Xk = 65.61843497474719 , |X* - Xk| = 58.54736716288171
3 ) Xk = 36.49619050180289 , |X* - Xk| = 29.425122689937417
```

```
996 ) Xk = 4.468916618519904 , |X* - Xk| = 2.6021511933455717
997 ) Xk = 6.949247401804524 , |X* - Xk| = 0.1218204100609519
998 ) Xk = 12.799946776020409 , |X* - Xk| = 5.728878964154934
999 ) Xk = 9.585765995697907 , |X* - Xk| = 2.514698183832431
1000 ) Xk = 7.138560073854652 , |X* - Xk| = 0.06749226198917668
За 1000 итераций метод Ньютона так и не смог найти никаких корней...

Process finished with exit code 0
```

Позже, я изменил значение 1000 на 30, чтобы графики было проще сравнить друг с другом.

Итак, подытожим все вышесказанное графиками (в письме они также будут прикреплены, под именем Task_2_3_Графики_Сходимостей.jpg)

Графики



Здесь NM – означает Метод Ньютона.

NM1 и NM2 показывают диаграммы сходимостей для двух корней из первого случая (когда существует только 2 корня). Видно, что графики сошлись очень быстро

На графике NM3 показана диаграмма сходимости для единственного корня из второго случая (когда существует только 1 корень). Видно, что график сошёлся, хотя не так быстро и не так точно, как предыдущие два, но опять же, вероятно, это из-за погрешностей, так как в функции второго случая были корни и большие коэффициенты.

На графике NM4 показана диаграмма сходимости для первых 30 итераций из третьего случая (корней не существует). Тут видно, что даже со временем невязка не уменьшается, а только периодически изменяется то вверх, то вниз.

В целом, это всё, что я хотел описать в своём отчёте. Я надеюсь, я достаточно описал все свои шаги, хотя отчёт и без того уже получился на 30 страниц. В целом можно сказать, что был получен полезный опыт относительно методов Бисекции, Хорд и Ньютона.