Завдання 1

$50x^7 + 717x^6 + 675x^5 - 887x^4 - 791x^3 + 165x^2 + 96x - 7 = 0$

відокремлення коренів

За основною теоремою алгебри, враховуючи кратність, рівняння має 7 коренів. За теоремою 3 маємо формулу:

$$\frac{1}{1 + \frac{B}{|a_0|}} \le |x_{*i}| \le 1 + \frac{A}{|a_n|},$$

де $A = \max\{|a_6|, \ldots, |a_0|\} = |a_4| = 887$

$B = \max\{|a_7|, \ldots, |a_1|\} = |a_4| = 887$

тому:

$$\frac{1}{1 + \frac{887}{7}} \le |x_{*i}| \le 1 + \frac{887}{50} \quad \Rightarrow \quad 0.007829 \le |x_{*i}| \le 18.74$$

додатні корені:

$0.007829 \le x^{+}_{*i} \le 18.74$

від'ємні корені:

$-18.74 \le x^{-}_{*i} \le -0.007829$

Розіб'ємо проміжки існування коренів на ще менші, і знайдемо ті, на яких функція змінює знак

```
def func(x):
    return (50*x**7 + 717*x**6 + 675*x**5 - 887*x**4 - 791*x**3 + 165*x**2 + 96*x -7 )

def main():

    t =-18.74

    #Від'ємні корені
    while t < -0.007829:

        print(f"f({round(t,6)}) має знак ", end = '')

        if(func(t) > 0):
            print("+")

        else:
            print("-")

        t = t + 0.1

    print("\n\n--------------------------------------\n\n")

    t = 0.007829
    #додатні корені
    while t < 18.74:

        print(f"f({round(t,6)}) має знак ", end = '')

        if(func(t) > 0):
            print("+")

        else:
            print("-")

        t = t + 0.1



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

Отримали 7 проміжків, на яких функція змінює знак. Оскільки рівняння всього має 7 коренів, то на кожному з цих проміжків буде лише один корінь

$(-13.24; -13.14) \quad (-1.34; -1.24) \quad (-0.94; -0.84) \quad (-0.44; -0.34) \quad (0.007829; 0.107829) \quad (0.307829; 0.407829) \quad (0.907829; 1.007829)$

метод бісекцій

```
import math

bisection_1 = []

intervals = [(-13.24, -13.14), (-1.34, -1.24), (-0.94, -0.84), (-0.44, -0.34),
             (0.007829, 0.107829), (0.307829, 0.407829), (0.907829, 1.007829)]

eps = 0.00001

iterations = []

def print_(c,f):

    if c < 0:
        if func(c) < 0: print(f"{c} \t {func(c)}")
        else: print(f"{c} \t  {f(c)}")
    else:
        if func(c) < 0: print(f" {c} \t {func(c)}")
        else: print(f" {c} \t  {f(c)}")


print("       змінна                      функція")

iter = 0

for interval in intervals:

    a = interval[0]
    b = interval[1]
    c = (a+b)/2

    while b - a >= eps or abs(func(c)) >= eps:
        c = (a+b)/2

        if func(c) == 0:
            break

        if func(a)*func(c) < 0:
            b = c
        else:
            a = c

        iter = iter + 1

    bisection_1.append(c)
    print_(c,func)

iterations.append(iter)
print("\n\nкількість ітерацій: ", iter)
```

```
       змінна                      функція
-13.224691494160549       1.154497113020625e-06
-1.3061955833435057       4.819018954549392e-06
-0.8791226768493653       3.7610309391311603e-06
-0.36542667388916017      3.16874588435212e-06
 0.06777143621826172      4.6110278413280525e-06
 0.37500581884765627     -5.816584952356152e-06
 0.9926592304744722      -4.528676385007202e-07


кількість ітерацій:  159
```

метод хорд

спочатку відділимо проміжки ізоляції кореня, на яких друга похідна функції $f(x)$ зберігає знак.

$f(x) = 50x^7 + 717x^6 + 675x^5 - 887x^4 - 791x^3 + 165x^2 + 96x - 7$

$f'(x) = 350x^6 + 4302x^5 + 3375x^4 - 3548x^3 - 2373x^2 + 330x + 96$

$f''(x) = 2100x^5 + 21510x^4 + 13500x^3 - 10644x^2 - 4746x + 330$

```
def second_derr(x):
    return (2100*x**5 + 21510*x**4 + 13500*x**3 - 10644*x**2 - 4746*x + 330)

intervals = [(-13.24, -13.14), (-1.34, -1.24), (-0.94, -0.84), (-0.44, -0.34),
             (0.007829, 0.107829), (0.307829, 0.407829), (0.907829, 1.007829)]

for interval in intervals:

    t = interval[0]

    while t < interval[1]:

        print(f"f({round(t,6)}) має знак ", end ='')

        if func(t) < 0:
            print("- ", end = '\t')
        elif func(t) == 0:
            print("0 ", end = '\t')
        else:
            print("+ ", end = '\t')

        print(f"f\'\'({round(t,6)}) має знак ", end ='')

        if second_derr(t) < 0:
            print("-")
        elif second_derr(t) == 0:
            print("0")
        else:
            print("+")


        t = t + 0.01

    print("----------------------------")
```

Як бачимо проміжок (0.057829; 0.067829) потрібно дослідити більш детально

```
# (0.057829; 0.067829)

t = 0.057829

while t <= 0.067829:

    print(f"f({round(t,6)}) має знак ", end ='')

    if func(t) < 0:
        print("- ", end = '\t')
    elif func(t) == 0:
        print("0 ", end = '\t')
    else:
        print("+ ", end = '\t')

    print(f"f\'\'({round(t,6)}) має знак ", end ='')

    if second_derr(t) < 0:
        print("-")
    elif second_derr(t) == 0:
        print("0")
    else:
        print("+")

    t = t + 0.001
```

```
f(0.057829) має знак -  f''(0.057829) має знак +
f(0.058829) має знак -  f''(0.058829) має знак +
f(0.059829) має знак -  f''(0.059829) має знак +
f(0.060829) має знак -  f''(0.060829) має знак +
f(0.061829) має знак -  f''(0.061829) має знак -
f(0.062829) має знак -  f''(0.062829) має знак -
f(0.063829) має знак -  f''(0.063829) має знак -
f(0.064829) має знак -  f''(0.064829) має знак -
f(0.065829) має знак -  f''(0.065829) має знак -
f(0.066829) має знак -  f''(0.066829) має знак -
f(0.067829) має знак +  f''(0.067829) має знак -
```

отже отримали сім проміжків, на яких знаходяться корені, і на яких друга похідна не змінює знак:

$(-13.23; -13.22), \quad (-1.31; -1.3), \quad (-0.88; -0.87), \quad (-0.37; -0.36), \quad (0.066829; 0.067829), \quad (0.367829; 0.377829)$

$(0.987829, 0.997829)$

тепер застосуємо метод хорд для цих проміжків

```
new_intervals = [(-13.23, -13.22), (-1.31, -1.3),  (-0.88, -0.87),
                 (-0.37, -0.36),  (0.066829, 0.067829), (0.367829, 0.377829),
                 (0.987829,0.997829)]

secant_1 = []

eps = 0.00001

print("       змінна                      функція")

iter = 0

for interval in new_intervals:

    a = interval[0]
    b = interval[1]

    if second_derr(a)*func(a) > 0:
        N = a
        x_k1 = b
    else:
        N = b
        x_k1 = a

    x_k2 = x_k1 + 0.00002

    k = 0

    while abs(x_k2 - x_k1) >= eps or abs(func(x_k2)) >= eps:

        if k!= 0 :
            x_k1 = x_k2

        x_k2 = x_k1 - (func(x_k1)*(x_k1 - N))/(func(x_k1) - func(N))

        k = 1

        iter = iter + 1

    secant_1.append(x_k2)
    print_(x_k2, func)


iterations.append(iter)
print("\n\nкількість ітерацій: ", iter)
```

```
       змінна                      функція
-13.224691494160552       2.953697730845306e-07
-1.3061955787150221      -1.0656594895408489e-06
-0.8791226892997144       1.3290346601024794e-08
-0.3654266493688729      -1.5019909938018827e-07
 0.06777139289849798      1.3930279152418734e-10
 0.375005775789623        2.8573341630533378e-06
 0.9926592293393797      -3.1465596208590796e-06


кількість ітерацій:  24
```

метод Ньютона

спочатку виберемо для кожного проміжка початкове наближення так, щоб виконувалась умова "$f(x) \cdot f''(x) > 0$"

```
X = []
k = 0

intervals = [(-13.24, -13.14), (-1.34, -1.24), (-0.94, -0.84), (-0.44, -0.34),
             (0.007829, 0.107829), (0.307829, 0.407829), (0.907829, 1.007829)]


print("    проміжок          п. наближення")

for interval in intervals:

    if func(interval[0])*second_derr(interval[0]) > 0:
        X.append(interval[0])
    else:
        X.append(interval[1])

    print(f"({interval[0]};{interval[1]})   \t {X[k]}")

    k = k + 1
```

```
    проміжок          п. наближення
(-13.24;-13.14)           -13.24
(-1.34;-1.24)             -1.34
(-0.94;-0.84)             -0.94
(-0.44;-0.34)             -0.34
(0.007829;0.107829)       0.107829
(0.307829;0.407829)       0.407829
(0.907829;1.007829)       1.007829
```

тепер для кожного проміжка використаймо метод Ньютона

```
intervals = [(-13.24, -13.14), (-1.34, -1.24), (-0.94, -0.84), (-0.44, -0.34),
             (0.007829, 0.107829), (0.307829, 0.407829), (0.907829, 1.007829)]

newton_1 = []

iter = 0
k = 0
eps = 0.00001
step = 0

def first_derr(x):
    return (350*x**6 + 4302*x**5 + 3375*x**4 - 3548*x**3 - 2373*x**2 + 330*x + 96 )

for interval in intervals:

    x_k0 = X[k]
    x_k1 = x_k0 + 2*eps

    while abs(x_k1 - x_k0) >= eps or abs(func(x_k1)) >= eps:

        if step != 0:
            x_k0 = x_k1
        x_k1 = x_k0 - (func(x_k0)/first_derr(x_k0))

        step = 1
        iter = iter + 1

    step = 0
    k = k + 1
    print_(x_k1, func)
    newton_1.append(x_k1)

iterations.append(iter)
print(f"\nкількість ітерацій: {iter}")
```

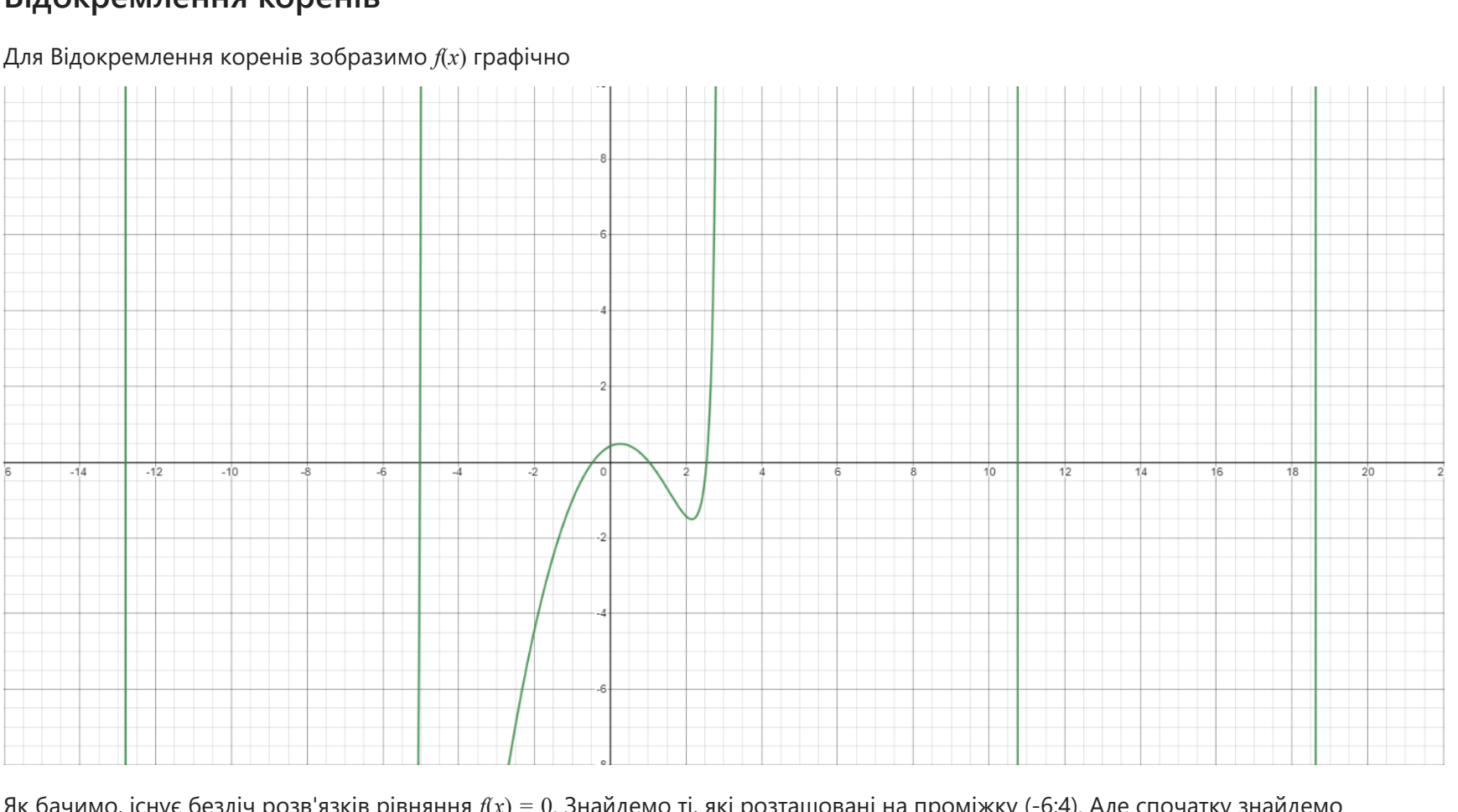
```
-13.224691494160552       2.953697730845306e-07
-1.306195579553216        2.5124791136477143e-11
-0.8791226893438664      -1.4210854715202004e-13
-0.3654266504817621       4.360813932180463e-10
 0.06777139289716755     -2.3065993559612252e-12
 0.375005789973663       -2.5224267119483557e-13
 0.9926592307376186       1.716060751277837e-07

кількість ітерацій: 24
```

Завдання 2

$tg(0.4x + 0.4) = x^2$

Відокремлення коренів

Для Відокремлення коренів зобразимо $f(x)$ графічно



Як бачимо, існує безліч розв'язків рівняння $f(x) = 0$. Знайдемо ті, які розташовані на проміжку (-6;4). Але спочатку знайдемо вертикальні асимптоти $f(x)$, які належать проміжку (-6;4):

$$0.4x + 0.4 = \frac{\pi}{2} + \pi n \quad \Rightarrow \quad x = \frac{\pi}{0.8} - 1 + \frac{\pi n}{0.4}$$

1. $n = -1 : x = \frac{\pi}{0.8} - 1 - \frac{\pi}{0.4} = -4.92699$

2. $n = 0 : x = \frac{\pi}{0.8} - 1 = 2.92699$

```
def second_func(x):
    return(math.tan(0.4*x+0.4) - x**2)

t = -6

while t < 4:

    print(f"f({round(t,6)}) має знак ", end = '')


    if second_func(t) < 0:
        print("-")
    elif second_func(t) > 0:
        print("+")
    else:
        print("0")

    t = t + 0.1
```

Отримуємо 6 проміжків, на яких функція змінює знак, хоча на цьому проміжку лише 4 корені. Це пояснюється тим, що на проміжках (-5; -4.9) та (2.9; 3) знаходяться вертикальні асимптоти $x = -4.92699$ та $x = 2.92699$ відповідно, а, отже, на цих проміжках не буде коренів. Тому отримуємо 4 проміжки, на кожному яких знаходится лише один корінь: (-5.1; -5), (-0.5; -0.4), (1; 1.1), (2.5; 2.6)

метод бісекцій

```
intervals = [(-5.1,-5), (-0.5, -0.4), (1, 1.1),  (2.5,  2.6)]
iter = 0
eps = 0.00001

bisection_2 = []

print("       змінна                      функція")

for interval in intervals:

    a = interval[0]
    b = interval[1]
    c = (a+b)/2


    while b - a >= eps or abs(second_func(c)) >= eps:

        if second_func(c) == 0:
            break

        if second_func(a)*second_func(c) < 0:
            b = c
        elif second_func(b)*second_func(c) < 0:
            a = c

        c = (a+b)*0.5

        iter = iter + 1

    print_(c,second_func)
    bisection_2.append(c)

iterations.append(iter)
print(f"\nкількість ітерацій: {iter}")
```

```
       змінна                      функція
-5.025910758972167        -7.000747679342112e-06
-0.4658172607421875       -2.867472523870873e-07
 1.0248565673828123        3.4839937905761786e-06
 2.543628692626953         7.153056181152806e-06

кількість ітерацій: 62
```

метод хорд

спочатку відділимо проміжки ізоляції кореня, на яких друга похідна функції $f(x)$ зберігає знак.

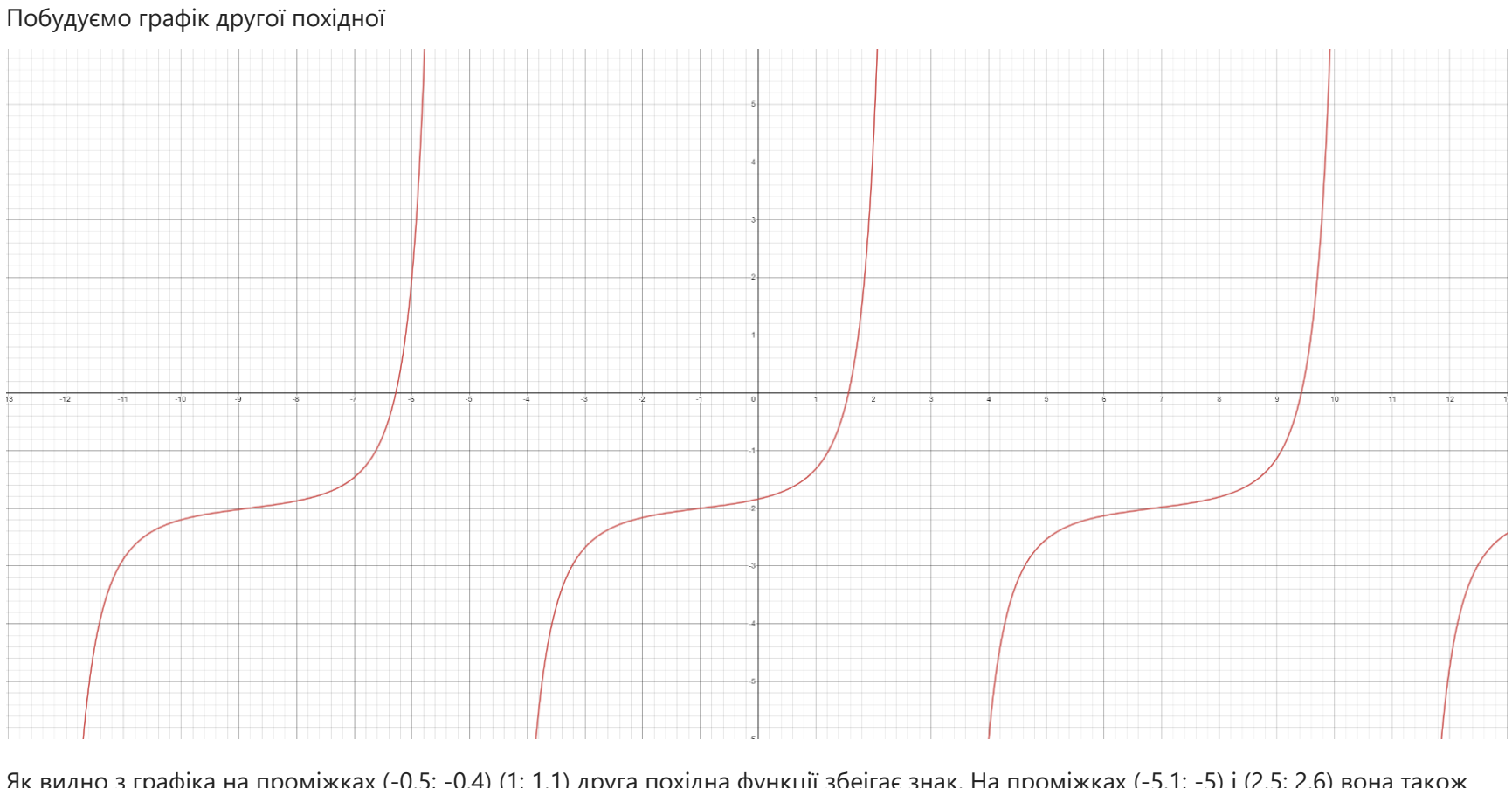
$f(x) = tg(0.4x + 0.4) - x^2$

$$f'(x) = \frac{0.4}{\cos^2(0.4x + 0.4)} - 2x$$

$$f''(x) = \frac{0 - 0.4(\cos^2(0.4x + 0.4))'}{\cos^4(0.4x + 0.4)} - 2 = \frac{-0.4 \cdot 2\cos(0.4x + 0.4) \cdot (-\sin(0.4x + 0.4)) \cdot 0.4}{\cos^4(0.4x + 0.4)} - 2 = \frac{0.32\sin(0.4x + 0.4)}{\cos^3(0.4x + 0.4)} - 2$$

примітка: $f''(x)$ матиме такі ж вертикальні асимптоти як і $f(x)$

Побудуємо графік другої похідної



Як видно з графіка на проміжках (-0.5; -0.4) (1; 1.1) друга похідна функції збеігає знак. На проміжках (-5.1; -5) і (2.5; 2.6) вона також буде зберігати знак, бо кінці цих відрізків знаходяться по одну сторону від вертикальних асимптот $x = -4.92699$ і $x = 2.92699$ відповідно

```
intervals = [(-5.1,-5), (-0.5, -0.4), (1, 1.1),  (2.5,  2.6)]

secant_2 = []

iter = 0
step = 0

def second_derr_2(x):
    return (((0.32*math.sin(0.4*x+0.4))/((math.cos(0.4*x+0.4))**3)) -2)

for interval in intervals:
    a = interval[0]
    b = interval[1]

    if second_func(a)*second_derr_2(a) > 0:
        N = a
        x_k0 = b
    else:
        N = b
        x_k0 = a

    x_k1 = x_k0 + 2*eps

    while abs(x_k1 - x_k0) >= eps or abs(second_func(x_k1)) >= eps:

        if step != 0:
            x_k0 = x_k1

        x_k1 = x_k0 - ((second_func(x_k0))/(second_func(x_k0)-second_func(N)))*(x_k0 - N)

        step = 1
        iter = iter + 1

    print_(x_k1, second_func)
    secant_2.append(x_k1)

iterations.append(iter)
print(f"\nкількість ітерацій: {iter}")
```

```
-5.025910753407914        -5.5224689923250025e-06
-0.4658170261667159        3.003926707378923e-08
 1.0248593987608956        6.246033890100477e-08
 2.543627652606603        -5.3868201010942585e-06

кількість ітерацій: 26
```

метод Ньютона

спочатку виберемо для кожного проміжка початкове наближення так, щоб виконувалась умова "$f(x) \cdot f''(x) > 0$"

```
X = []
k = 0

intervals = [(-5.1,-5), (-0.5, -0.4), (1, 1.1),  (2.5,  2.6)]


print(" проміжок           п. наближення")

for interval in intervals:

    if second_func(interval[0])*second_derr_2(interval[0]) > 0:
        X.append(interval[0])
    else:
        X.append(interval[1])

    print(f"({interval[0]};{interval[1]})   \t\t {X[k]}")

    k = k + 1
```

```
 проміжок           п. наближення
(-5.1;-5)                  -5
(-0.5;-0.4)                -0.5
(1;1.1)                    1.1
(2.5;2.6)                  2.6
```

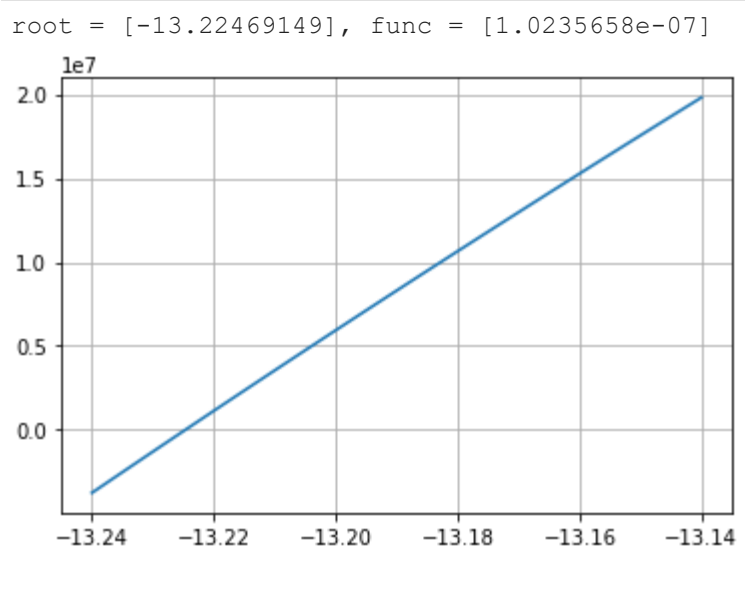
```
iter = 0
step = 0
k = 0
newton_2 = []

def first_derr(x):
    return (((0.4)/((math.cos(0.4*x+0.4))**2)) - 2*x)

for interval in intervals:

    x_k0 = X[k]
    x_k1 = x_k0 + 2*eps

    while abs(x_k1 - x_k0) >= eps or abs(second_func(x_k1)) >= eps:

        if step != 0:
            x_k0 = x_k1

        x_k1 = x_k0 - (second_func(x_k0))/(first_derr(x_k0))

        step = 1
        iter = iter + 1

    k = k + 1
    step = 0
    print_(x_k1, second_func)
    newton_2.append(x_k1)

iterations.append(iter)
print(f"\nкількість ітерацій: {iter}")
```

```
-5.025910732589643         8.411472407487963e-09
-0.46581704841047333      -1.9695356456850277e-13
 1.024859450455537        -9.317879801074014e-12
 2.5436280993755207        1.198863230911229e-11

кількість ітерацій: 14
```

Використання бібліотечних функцій. Перше завдання

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

t=np.linspace(-13.24,-13.14,100)
plt.plot(t,func(t))
plt.grid()

x=fsolve(func,-13.1)
print(f"root = {x}, func = {func(x)}")
```

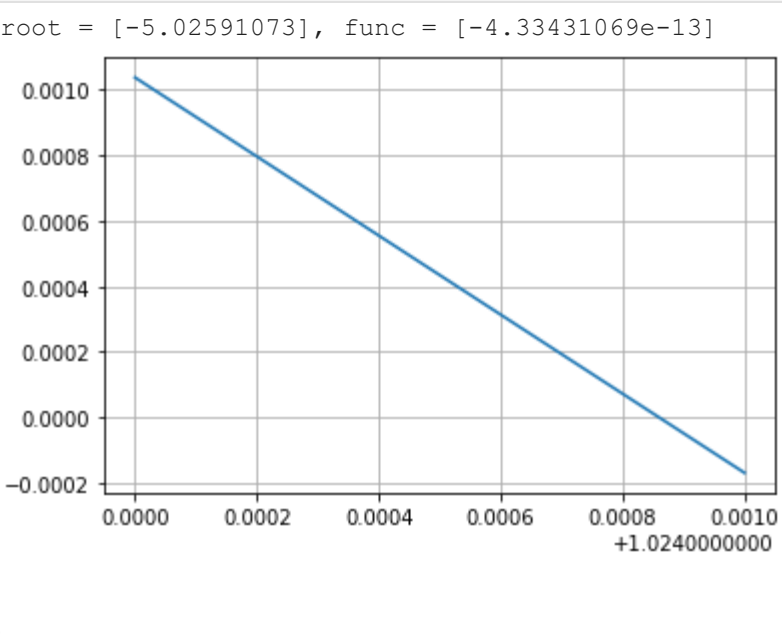
```
root = [-13.22469149], func = [1.0235658e-07]
```



Друге завдання

```
def second__func(x):
    return(np.tan(0.4*x+0.4) - x**2)

t=np.linspace(1.024,1.025,100)
plt.plot(t,second__func(t))
plt.grid()

x=fsolve(second_func,-13.1)
print(f"root = {x}, func = {second_func(x)}")
```

```
root = [-5.02591073], func = [-4.33431069e-13]
```



Висновок

```
print("завдання 1:\n")

print("корені:\n")

print("   м. бісекцій                  м. хорд                м. Ньютона")
for i in range(7):
    print(f"{bisection_1[i]} \t {secant_1[i]} \t {newton_1[i]}")

print("\nзначення функцій:\n")


print("   м. бісекцій                   м. хорд                         м. Ньютона")
for i in range(7):
    if i == 0:
        print(f"{func(bisection_1[i])} \t {func(secant_1[i])} \t         {func(newton_1[i])}")
    else:
        print(f"{func(bisection_1[i])} \t {func(secant_1[i])} \t {func(newton_1[i])}")


print("\nкількість ітерацій:\n")

print(f"метод бісекцій: {iterations[0]}")
print(f"метод хорд: {iterations[1]}")
print(f"метод Ньютона: {iterations[2]}")

print("\n-------------------------------------------------------------------------------------------\n")




print("завдання 2:\n")

print("корені:\n")

print("   м. бісекцій                  м. хорд                м. Ньютона")
for i in range(4):
    print(f"{bisection_2[i]} \t {secant_2[i]} \t {newton_2[i]}")

print("\nзначення функцій:\n")

print("   м. бісекцій                   м. хорд                         м. Ньютона")
for i in range(4):
    if i == 0 or i == 3:
        print(f"{second_func(bisection_2[i])} \t {second_func(secant_2[i])}      \t {second_func(newton_2[i])}")
    else:
        print(f"{second_func(bisection_2[i])} \t {second_func(secant_2[i])} \t\t {second_func(newton_2[i])}")


print("\nкількість ітерацій:\n")

print(f"метод бісекцій: {iterations[3]}")
print(f"метод хорд: {iterations[4]}")
print(f"метод Ньютона: {iterations[5]}")
```

```
завдання 1:

корені:

   м. бісекцій                  м. хорд                м. Ньютона
-13.224691494160549      -13.224691494160552     -13.224691494160552
-1.3061955833435057      -1.3061955787150221     -1.306195579553216
-0.8791226768493653      -0.8791226892997144     -0.8791226893438664
-0.36542667388916017     -0.3654266493688729     -0.3654266504817621
0.06777143621826172      0.06777139289849798     0.06777139289716755
0.37500581884765627      0.375005775789623       0.375005789973663
0.9926592304744722       0.9926592293393797      0.9926592307376186

значення функцій:

   м. бісекцій                   м. хорд                         м. Ньютона
1.154497113020625e-06    2.953697730845306e-07           2.953697730845306e-07
4.819018954549392e-06    -1.0656594895408489e-06         2.5124791136477143e-11
3.7610309391311603e-06   1.3290346601024794e-08          -1.4210854715202004e-13
3.16874588435212e-06     -1.5019909938018827e-07         4.360813932180463e-10
4.6110278413280525e-06   1.3930279152418734e-10          -2.3065993559612252e-12
-5.816584952356152e-06   2.8573341630533378e-06          -2.5224267119483557e-13
-4.528676385007202e-07   -3.1465596208590796e-06         1.716060751277837e-07

кількість ітерацій:

метод бісекцій: 159
метод хорд: 24
метод Ньютона: 24

-------------------------------------------------------------------------------------------

завдання 2:

корені:

   м. бісекцій                  м. хорд                м. Ньютона
-5.025910758972167       -5.025910753407914      -5.025910732589643
-0.4658172607421875      -0.4658170261667159     -0.46581704841047333
1.0248565673828123       1.0248593987608956      1.024859450455537
2.543628692626953        2.543627652606603       2.5436280993755207

значення функцій:

   м. бісекцій                   м. хорд                         м. Ньютона
-7.000747679342112e-06   -5.5224689923250025e-06         8.411472407487963e-09
-2.867472523870873e-07   3.003926707378923e-08           -1.9695356456850277e-13
3.4839937905761786e-06   6.246033890100477e-08           -9.317879801074014e-12
7.153056181152806e-06    -5.3868201010942585e-06         1.198863230911229e-11

кількість ітерацій:

метод бісекцій: 62
метод хорд: 26
метод Ньютона: 14
```

Під час виконання цієї лабораторної роботи були розглянуті три методи знаходження коренів алгебраїчного рівняння: метод половинного ділення (бісекції), метод хорд і метод Ньютона. Як видно вище метод бісекцій найповільніший, в той час, коли метод Ньютона найшвидший.