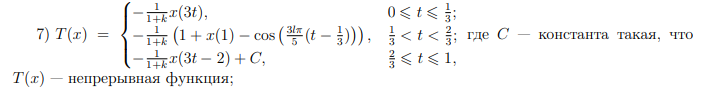
**Расчётно-графическая работа по функциональному анализу**

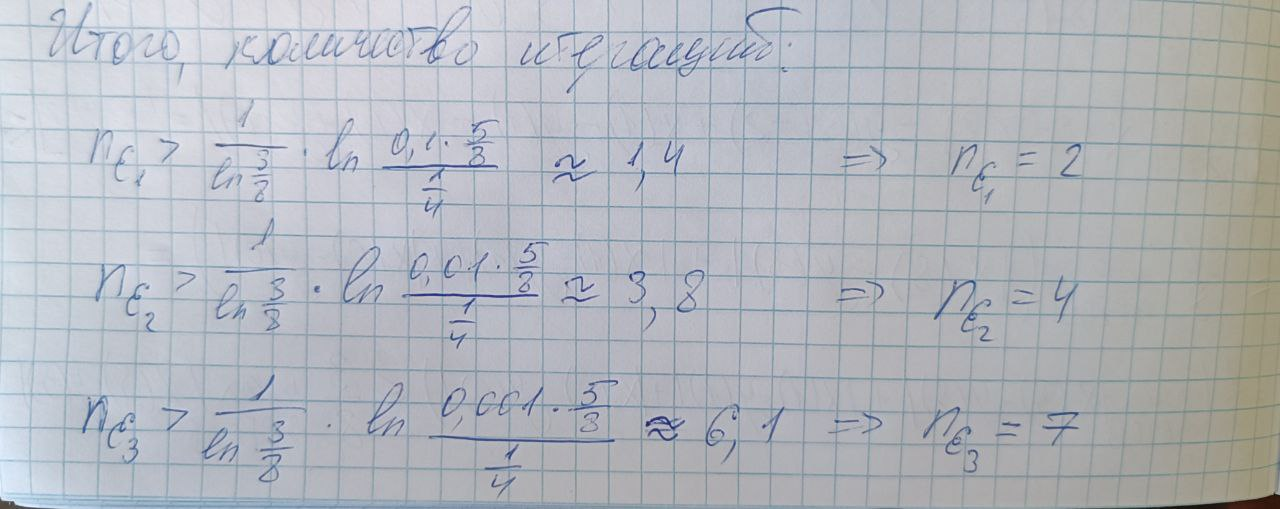
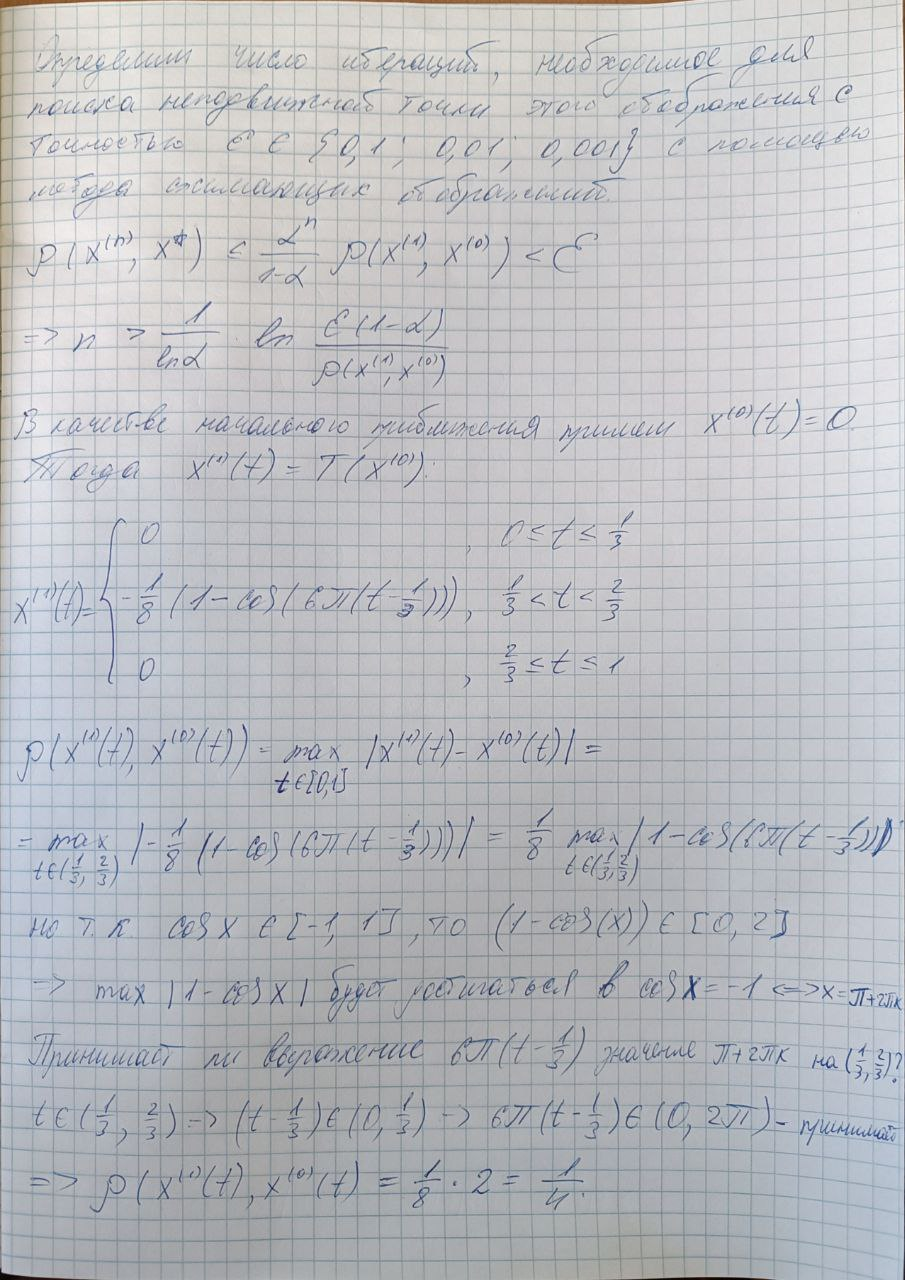
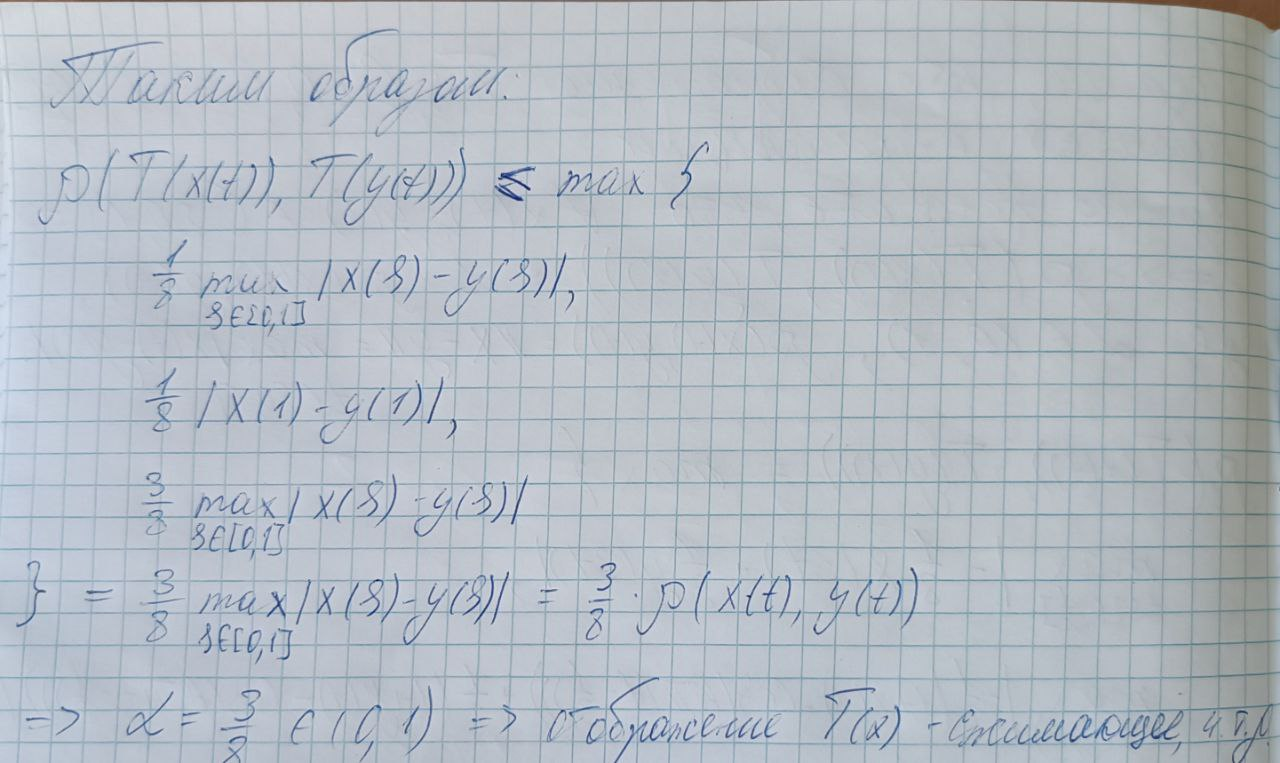
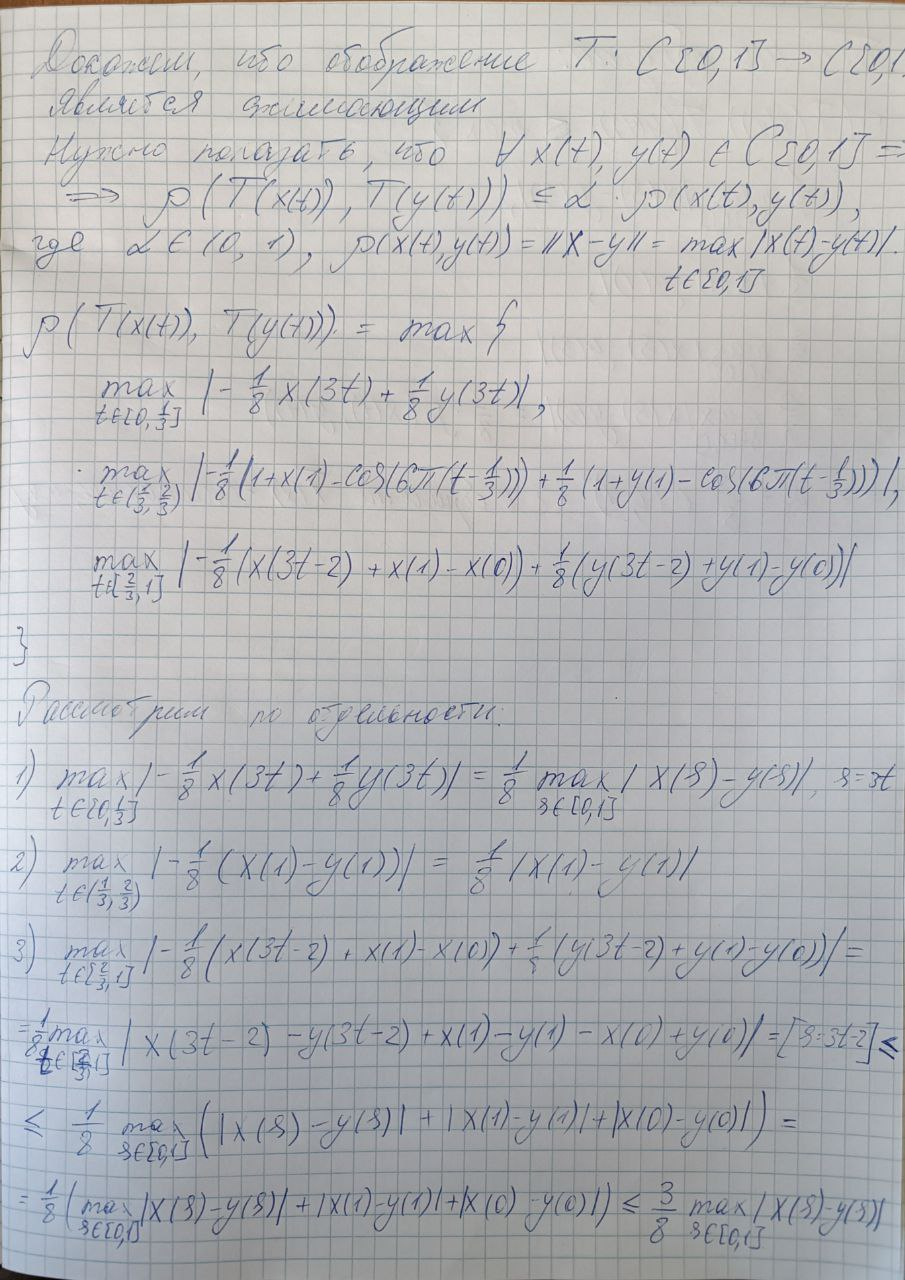
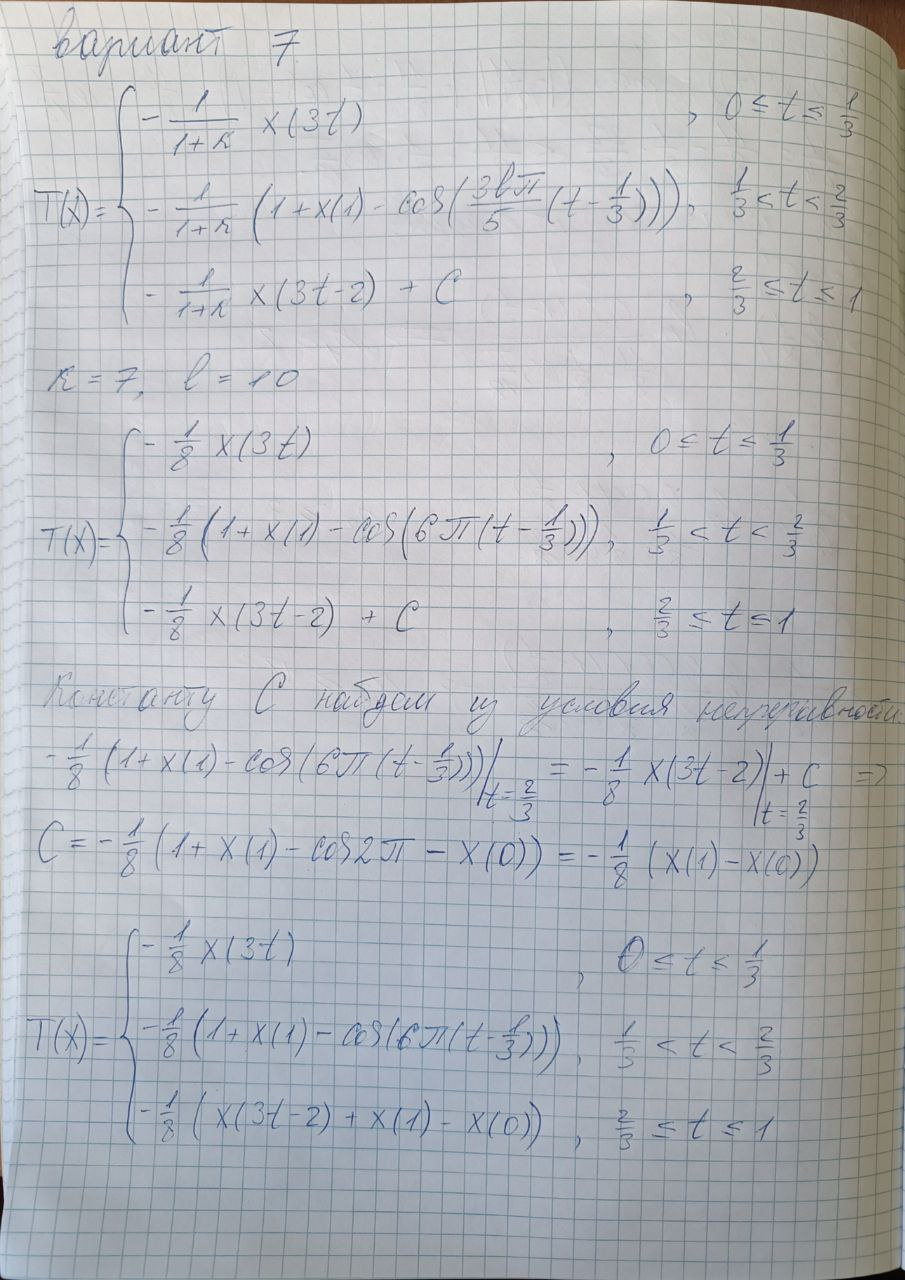
В приведённых ниже вариантах k ∈ {1, 3, 6, 7, 8, 9, 10, 12} — номер группы, l = 15, если номер студента в списке группы не больше 10, l = 10, если номер студента в списке группы от 11 до 20, l = 5, если номер студента не меньше 21. Вариант выбирается как остаток от деления номера по списку группы на 10.

**Задание I**

Докажите, что приведённое ниже отображение T : C[0; 1] → C[0; 1] является сжимающим. Определите число итераций, необходимое для поиска неподвижной точки этого отображения с точностью ε ∈ {10−1, 10−2, 10−3} с помощью метода сжимающих отображений. С помощью вычислительной техники постройте график функции, являющейся неподвижной точкой отображения T. Проверьте результаты при различных значениях ε и различных начальных приближениях в методе сжимающих отображений. Продемонстрируйте несколько графиков, получающихся при промежуточных вычислениях.

**Вариант 7**





**Краткое описание**

Зададим начальное приближение функции x0, массив точек ts, в которых будем вычислять функцию, а также определим словари prevX и curX, которые по порядковому номеру точки из массива ts хранят значение функции в этой точке. Словарь prevX хранит значение для функции, вычисленной на предыдущей итерации (изначально туда записываются значения функции x0), а словарь curX вычисляется на текущей итерации. В конце каждой итерации вычисляется текущая погрешность, и, если она не превышает заданный эпсилон, цикл завершается.

**Код**

from random import random

import numpy as np

import matplotlib.pyplot as plt

ALPHA = 0.375

COUNT\_POINTS = int(1e4 - 1)

EPS = 1e-1

def plot(x, y, iter):

    plt.plot(x, y, linestyle='-', color=(random(), random(), random()), label=f"{iter} итераций")

def x0(t):

    return 0

# def x0(t):

#     return t

# def x0(t):

#     return t \*\* 2 - t

ts = np.linspace(0, 1, COUNT\_POINTS)

prevX = dict()

for i in range(COUNT\_POINTS):

    prevX[i] = x0(ts[i])

iter = 1

while (True):

    curX = dict()

    isPlot = iter > 4

    # Строим функцию

    for i in range(COUNT\_POINTS):

        t = i/(COUNT\_POINTS-1)

        prevX\_0 = prevX[0]

        prevX\_1 = prevX[COUNT\_POINTS-1]

        if 0 <= i < COUNT\_POINTS/3:

            curX[i] = -1/8 \* prevX[3 \* i]

        elif COUNT\_POINTS/3 <= i < 2\*COUNT\_POINTS/3:

            curX[i] = -1/8 \* (1 + prevX\_1 - np.cos(6 \* np.pi \* (t - 1/3)))

        elif 2\*COUNT\_POINTS/3 <= i < COUNT\_POINTS:

            curX[i] = -1/8 \* (prevX[3 \* i - 2\*COUNT\_POINTS] + prevX\_1 - prevX\_0)

    if (isPlot):

        plot(ts, curX.values(), iter)

    # Расстояние

    rho = 0

    for i in range(COUNT\_POINTS):

        rho = max(rho, abs(curX[i] - x0(ts[i])))

    # Погрешность

    curEps = ALPHA \*\* iter / (1 - ALPHA) \* rho

    if (curEps < EPS):

        if (not isPlot):

            plot(ts, curX.values(), iter)

        break

    iter += 1

    prevX = curX

print(f"{iter} итераций")

plt.title(f"Неподвижная точка (точность {EPS})")

plt.legend()

plt.grid(True)

plt.show()

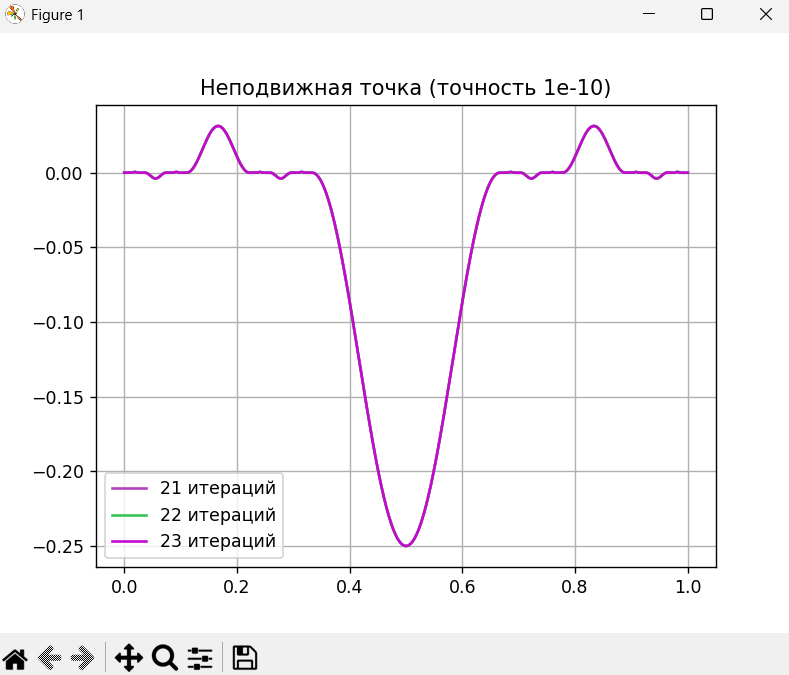
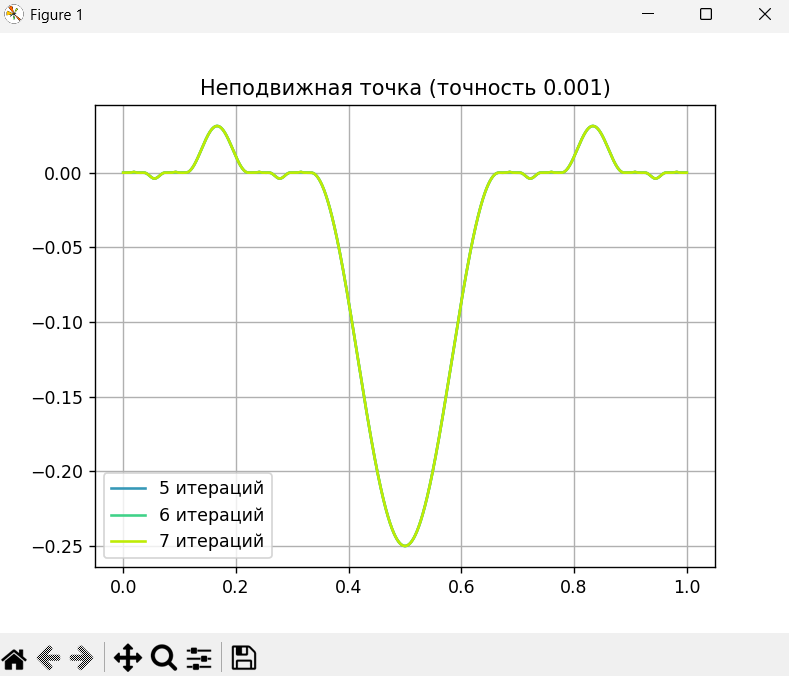
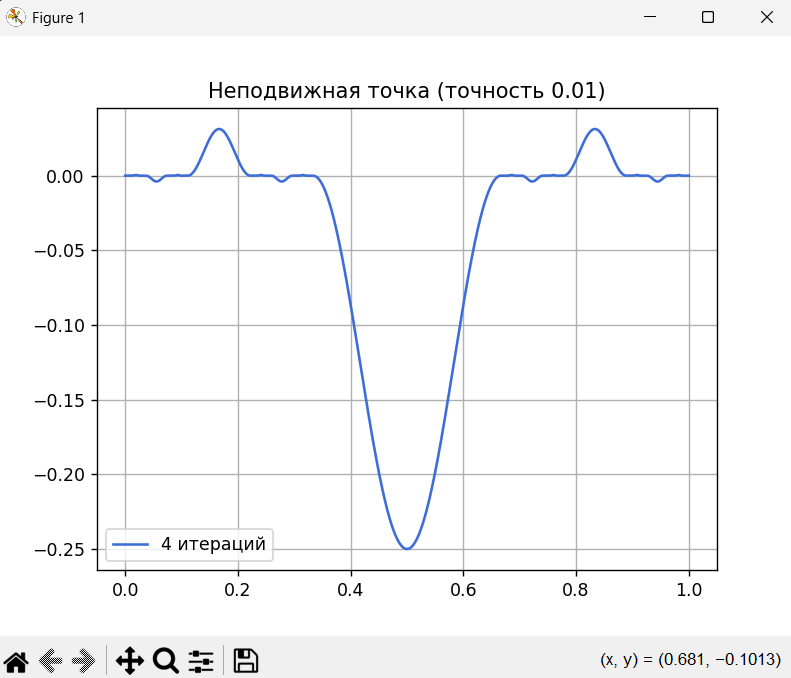
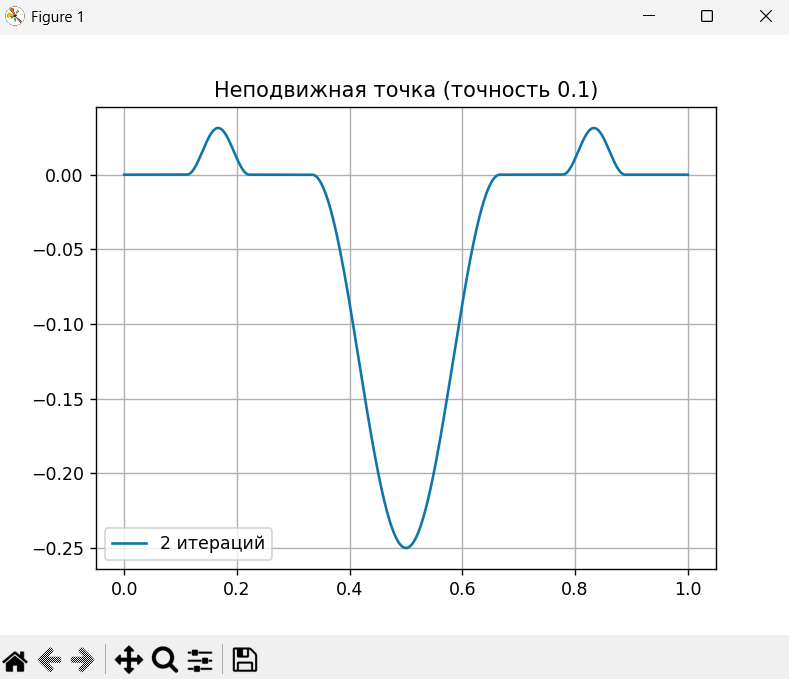
**Поиск неподвижной точки**

Далее программа тестировалась с различными начальными приближениями и различной точностью. На скриншотах видно, за сколько итераций удалось достичь нужной точности при разных начальных приближениях.

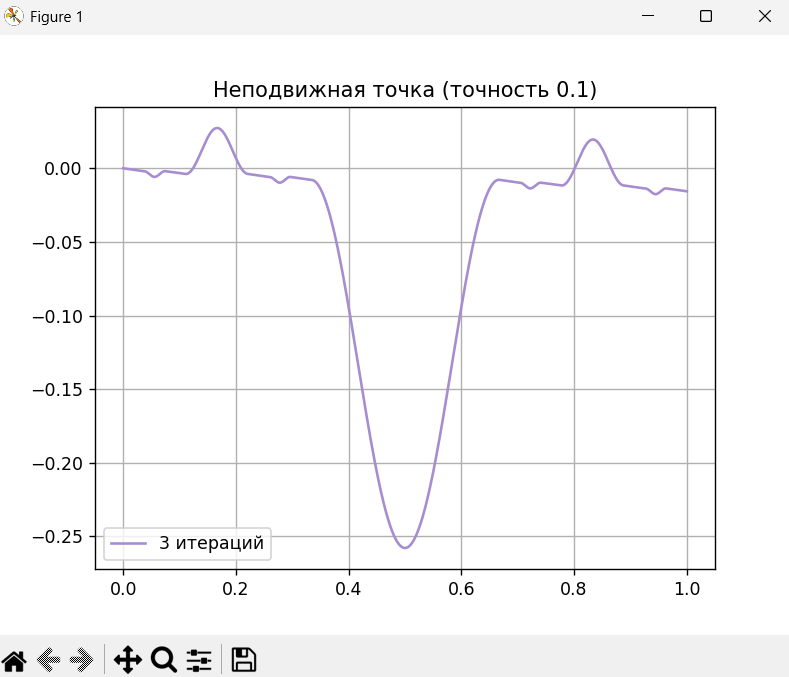
Заметим, что количество итераций, сделанных численно, совпало с теоретической оценкой.

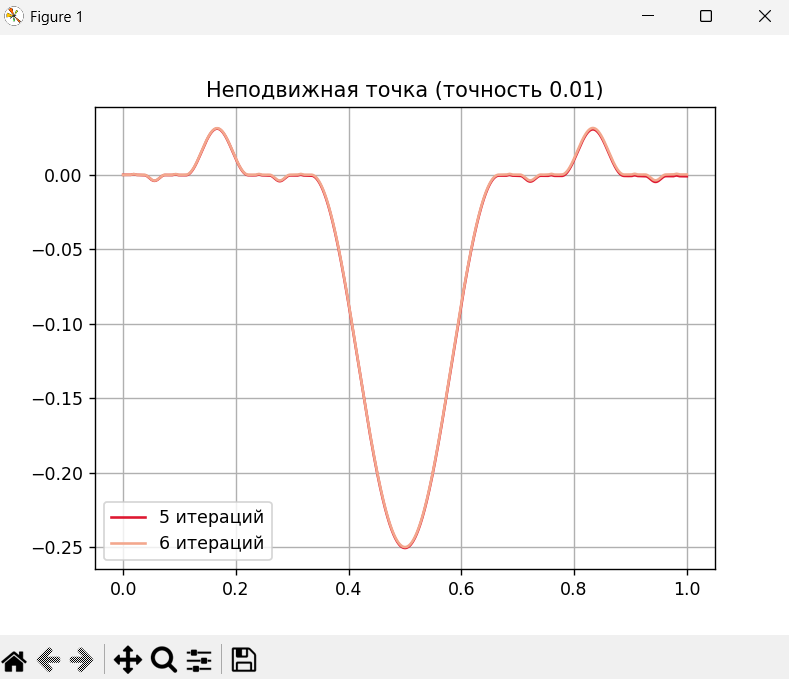
После первого приближения нулем стало видно, что неподвижная точка отображения походит на параболу t2 - t. Поэтому, задав начальное приближение этой параболой, получили меньшее число итераций для точности 0.001 по сравнению с начальным приближением нулем.

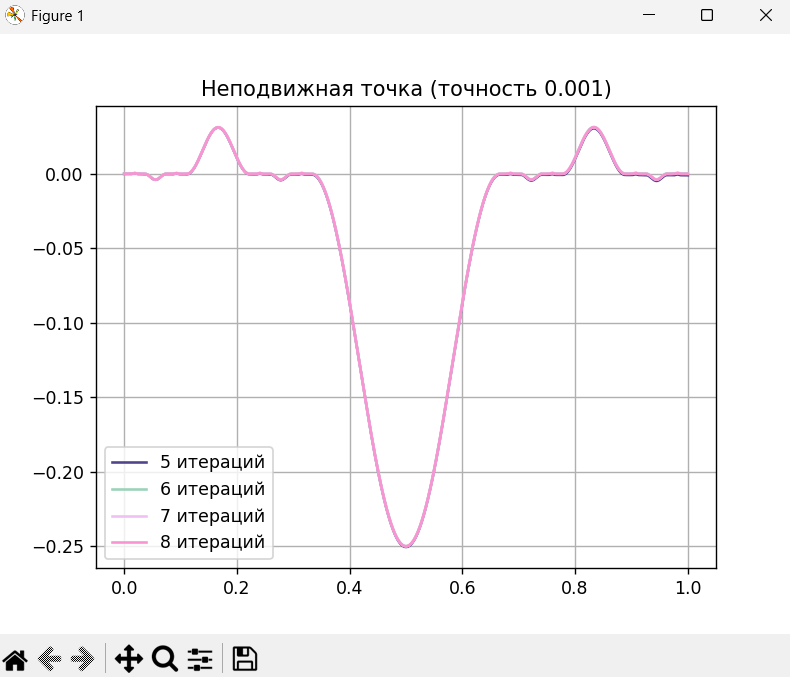
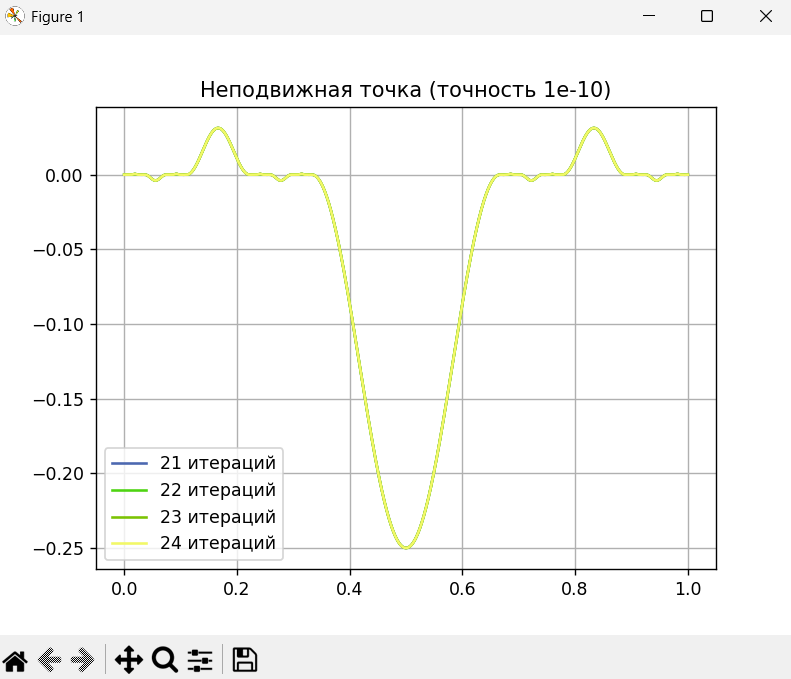
Начальное приближение x0(t) = 0:



Начальное приближение x0(t) = t:





Начальное приближение x0(t) = t2 - t:

