Министерство науки и высшего образования РФ

ФГАОУ ВО Пермский национальный исследовательский

политехнический университет

Кафедра «Вычислительная математика, механика и биомеханика»

Отчет по лабораторной работе № 3

Тема «Оптимизация функции нескольких переменных на ограниченном множестве»

по дисциплине «Теория принятия решений»

Выполнил: студент группы ИСТ-22-1б Петраков М.В.

Проверил: доцент каф. ВММБ Бояршинова И. Н.

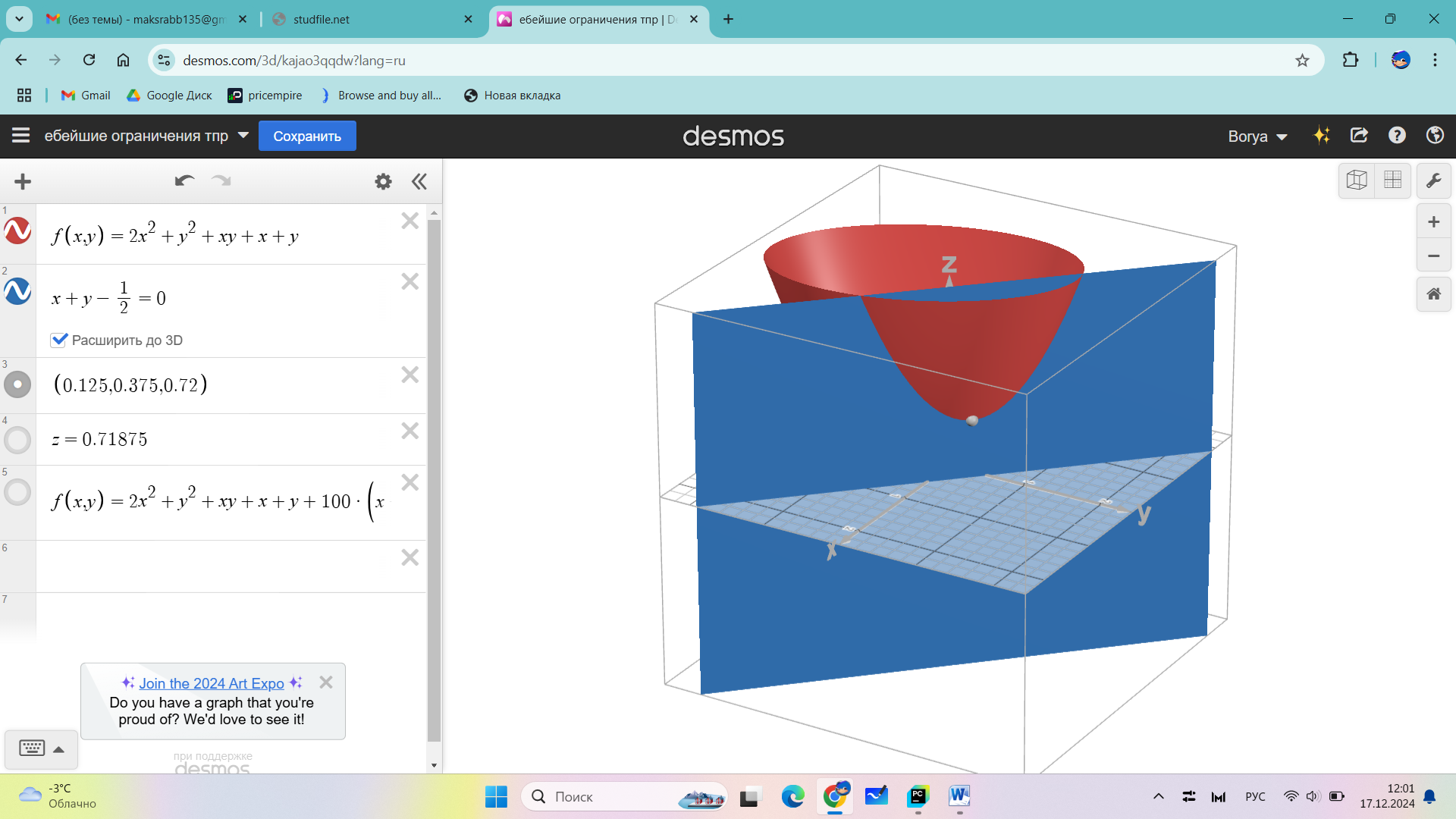
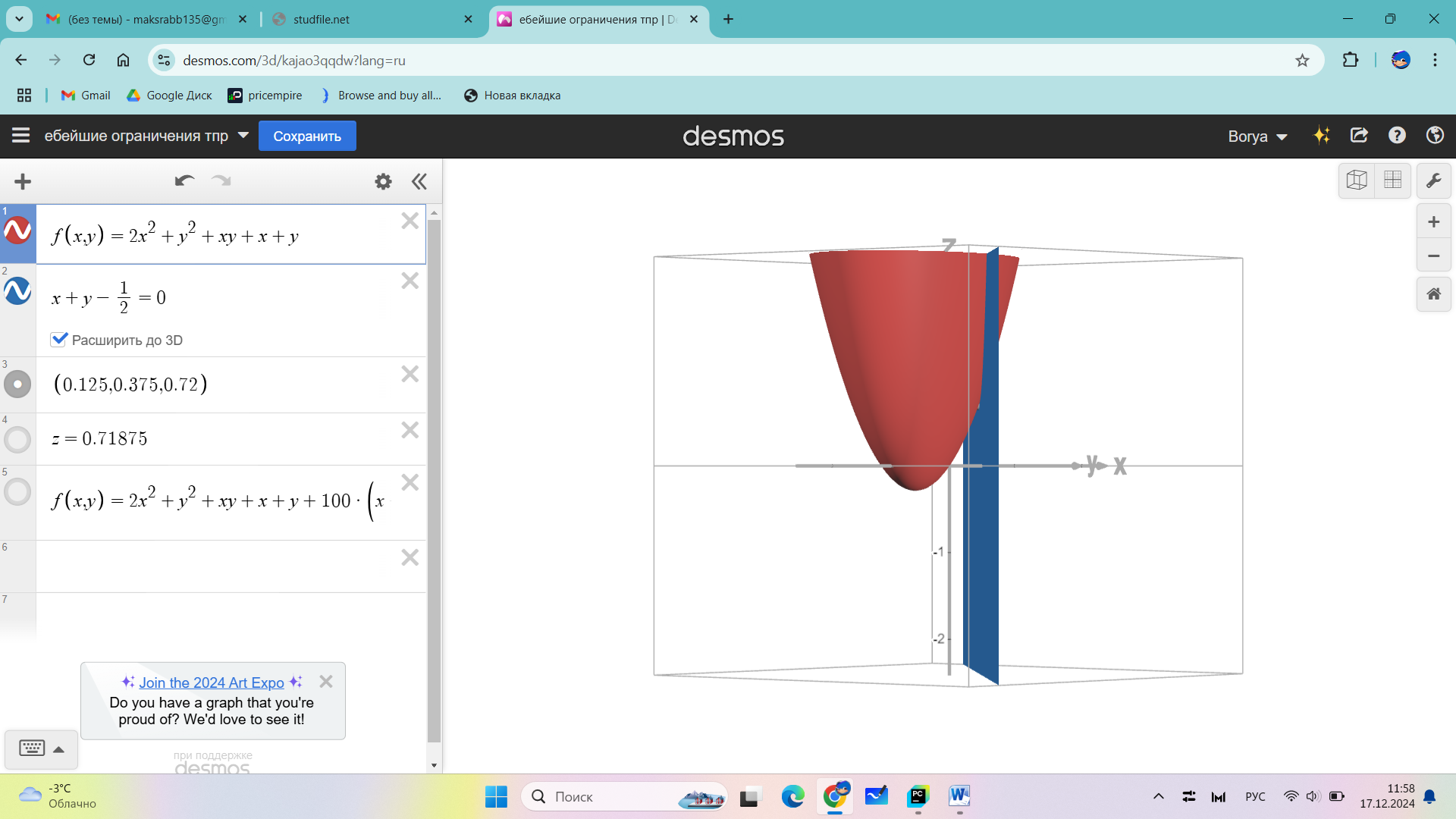
Пермь, 2024

Постановка задания

Найти минимум функции  на плоскости, уравнение которой задаётся самостоятельно.

1. Обоснование применимости методов

В качестве ограничения выбрано уравнение . Для выяснения возможности применить методы оптимизации необходимо рассмотреть пересечение функции с плоскостью.



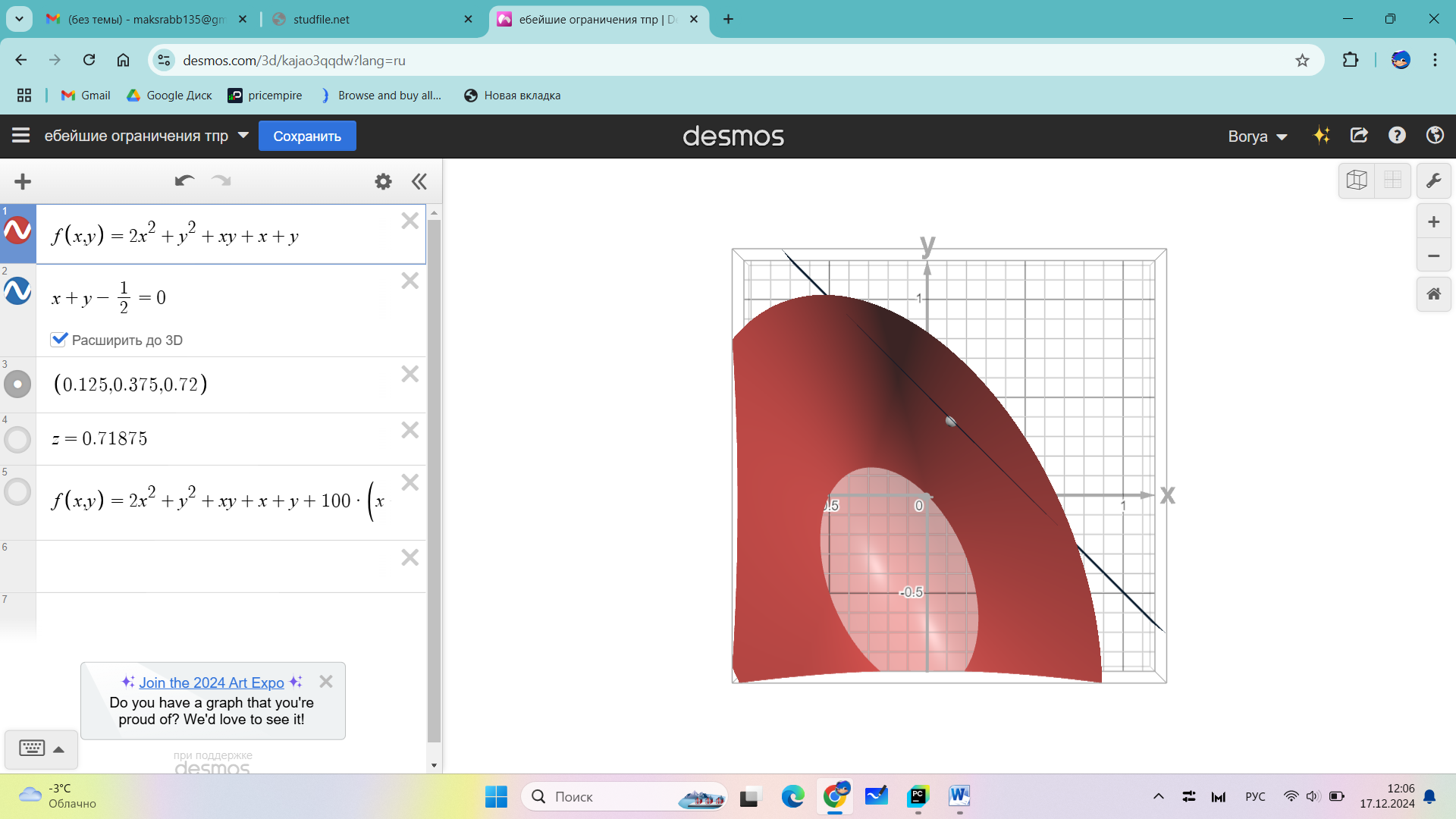


Рис. 1 – Пересечение функции  и плоскости 

Из графика видно, что функция принимает только одно минимальное значение, которое на плоскости находится выше нуля, и которое функция принимает при положительных переменных.

2. Краткое описание алгоритма методов

Применяется метод штрафных функций. Суть его заключается в том, чтобы расстояние от плоскости становится функцией штрафа, который возрастает при отдалении точки от плоскости ограничений.

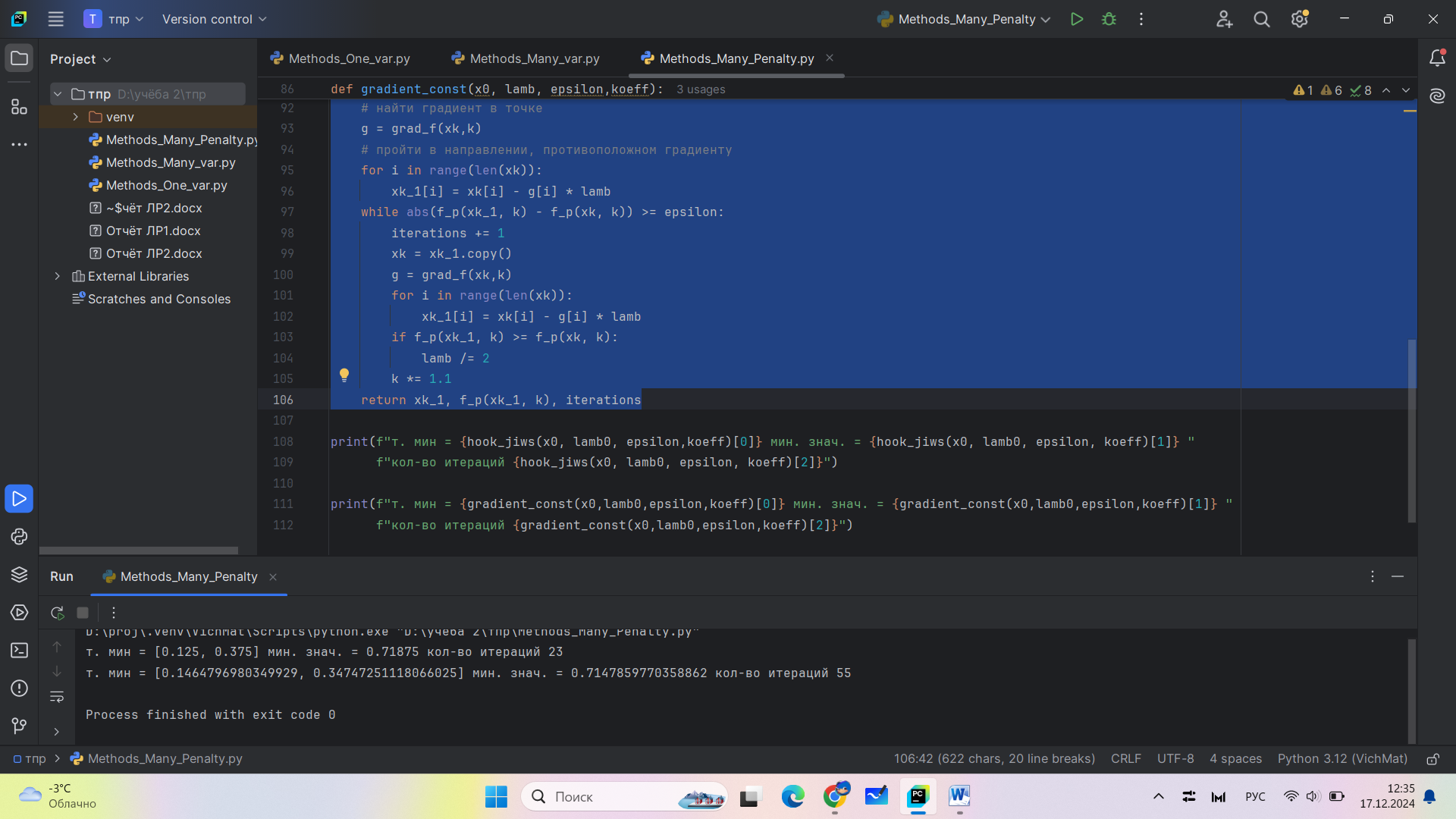
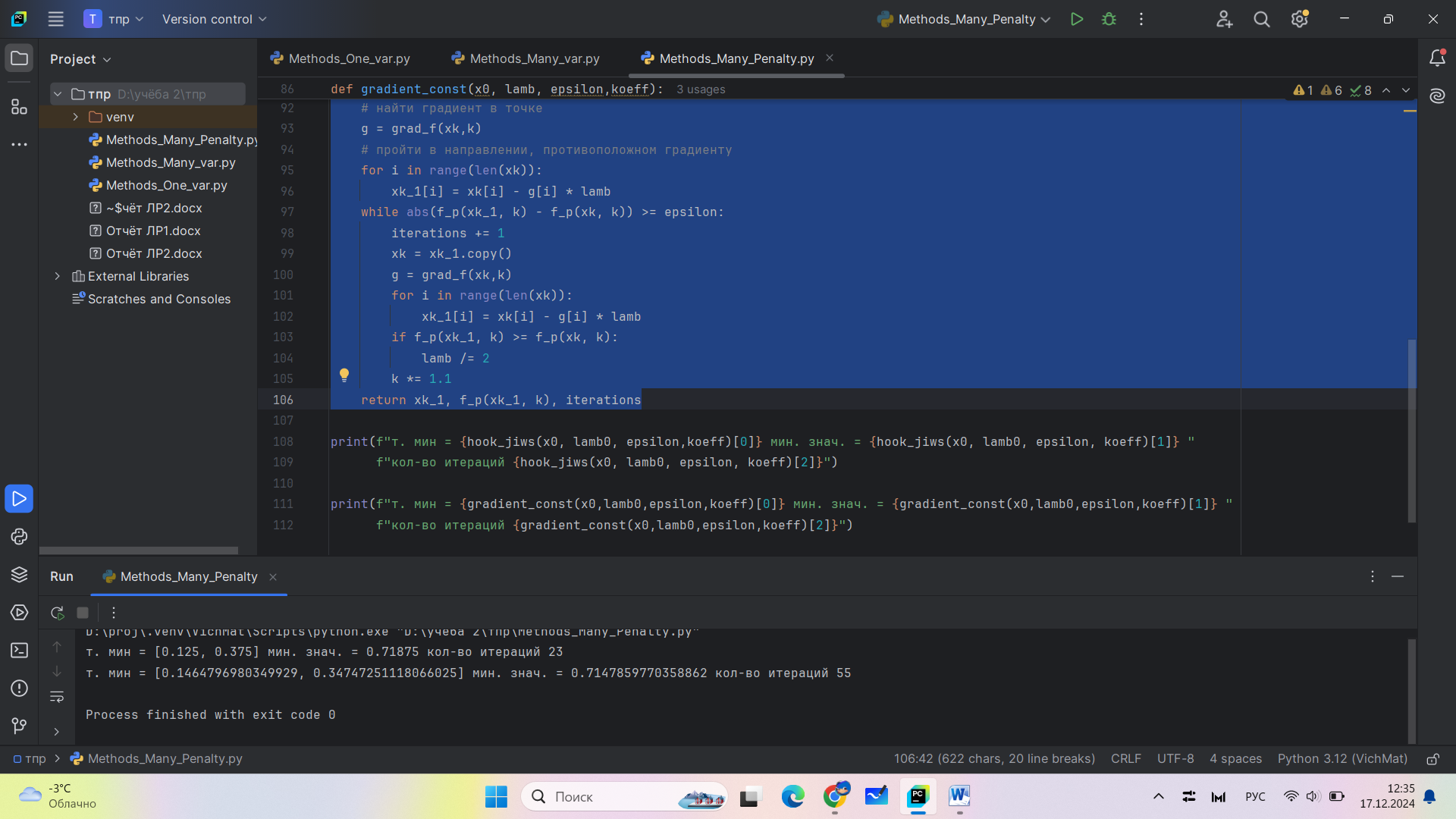
Метод градиентного спуска с постоянным шагом:

Из-за существования дополнительной функции частная производная также должна быть дополнена для поиска функции. Функция становится овражной, поэтому поиск занимает существенно большее количество итераций.

def grad\_f(x,k):  
 # Чисто из-за того, как заданы ограничения, можно составить производную  
 x1 = 4 \* x[0] + x[1] + 1 + k \* 2 \* (x[0]+x[1] - 0.5)  
 x2 = 2 \* x[1] + x[0] + 1 + k \* 2 \* (x[0]+x[1] - 0.5)  
 return [x1,x2]

def gradient\_const(x0, lamb, epsilon,koeff):  
 # точки  
 xk = x0.copy()  
 xk\_1 = xk.copy()  
 iterations = 1  
 k = koeff  
 # найти градиент в точке  
 g = grad\_f(xk,k)  
 # пройти в направлении, противоположном градиенту  
 for i in range(len(xk)):  
 xk\_1[i] = xk[i] - g[i] \* lamb  
 while abs(f\_p(xk\_1, k) - f\_p(xk, k)) >= epsilon:  
 iterations += 1  
 xk = xk\_1.copy()  
 g = grad\_f(xk,k)  
 for i in range(len(xk)):  
 xk\_1[i] = xk[i] - g[i] \* lamb  
 if f\_p(xk\_1, k) >= f\_p(xk, k):  
 lamb /= 2  
 k \*= 1.1  
 return xk\_1, f\_p(xk\_1, k), iterations

Результаты работы метода:

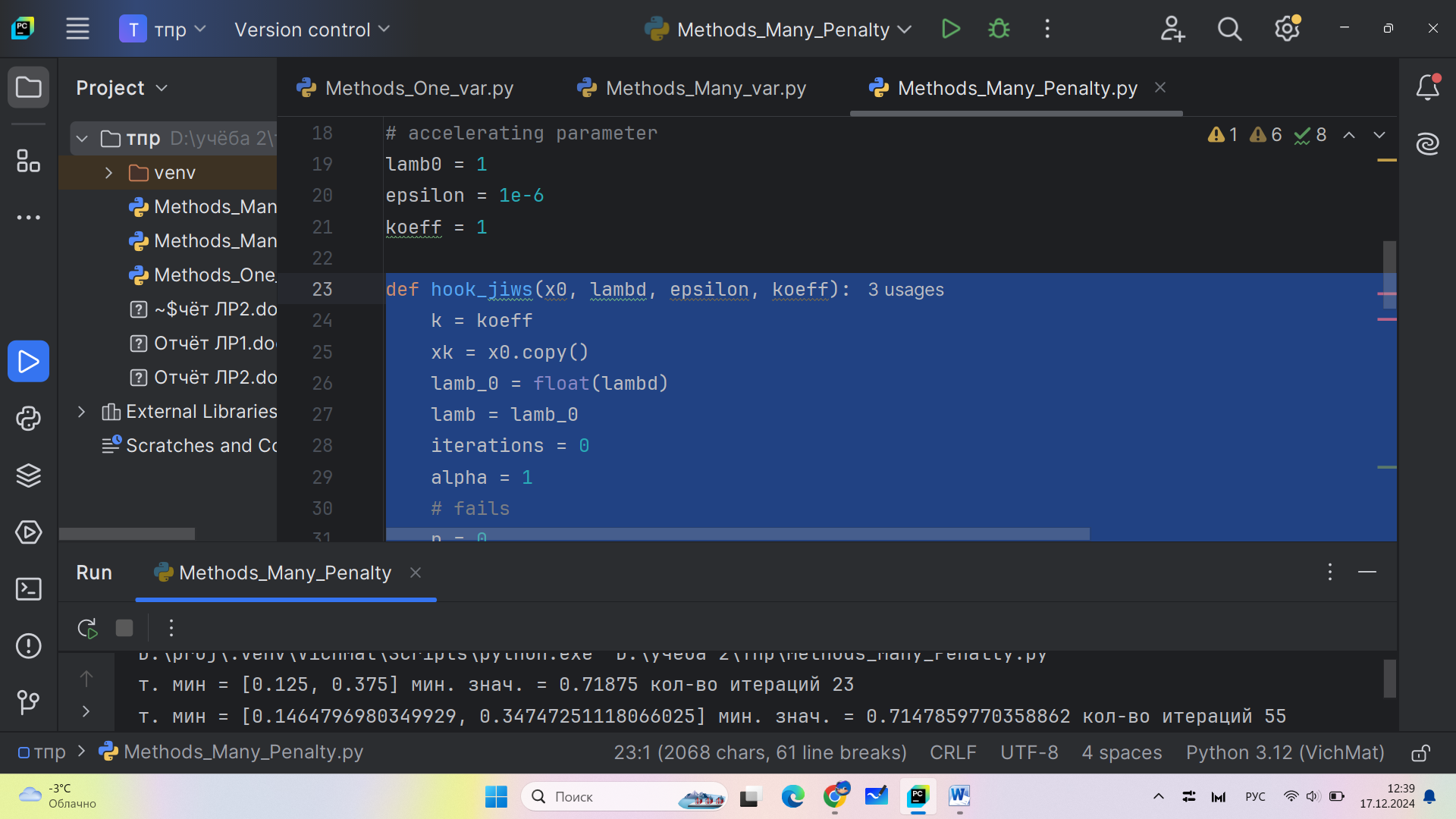


Метод Хука-Дживса:

Применимость этого метода не уменьшилась в сравнении с предыдущим – итераций достаточно, чтобы функция с большим количеством переменных оптимизировалась долгое количество времени.

def hook\_jiws(x0, lambd, epsilon, koeff):  
 k = koeff  
 xk = x0.copy()  
 lamb\_0 = float(lambd)  
 lamb = lamb\_0  
 iterations = 0  
 alpha = 1  
 # fails  
 p = 0  
 while lamb >= epsilon:  
 iterations += 1  
 elems = [xk]  
 # пройти по координатам точки  
 for i in range(len(xk)):  
 temp\_elems = []  
 # создать по 3 точки для каждой из существующих на этапе работы с конкретной координатой  
 for elem in elems:  
 x1 = elem.copy()  
 # это для xi - λ  
 x1[i] -= lamb  
 x2 = elem.copy()  
 x3 = elem.copy()  
 # это для xi + λ  
 x3[i] += lamb  
 temp\_elems.append(x1)  
 temp\_elems.append(x2)  
 temp\_elems.append(x3)  
 # и закинуть новые точки к существующим  
 elems = elems + temp\_elems  
 temp\_elems.clear()  
 # удалить все повторяющиеся точки  
 for elem in elems:  
 if elems.count(elem)!=1:  
 elems.remove(elem)  
 # удалить из списка точек начальную для определения направления  
 elems.remove(xk)  
 # найти минимумы функции  
 func\_values = []  
 for elem in elems:  
 func\_values.append(f\_p(elem,k))  
 min\_f = min(func\_values)  
 min\_x = elems[func\_values.index(min\_f)]  
 f\_x = f\_p(xk,k)  
 # и сравнить значений  
 if round(min\_f,4) >= round(f\_x,4):  
 p += 1  
 # не получилось, надо изменить "шаг"  
 # lamb = lamb - (lamb\_0 / m.exp(p))  
 lamb /= 2.0  
 else:  
 xk = xk  
 for i in range(len(min\_x)):  
 # получилось, по формуле x0 переходит в следующую итерацию  
 min\_x[i] -= xk[i]  
 min\_x[i] \*= alpha  
 xk[i] = min\_x[i] + xk[i]  
 p = 0  
 # очистить списки точек, все операции в итерации завершены  
 func\_values.clear()  
 elems.clear()  
 k \*= 10  
 return xk, f\_p(xk,k), iterations

Результат работы метода:



Выводы о работе методов

Метод штрафных функций через определённые преобразования позволяет находить условные экстремумы функций. Однако поиск может быть осложнён трудностями в поиске коэффициента штрафной функции.