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 «Линейное программирование»

по дисциплине «Теория принятия решений»

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

Проверил: доцент каф. ВММБ Бояршинова И. Н.

Пермь, 2024

1. Постановка задачи

Решить задачу линейного программирования:

F=3x1 – 2x2 → max;

2x1 + x2 ≤ 11,

–3x1 + 2x2 ≤ 10,

3x1 + 4x2 ≥ 20,

x1, x2 ≥ 0.

Данную задачу можно решить 2 способами: симплекс-методом и графическим.

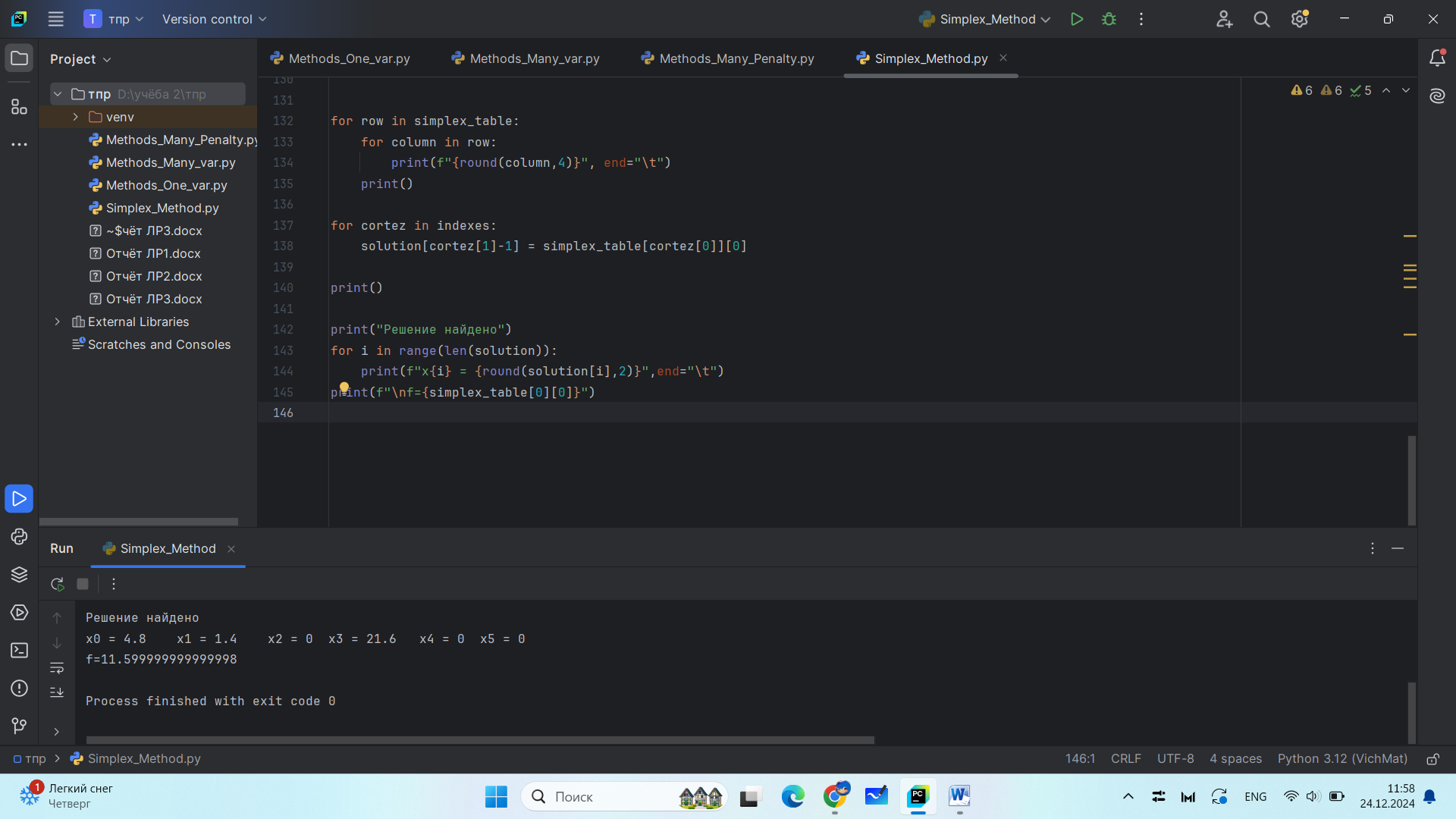
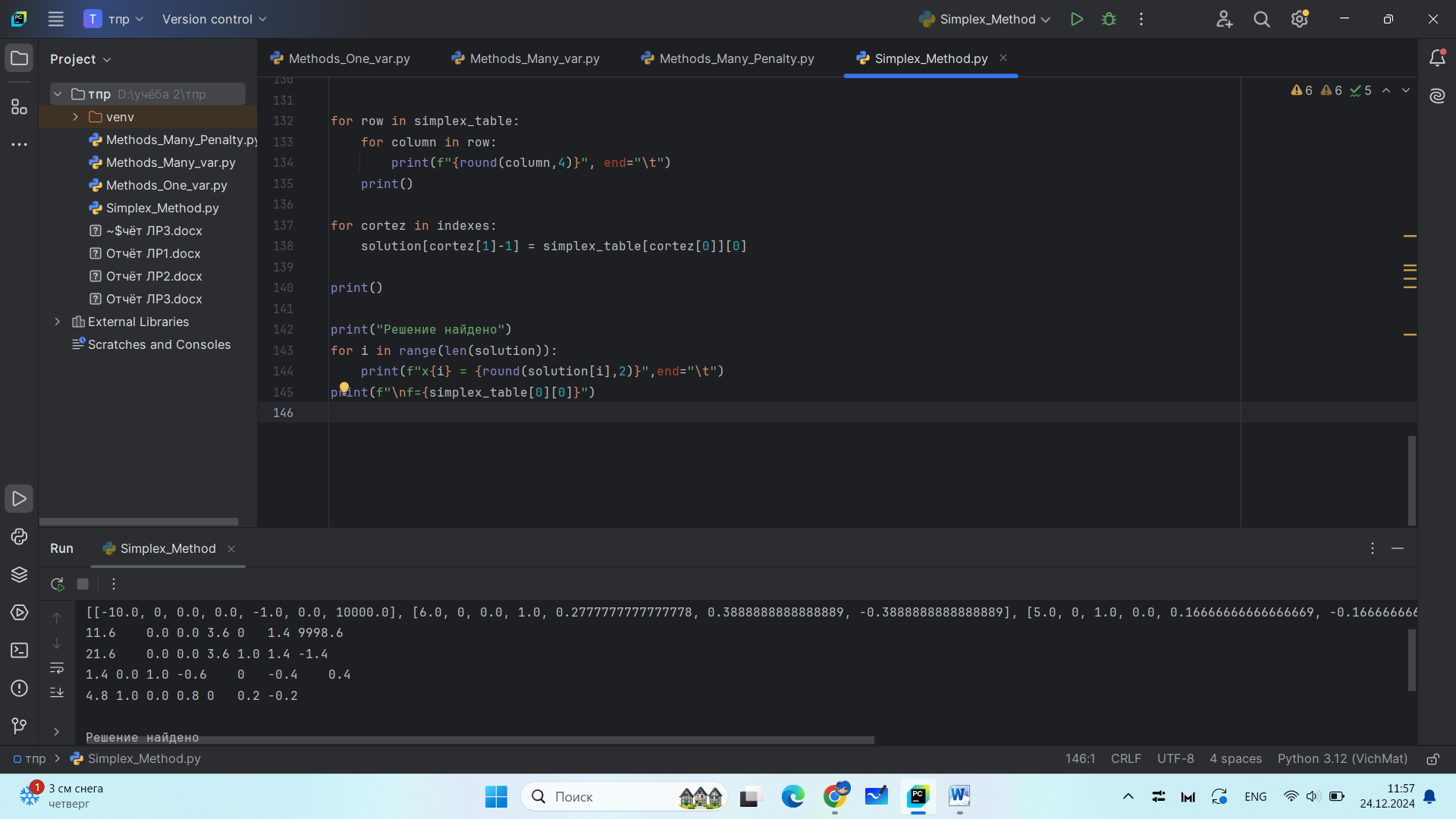
Из графика видно, что функция принимает только одно минимальное значение, которое на плоскости находится выше нуля, и которое функция принимает при положительных переменных.

2. Краткое описание алгоритма методов

Применяется симплекс-метод. Суть его заключается в том, что строится таблица коэффициентов перед базисными переменными и при помощи выбора ведущего элемента значения в таблице вычисляются до тех пор, пока не будет найдено оптимальное решение или не будет доказано, что задача решений не имеет.

M = 1e4  
  
simplex\_table = [  
 [-20\*M,-3\*M-3,-4\*M+2,0,0,M,0],  
 [11,2,1,1,0,0,0],  
 [10,-3,2,0,1,0,0],  
 [20,3,4,0,0,-1,1]  
]  
  
solution = [0,0,0,0,0,0]  
indexes = []  
  
def find\_leading\_column(matrix):  
 temp\_matrix = matrix[0].copy()  
 temp\_matrix.pop(0)  
 lead\_column = temp\_matrix.index(min(temp\_matrix))  
 return lead\_column + 1  
  
def find\_leading\_row(matrix):  
 lead\_column = find\_leading\_column(matrix)  
 quotients = []  
 for i in range(1, len(matrix)):  
 if matrix[i][lead\_column] > 0:  
 quotients.append(matrix[i][0]/matrix[i][lead\_column])  
 else:  
 quotients.append(1e8)  
 lead\_row = quotients.index(min(quotients))  
 return lead\_row + 1  
  
def write\_new\_table(matrix):  
 lead\_row = find\_leading\_row(matrix)  
 lead\_column = find\_leading\_column(matrix)  
 new\_matrix = []  
 matrix\_row = []  
 # YOSHA!  
 lead\_element = matrix[lead\_row][lead\_column]  
 for i in range(len(matrix)):  
 if i != lead\_row:  
 for j in range(len(matrix[0])):  
 if j != lead\_column:  
 matrix\_row.append(  
 matrix[i][j] - (matrix[i][lead\_column] \* matrix[lead\_row][j]) / lead\_element  
 )  
 else:  
 matrix\_row.append(0)  
 else:  
 for j in range(len(matrix[0])):  
 matrix\_row.append(matrix[i][j] / lead\_element)  
 # print(matrix\_row)  
 new\_matrix.append(matrix\_row.copy())  
 matrix\_row.clear()  
 return new\_matrix  
  
# чисто проверка на неотрицательность элементов первой строки  
def simplex\_done(matrix):  
 for i in range(len(matrix[0])):  
 if matrix[0][i] < 0:  
 return False  
 return True  
  
def simplex\_unsolving(matrix):  
 for i in range(len(matrix)):  
 for j in range(len(matrix[i])):  
 if matrix[i][j] > 0:  
 return False  
 return True  
  
while not(simplex\_done(simplex\_table)):  
 if simplex\_unsolving(simplex\_table):  
 print('решений у задачи нет')  
 break  
 print(simplex\_table)  
 indexes.append((find\_leading\_row(simplex\_table), find\_leading\_column(simplex\_table)))  
 simplex\_table = write\_new\_table(simplex\_table)  
  
  
for row in simplex\_table:  
 for column in row:  
 print(f"{round(column,4)}", end="\t")  
 print()  
  
for cortez in indexes:  
 solution[cortez[1]-1] = simplex\_table[cortez[0]][0]  
  
print()  
  
print("Решение найдено")  
for i in range(len(solution)):  
 print(f"x{i} = {round(solution[i],2)}",end="\t")  
print(f"\nf={simplex\_table[0][0]}")

Результаты работы метода:



Графический способ:

Его применимость ограничивается мерностью пространства поиска решений

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.