

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И
РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2

По дисциплине «Методы оптимизации и управления»
По теме «Основная фаза симплекс-метода»

Выполнил:
студент гр. 753502
Василюк В.И.
Проверил:
Дугинов О.И.

Минск 2020

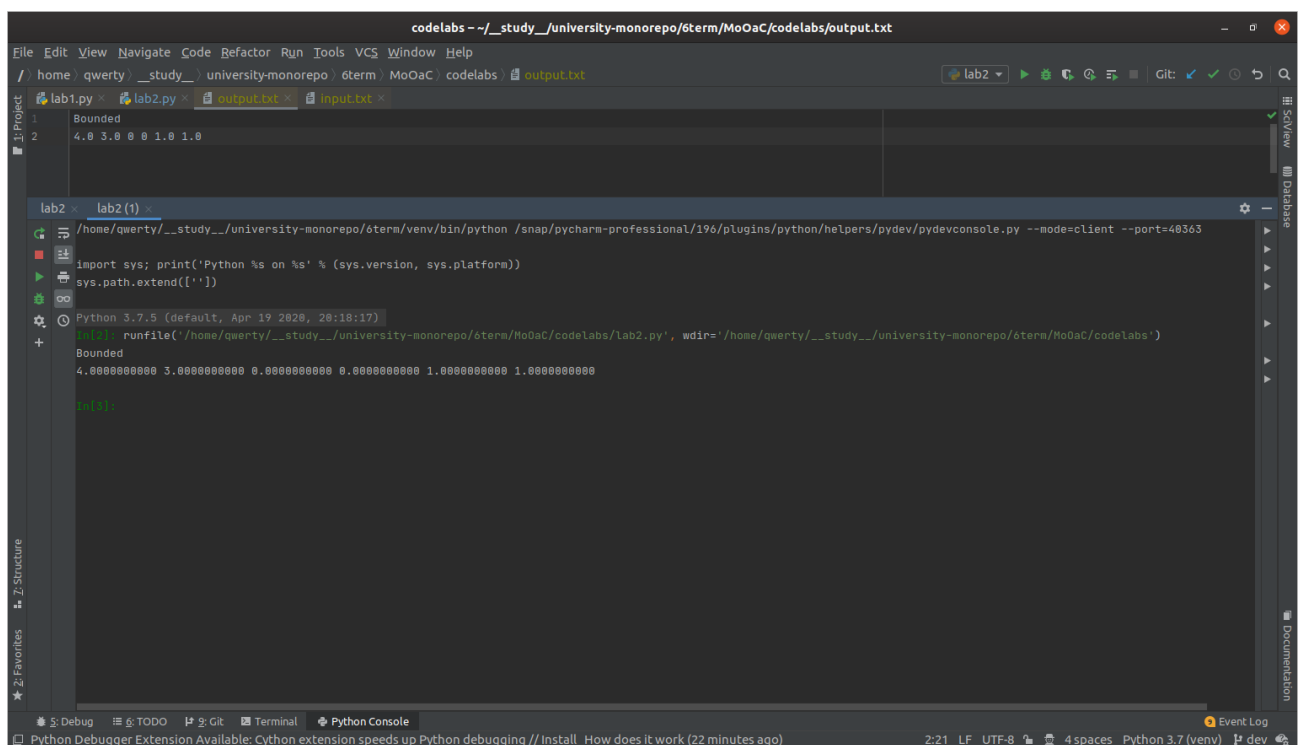
Введение

Целью данной лабораторной работы было изучение и программная реализация основной фазы симплекс-метода.

1. Теоретические сведения

Симплекс-метод — алгоритм решения оптимизационной задачи линейного программирования путём перебора вершин выпуклого многогранника в многомерном пространстве. Симплекс-метод позволяет эффективно найти оптимальное решение, избегая простой перебор всех возможных угловых точек. Основной принцип метода: вычисления начинаются с какого-то «стартового» базисного решения, а затем ведется поиск решений, «улучшающих» значение целевой функции. Это возможно только в том случае, если возрастание какой-то переменной приведет к увеличению значения функционала.

2. Результаты выполнения программы



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
/ home qwerty / __study__ / university-monorepo / 6term / MoOaC / codelabs / output.txt
lab1.py lab2.py output.txt input.txt
1 Bounded
2 4.0 3.0 0 0 1.0 1.0

lab2 lab2 (1)
/home/qwerty/__study__/university-monorepo/6term/venv/bin/python /snap/pycharm-professional/196/plugins/python/helpers/pydev/pydevconsole.py --mode=client --port=40363
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend([''])
Python 3.7.5 (default, Apr 19 2020, 20:18:17)
lab2: runfile('/home/qwerty/__study__/university-monorepo/6term/MoOaC/codelabs/lab2.py', wdir='/home/qwerty/__study__/university-monorepo/6term/MoOaC/codelabs')
Bounded
4.000000000 3.000000000 0.000000000 0.000000000 1.000000000 1.000000000
lab2:

Debug TODO Git Terminal Python Console Event Log
Python Debugger Extension Available: Cython extension speeds up Python debugging // Install How does it work (22 minutes ago) 2:21 LF UTF-8 4 spaces Python 3.7 (venv) dev
```

3. Программный код

```
import numpy as np
import math

def main_phase_simplex_method(matrix_A, x, Jb, c):
    transposed_matrix_a = np.array(get_transposed_matrix(matrix_A))
    basis_matrix = np.array(get_basis_matrix(transposed_matrix_a, Jb))
    inverse_basis_matrix = np.linalg.inv(np.array(basis_matrix))
    column = 0
    vector = []

    is_first_iteration = True

    while True:
        if not is_first_iteration:
            inverse_basis_matrix = get_optimized_inverse_matrix(basis_matrix,
            inverse_basis_matrix, column, vector)
            basis_matrix = np.array(get_basis_matrix(transposed_matrix_a, Jb))
        else:
            is_first_iteration = False

        potential_vector = get_potential_vector(inverse_basis_matrix, c, Jb)
        delta = get_delta(potential_vector, matrix_A, c)

        # Решение оптимально, выход из цикла
        if is_optimal_solution(delta, Jb):
            print('Bounded')
            print(*["{0:0.10f}".format(i) for i in x])
            # with open('output.txt', 'w') as f:
            #     f.write('Bounded\n')
            #     for elem in x:
            #         f.write(str(elem) + ' ')
            return

        j0 = get_index_of_first_negative_element(delta)
        z = inverse_basis_matrix @ transposed_matrix_a[j0]
```

```

theta = get_vector_theta(z, x, Jb)

if not have_solution(theta):
    print('Unbounded')
    # with open('output.txt', 'w') as f:
    #     f.write('Unbounded')
    return

index_min, theta0 = min(enumerate(theta)
                        , key=lambda pair: pair[1])
Jb[index_min] = j0 + 1
# Defining of vector and column number for optimized inversion of matrix
column = index_min
vector = matrix_A[:, j0]
x = get_new_plan_x(x, theta0, Jb, j0, z)

def get_transposed_matrix(matrix):
    return [list(elem) for elem in zip(*matrix)]

def get_basis_matrix(transposed_matrix, Jb):
    transposed_basis_matrix = [transposed_matrix[index - 1] for index in Jb]
    return get_transposed_matrix(transposed_basis_matrix)

def get_potential_vector(inverse_basis_matrix, c, Jb):
    basis_c = [c[index - 1] for index in Jb]
    return basis_c @ inverse_basis_matrix

def get_delta(potential_vector, matrix_A, c):
    return potential_vector @ matrix_A - c

def is_optimal_solution(delta, Jb):
    for index, value in enumerate(delta):
        if index + 1 not in Jb and value < 0:
            return False
    return True

def get_index_of_first_negative_element(list):
    return next(index for index, value in enumerate(list) if value < 0)

def get_vector_theta(z, x, Jb):
    return [x[Jb[index] - 1] / value if value > 0 else math.inf for index, value in enumerate(z)]

def have_solution(theta):
    for elem in theta:
        if elem is not math.inf:
            return True
    return False

def get_new_plan_x(x, theta0, Jb, j0, z):

```

```

new_plan_x = [0] * len(x)
for index, value in enumerate(Jb):
    new_plan_x[value - 1] = x[value - 1] - theta0 * z[index]
new_plan_x[j0] = theta0
return new_plan_x

```

```

def get_optimized_inverse_matrix(matrix, inverse_matrix, column, vector):
    l = inverse_matrix @ vector

```

```

    if l[column] == 0:
        return None

```

```

    l_tilda = np.copy(l)
    l_tilda[column] = -1

```

```

    l_upper = -1. / l[column] * l_tilda
    q = np.eye(len(matrix))
    q[:, column] = l_upper

```

```

    answer = q @ inverse_matrix

```

```

    return answer

```

```

def get_list_int_from_string(string):
    return [int(elem) for elem in string.split()]

```

```

if __name__ == '__main__':
    matrix_A = []
    b = []
    x = []
    Jb = []
    c = []

```

```

    with open('input.txt', 'r') as f:
        m, n = [int(elem) for elem in f.readline().split()]
        for _ in range(m):
            matrix_A.append(get_list_int_from_string(f.readline()))
        matrix_A = np.array(matrix_A)
        b = np.array(get_list_int_from_string(f.readline()))
        c = np.array(get_list_int_from_string(f.readline()))
        x = np.array(get_list_int_from_string(f.readline()))
        Jb = np.array(get_list_int_from_string(f.readline()))

```

```

    main_phase_simplex_method(matrix_A, x, Jb, c)

```

Выводы

В данной лабораторной работе было реализовано выполнение основной фазы симплекс-метода. Основную фазу симплекс-метода целесообразно применять, когда имеется стартовый базисный допустимый план или была реализована начальная фаза симплекс метода.