МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Лабораторна робота № 9

З дисципліни

“Математичні методи дослідження операцій”

Виконав:

Студент групи КН-314

Ляшеник Остап

Прийняв

Шиманський Володимир Михайлович

Львів - *2023*

Постановка завдання

```
import numpy as np
from sympy import *
from sympy.abc import x, y, t
import math
from prettytable import PrettyTable
from scipy.optimize import minimize


def tableu(A, b, c, eq):
    A1 = []
    zeroes = []
    for i in range(len(eq)):
        if eq[i] == '>=':
            x = -1 * A[i]
            A1.append(x)
            b[i] = (-1) * b[i]
        else:
            A1.append(A[i])
    # zeroes should be initialized as an empty list
    zeroes = []
    # appending zeros to zeroes
    for i in range(len(eq)):
        zero = [0] * len(eq)
        zero[i] = 1
        zeroes.append(zero)
    zeroes = np.transpose(zeroes)
    table = np.hstack((A1, zeroes))
    model = [0] * len(table[0])
    for i in range(len(c)):
        model[i] = c[i]
    return table, b, model


def find_basis_variables(A):
    A_T = A.T
    constraint = np.sum(A_T == 1, axis=1) == 1
    constraint &= np.sum(A_T == 0, axis=1) == A_T.shape[1] - 1
    indexes = np.where(constraint)[0]
    variables_order = np.zeros(len(indexes))
    for idx in indexes:
        right = np.where(A_T[idx] == 1)
        variables_order[right] = idx
    return variables_order


def simplex(c, A, b):
    basic_variables = find_basis_variables(A)
    while True:
```

```
        cb = np.array([c[int(i)] for i in basic_variables])
        dot_product = np.dot(cb, A)
        for i in range(len(dot_product)):
            dot_product[i] -= c[i]
        if np.all(dot_product <= 0):
            return basic_variables, cb, np.dot(cb, b.T), b
        main_col = np.argmax(dot_product, axis=0)
        if np.all(A[:, main_col] <= 0):
            print("Can not solve this!")
            return None
        rel = np.zeros_like(b)
        np.divide(b, A[:, main_col], out=rel, where=A[:, main_col] > 0)
        rel[A[:, main_col] <= 0]
        rel1 = np.copy(rel)
        for i in range(len(rel1)):
            if rel1[i] == 0:
                rel1[i] = 10000000
                break
        main_row = np.argmin(rel1, axis=0)
        pivot = A[main_row][main_col]
        old_var = int(basic_variables[main_row])
        basic_variables[main_row] = main_col
        old_col = np.copy(A[:, old_var])
        b_old = np.copy(b)
        for i in range(len(b)):
            b[i] = (b_old[i] * pivot - b_old[main_row] * A[i][main_col]) / pivot
        b[main_row] = b_old[main_row] / pivot
        for i in range(len(b)):
            for j in range(len(c)):
                if i != main_row and j not in basic_variables:
                    A[i][j] = (pivot * A[i][j] - A[i][main_col] * A[main_row][j]) / pivot
        A[main_row] = A[main_row] / pivot
        A[:, main_col] = old_col


def check_basis(a):
    rank = np.linalg.matrix_rank(a)
    if rank == table.shape[1]:
        return True
    return False


def add_artificial_variables(A, c):
    m, n = A.shape
    basis = np.eye(m)
    aset = set([tuple(x) for x in A.T])
    bset = set([tuple(x) for x in basis])
```

```python
    cols = np.array([x for x in bset - aset])
    result = np.hstack((A, cols.T))
    num_variables = len(cols)
    M = 10000000
    additional_c = np.array([M] * num_variables)
    return result, np.hstack((c, additional_c)), num_variables


def get_iteration(n):
    i = 1
    j, c = i, 0
    while i <= n:
        k = i
        i = j
        j = k + i
        c += 1
    return c


def fibonacci(n):
    if n == 1:
        return 1
    else:
        n1 = 1
        n2 = 2
        for i in range(3, n + 1):
            n1, n2 = n2, n1 + n2
        return n2


def get_iteration(n):
    i = 1
    j, c = i, 0
    while i <= n:
        k = i
        i = j
        j = k + i
        c += 1
    return c


def fibonacci_method(f, eps):
    a, b = -1000000, 1000000
    vars = round((b - a) / eps)
    n = get_iteration(vars) + 1
    k = 0
    while True:
        x1 = a + (fibonacci(n - k - 1) / fibonacci(n - k + 1)) * (b
- a)
```

```
        x2 = a + (fibonacci(n - k) / fibonacci(n - k + 1)) * (b - a)
        if f(x1) <= f(x2):
            b = x2
            x2 = x1
        else:
            a = x1
            x1 = x2
        k += 1
        if abs(b - a) <= eps:
            break
    x_min = (a + b) / 2
    return x_min


def print_all(dt_pr, rel, A, b):
    print("Оцінковий рядок :")
    print(dt_pr)
    print("xв/xr :")
    for row in rel.T:
        print(row)
    print(A)
    print("План")
    print(b)
    print("-----------------")


def func(x):
    return x[0] ** 2 + x[1] ** 2 - 10 * x[0] - 20 * x[1]

def g(x, y):
    return 9*x + 8*y - 72

def g1(x, y):
    return x + 2*y - 10


x1, x2 = 2, 1
eps = 0.0000001
A = np.array([[3, 2], [1, 2]], dtype=np.float64)
b = np.array([1, 4], dtype=np.float64)
eq = ["=<", "=<"]
flag = False
# перевірка на обмеження
if g(x1, x2) <= 0 and g1(x1, x2) <= 0:
    flag = True
F = x ** 2 + y ** 2 - x - 4
if flag:
    k = 0
    dx_x = diff(F, x)
```

```
dx_y = diff(F, y)
Fx_0 = lambdify((x, y), dx_x)
Fy_0 = lambdify((x, y), dx_y)
while True:
    k += 1
    print("-----------------------------------------------")
    print("Step", k)
    b1 = np.copy(b)
    # лінеалізуємо функцію
    Fx = diff(F, x)
    Fy = diff(F, y)
    Fx_f = lambdify(x, Fx)
    Fy_f = lambdify(y, Fy)
    x_x0 = x - x1
    x_y0 = y - x2
    cx = Fx_f(x1) * x_x0 + Fy_f(x2) * x_y0
    cx_dx = diff(cx, x)
    cx_dy = diff(cx, y)
    c = np.array([cx_dx, cx_dy], dtype=np.float64)
    # симплек метод
    table, b, model = tableu(A, b, c, eq)
    vars, cb, res, plan = simplex(model, table, b1)

   ans = [0] * 2
    for i in range(len(vars)):
        if int(vars[i] + 1) <= 2:
            ans[int(vars[i])] = plan[i]

    h_1 = ans[0] - x1  # напрям спуску
    h_2 = ans[1] - x2
    b_1 = h_1 * t
    b_2 = h_2 * t
    x_1 = x1 + b_1
    x_2 = x2 + b_2
    F_xy = lambdify((x, y), F)
    F_xy_1 = F_xy(x_1, x_2)
    F_xy_2 = lambdify(t, F_xy_1)
    # beta
    solver = fibonacci_method(F_xy_2, eps)
    # 0 > beta <= 1
    if solver <= 1:
        arg_min = solver
    else:
        arg_min = 1
    # xk наближення
    x_n_1 = x1 + (arg_min * h_1)
    y_n_1 = x2 + (arg_min * h_2)
    # мінімум функції res = F_xy(x_n_1, y_n_1)
    # похибка
    x_x_1 = x1 - x_n_1
```

```python
        x_x_2 = x2 - y_n_1
        mod = math.sqrt(x_x_1 ** 2 + x_x_2 ** 2)
        x1 = x_n_1
        x2 = y_n_1
        table = PrettyTable()
        table.field_names = ["X1", "X2", "Res"]
        table.add_row([x1, x2, res])
        print(table)
        if mod <= eps:
            break
        print()
else:
    print("Початкові точки не виконують обмеження")
```

```
-----------------------------------------------
Step 1
+---------------------+---------------------+-----+
|          X1         |          X2         | Res |
+---------------------+---------------------+-----+
| 0.39999995026777846 | 0.19999997513388923 | 0.0 |
+---------------------+---------------------+-----+

-----------------------------------------------
Step 2
+--------------------+---------------------+-----------------------+
|         X1         |          X2         |          Res          |
+--------------------+---------------------+-----------------------+
| 0.3499999849811174 | 0.04999998602598579 | -0.06666669982148102 |
+--------------------+---------------------+-----------------------+

-----------------------------------------------
Step 3
+--------------------+-------------------+---------------------+
|         X1         |         X2        |          Res        |
+--------------------+-------------------+---------------------+
```

```
| 0.3499999891047572 | 0.049999998396913 | -0.1000000100125884 |
+--------------------+-------------------+---------------------+
```