Report

Team information.

- Team leader: Dmitrii Kuzmin
- Team member 1: Nazgul Salikhova - 5
- Team member 2: Yaroslava Bryukhanova - 5
- Team member 3: Apollinaria Chernikova - 5
- Team member 4: Mikahil Romanov - 5

Link to the product.

- The product is available: https://github.com/Post-Modern28/SimplexOptimization/tree/main/Interior_Point_Algo

Programming language.

- Programming language: Python

Linear programming problems:

Example of task #1

Maximize and minimize

$$F(x_1, x_2) = x_1 + x_2$$

subject to

$$\begin{cases} 2x_1 + 4x_2 \leq 16, & (1) \\ x_1 + 3x_2 \geq 9, & (2) \\ x_1 \geq 0, & (3) \\ x_2 \geq 0 & (4). \end{cases}$$

Output from Interrior-Point Algorithm: (minimization)

```
The initial solution is: [0.5 3.5 1.  2. ]
---------------------------
When a = 0.5:
After the 20 iterations the final answer is:
x1 = 7.999930898924811
x2 = 2.3033812944680903e-05
---------------------------
The value of objective function is: 7.999953932737756
---------------------------
When a = 0.9:
After the 10 iterations the final answer is:
x1 = 7.999977913870253
x2 = 9.2279083788721e-06
---------------------------
The value of objective function is: 7.9999871417786315
---------------------------
```

Output from Interrior-Point Algorithm: (maximization)

```
The initial solution is: [0.5 3.5 1.  2. ]
---------------------------
When a = 0.5:
After the 21 iterations the final answer is:
x1 = 1.2128164741504175e-05
x2 = 1.2128164742176472e-05
---------------------------
The value of objective function is: -2.4256329483680647e-05
---------------------------
When a = 0.9:
After the 11 iterations the final answer is:
x1 = 2.556246000960142e-06
x2 = 1.0050666615627886e-06
---------------------------
The value of objective function is: -3.5613126625229307e-06
---------------------------
```

Example of task #2

- Maximization

- Objective function: $5x_1 + 4x_2 \to max$

- Constraint functions: $\begin{cases} 6x_1 + 4x_2 \leq 24 \\ x_1 + 2x_2 \leq 6 \\ -x_1 + x_2 \leq 1 \\ x_j \geq 0,\ j \in \{1,2\} \end{cases}$

  Output from Simplex Method Algorithm:
  x1 = 3.0, x2 = 1.5
  Maximal value of the function: 21.0

  Output from Interrior-Point Algorithm:
  The final answer is:

The initial solution is: [3. 0. 6. 3. 4.]
When a = 0.5:
After the 17 iterations the final answer is:
x1 = 3.9999847413648997
x2 = 0.0
x3 = 9.1552734375e-05
When a = 0.9:
After the 7 iterations the final answer is:
x1 = 3.9999990005804587
x2 = 0.0
x3 = 5.999999999999985e-06
The value of objective function is: 19.999995002902292

Input

The input contains:

- A vector of coefficients of objective function - C. (type – float; enter coefficients in one line separated by a space)
- The number of constraints (type – integer)
- A matrix of coefficients of constraint function - A. (type – float, enter the coefficients separated by a space for each constraint on a new line)
- A vector of right-hand side numbers - b. (type – float, enter right-hand side numbers of each constraint in one line separated by a space)
- The approximation accuracy ϵ. (type – integer, enter the number of decimal places)
- Maximize/Minimize (Type – integer; 1 – maximize, 2 - minimize)
- Initial solution(Type – float)

Output/Results

The output contains:

- The string ”The method is not applicable!” or
- The string ”The problem does not have solution!” or
- A vector of decision variables - x^* by Interior-Point algorithm (when α = 0.5 and α = 0.9) and by Simplex method from programming Task 1.
- Maximum (minimum) value of the objective function when α = 0.5 and α = 0.9.

Code:

```
import numpy as np


def interior_point_algorithm(main_function: np.array, constraints_matrix:
np.array, init_point: np.array, a,
```

```python
                                 num_of_constraints) -> int:
    print("When a = " + str(a) + ":")
    c = main_function
    A = constraints_matrix
    D = np.diag(init_point)
    D_prev = D - 100
    n = 1
    diff = D - D_prev
    x = init_point
    r = max(np.amax(diff), np.amin(diff), key=abs)
    while abs(r) > 0.0001:
        # print(f"Iteration {n}:")
        x_old = x
        A_prime = A @ D
        c_prime = D @ c
        A_prime_def_inverse = A_prime.transpose() @ np.linalg.inv(A_prime @
A_prime.transpose()) @ A_prime
        I = np.eye(A_prime_def_inverse.shape[0])
        P = I - A_prime_def_inverse
        c_proj = P @ c_prime
        min_neg = np.min(c_proj)
        if min_neg >= 0:
            break
        v = abs(np.min(c_proj))
        y = np.ones(A_prime_def_inverse.shape[0]) + a / v * c_proj
        x = D @ y
        answer = x
        # print(answer)
        D = np.diag(x)
        diff = x - x_old
        r = np.linalg.norm(diff, ord=2)
        n += 1

    print("After the " + str(n) + " iterations" + " the final answer is:")
    for i in range(num_of_constraints):
        print(f"x{i + 1} = {answer[i]}")
    print("---------------------------")
    return answer


def prepare_for_algorithm(c_array, A_array, b_vector):
    n = len(b_vector)
    c_array = np.append(c_array, [0] * n)

    x = np.array(list(map(float, input("Enter the initial
solution:\n").split())))

    # Check the  method applicability
    A = np.array(A_array)
    b = np.array(b_vector)
    residuals = np.array([np.dot(A[i], x) - b[i] for i in range(n)])

    if np.all(residuals):
        print("The method is not applicable!")
        exit()
    else:
        print(f"The initial solution is: {x}")
        print("---------------------------")
    return c_array, x


if __name__ == "__main__":
    # INPUT
    A = []
    c = np.array(list(map(float, input("Enter the coefficients of the
function:\n").split())))
    num_of_constraints = int(input("How many constraints?\n"))
    print("Enter the coefficients of variables in constraints:")
    for i in range(num_of_constraints):
        A.append(list(map(float, input().split())))
        for j in range(num_of_constraints):
            if i == j:
                A[i].append(1)
            else:
                A[i].append(0)
    A = np.array(A)
    print("Enter the variables of right-hand side numbers in constraints:")
    b = np.array(list(map(float, input().split())))
    approximation_accuracy = int(input("Enter the approximation
accuracy(number of decimal places):\n"))

    alpha_1 = 0.5
```

```python
alpha_2 = 0.9

flag = int(input("maximize or minimize? [1/2]: \n"))
if flag == 1:
    pass
elif flag == 2:
    c = -c

c, x = prepare_for_algorithm(c, A, b)
answer = interior_point_algorithm(c, A, x, alpha_1, num_of_constraints)

objective_function = np.dot(c, answer)
print(f"The value of objective function is: {objective_function}")
print("---------------------------")

answer = interior_point_algorithm(c, A, x, alpha_2, num_of_constraints)

objective_function = np.dot(c, answer)
print(f"The value of objective function is: {objective_function}")
print("---------------------------")
```