Report 1

Team information.

- Team leader: Oleynik Maxim
- Team member 1: Lobov Gleb 5/5
- Team member 2: Fakhrutdinov Bulat 5/5
- Team member 3: Kachmazov Alexander 5/5
- Team member 4: Oleynik Maxim 5/5

  All members have made maximum participation

Link to the product.

- The product is available: https://github.com/Dart-NEW/IntroToOptimizationHW2

Programming language.

- Programming language: Python

Linear programming problem.

- Maximization or Minimization?
  Maximization

- Objective function:
  $z = 5x_1 + 4x_2$

- Constraint functions:

$$\begin{aligned} 6x_1 + 4x_2 &\leq 24, \\ x_1 + 2x_2 &\leq 6, \\ -x_1 + x_2 &\leq 1, \\ x_2 &\leq 2, \\ x_1 &\geq 0, \\ x_2 &\geq 0. \end{aligned}$$

Input

The input contains:

- A vector of coefficients of objective function - C.
- A matrix of coefficients of constraint function - A.
- A vector of right-hand side numbers - b.
- The approximation accuracy ϵ.

Output/Results

The output contains:

- The string "The method is not applicable!"
  or

- A vector of decision variables - X^*.
- Maximum (minimum) value of the objective function.

Code

Listing 1: - Python

```
import numpy as np

def not_applicable(n=None):
    print("The method is not applicable!", n)
    exit()

def project_to_feasible(x, A, b):
    while np.any(np.dot(A, x) > b):
        for i in range(len(b)):
            if np.dot(A[i], x) > b[i]:
                x -= (np.dot(A[i], x) - b[i]) / np.dot(A[i], A[i]) * A[i]
    return x

def compute_optimization(alpha, c, A, b, accuracy):
    x = np.ones(c.shape[0])
    x = project_to_feasible(x, A, b)
    while True:
        x_prev = x
        D = np.diag(x)
        A_ = np.dot(A, D)
        c_ = np.dot(D, c)
        I = np.eye(A_.shape[1])
        A_t = np.transpose(A_)
        A_A_t = np.dot(A_, A_t)
        inv = np.linalg.inv(A_A_t)
        P = np.subtract(I, np.dot(A_t, np.dot(inv, A_)))
        c_proj = np.dot(P, c_)
        v = abs(np.min(c_proj))
        x_ = np.ones(c.shape[0]) + (alpha/v) * c_proj
        x = np.dot(D, x_)
        if np.linalg.norm(np.subtract(x, x_prev), ord=2) < 0.0001:
            break

    optimum = 0
    results = []
    for i in range(len(x) - len(b)):
        optimum += c[i] * round(x[i], accuracy)
        results.append((f"x{i + 1}", round(x[i], accuracy)))
    return results, round(optimum, accuracy)

def row_operation(matrix, pivot_row, pivot_col):
    matrix[pivot_row] = [f"x{pivot_col}"] + [item / matrix[pivot_row][pivot_col
    for i in range(len(matrix)):
        if i != pivot_row:
            factor = -matrix[i][pivot_col] / matrix[pivot_row][pivot_col]
            for j in range(1, len(matrix[i])):
                matrix[i][j] = matrix[pivot_row][j] * factor + matrix[i][j]

def check_unboundedness(matrix, column):
    for row in range(1, len(matrix)):
        if matrix[row][column] > 0:
            return True
    not_applicable(1)
```

```
def simplex_method(c, A, b, accuracy):
    rows, cols = len(A), len(A[0])
    matrix = [[0] * (cols + rows + 2) for _ in range(rows + 1)]
    matrix[0][1:cols + 1] = [-i for i in c]

    for i in range(rows):
        matrix[i + 1][1:cols + 1] = A[i]
        matrix[i + 1][0] = f"s{i + 1}"
        matrix[i + 1][cols + i + 1] = 1
        matrix[i + 1][-1] = b[i]

    while True:
        min_val = 0
        pivot_row = 0
        pivot_col = 0
        for i in range(1, len(matrix[0])):
            check_unboundedness(matrix, i)
            if matrix[0][i] < min_val:
                min_val = matrix[0][i]
                pivot_col = i

        if min_val != 0:
            min_positive = float('inf')
            for i in range(1, rows + 1):
                try:
                    ratio = matrix[i][-1] / matrix[i][pivot_col]
                except ZeroDivisionError:
                    ratio = -1
                if 0 < ratio < min_positive:
                    min_positive = ratio
                    pivot_row = i
            row_operation(matrix, pivot_row, pivot_col)
        else:
            break

    results = []
    for i in range(1, len(matrix)):
        if matrix[i][0][0] == "x":
            results.append((matrix[i][0], round(matrix[i][-1], accuracy)))
    return results, round(matrix[0][-1], accuracy)

def get_input():
    n = int(input("Enter number of coefficients of objective function: "))
    print(f"Enter coefficients (one in each of the {n} lines):")
    c = np.array([float(input()) for _ in range(n)], float)
    rows = int(input("Enter number of rows of matrix: "))
    cols = int(input("Enter number of columns of matrix: "))
    if cols != n:
        not_applicable()
    print(f"Enter matrix {rows} {cols} (one row of the matrix in each of the {
    A = []
    for _ in range(rows):
        A.append(list(map(float, input().split())))
    print(f"Enter right-hand side numbers (one in each of the {rows} lines):")
    b = [float(input()) for _ in range(rows)]
    accuracy = int(input("Enter approximation accuracy: "))
    return c, A, b, accuracy

c, A, b, accuracy = get_input()
```

```
print("\n--- Projection Method Results (alpha = 0.5) ---")
proj_results_05, proj_optimum_05 = compute_optimization(0.5, c, A, b, accuracy)
for var, value in proj_results_05:
    print(f"{var}: {value}")
print("Maximum value:", proj_optimum_05)

print("\n--- Projection Method Results (alpha = 0.9) ---")
proj_results_09, proj_optimum_09 = compute_optimization(0.9, c, A, b, accuracy)
for var, value in proj_results_09:
    print(f"{var}: {value}")
print("Maximum value:", proj_optimum_09)

print("\n--- Simplex Method Results ---")
simplex_results, simplex_optimum = simplex_method(c, A, b, accuracy)
for var, value in simplex_results:
    print(f"{var}: {value}")
print("Maximum value:", simplex_optimum)
```