Assignment 1

Team Information

- **Team Leader:** Egor Chernobrovkin
- **Team Member 1:** Dmitrii Kuznetsov
- **Team Member 2:** Alexandra Starikova-Nasibullina

Link to the Product

- The product is available at: GitHub
- We prepared Jupyter Notebook `tests.ipynb` that contains 8 test cases for the Simplex method.

Programming Language

- **Programming Language:** Python

Linear Programming Problem

Problem 1

- **Maximization or Minimization?** Maximization
- **Objective Function:** Maximize
$$Z = 40x_1 + 30x_2$$
- **Constraint Functions:**
$$\begin{aligned} x_1 + x_2 &\leq 12 \\ 2x_1 + x_2 &\leq 16 \end{aligned}$$

Input

- **C** = [40, 30]
- **A** = [[1, 1], [2, 1]]
- **b** = [12, 16]
- **Accuracy** = 0.1

Output

- Maximum value of
$$Z = 33$$
at
$$x_1 = 4, x_2 = 5$$

Problem 2

- **Maximization or Minimization?** Maximization
- **Objective Function:** Maximize
$$Z = 2x_1 + 5x_2$$
- **Constraint Functions:**
$$\begin{aligned} x_1 + 4x_2 &\leq 24 \\ 3x_1 + 1x_2 &\leq 21 \\ x_1 + x_2 &\leq 9 \end{aligned}$$

Input

- **C** = [2, 5]
- **A** = [[1, 4], [3, 1], [1, 1]]
- **b** = [24, 21, 9]
- **Accuracy** = 0.5

Output

- Maximum value of
$$Z = 33$$
at
$$x_1 = 4, x_2 = 5$$

Problem 3

- **Maximization or Minimization?** Maximization
- **Objective Function:** Maximize
$$Z = x_1 + 2x_2 + 3x_3$$
- **Constraint Functions:**
$$\begin{aligned} x_1 + x_2 + x_3 &\leq 12 \\ 2x_1 + x_2 + 3x_3 &\leq 18 \end{aligned}$$

Input

- **C** = [1, 2, 3]
- **A** = [[1, 1, 1], [2, 1, 3]]
- **b** = [12, 18]
- **Accuracy** = 0.7

Output

- Maximum value of
$$Z = 27$$
at
$$x_1 = 0, x_2 = 9, x_3 = 3$$

Problem 4

- **Maximization or Minimization?** Maximization
- **Objective Function:** Maximize
$$Z = 9x_1 + 10x_2 + 16x_3$$
- **Constraint Functions:**
$$\begin{aligned} 18x_1 + 15x_2 + 12x_3 &\leq 360 \\ 6x_1 + 4x_2 + 8x_3 &\leq 192 \\ 5x_1 + 3x_2 + 3x_3 &\leq 180 \end{aligned}$$

Input

- **C** = [9, 10, 16]
- **A** = [[18, 15, 12], [6, 4, 8], [5, 3, 3]]
- **b** = [360, 192, 180]
- **Accuracy** = 0.00001

Output

- Maximum value of
$$Z = 400$$
at
$$x_1 = 0, x_2 = 8, x_3 = 20$$

Problem 5

- **Maximization or Minimization?** Maximization
- **Objective Function:** Maximize
$$Z = 6x_1 + 2x_2 + 2.5x_3 + 4x_4$$
- **Constraint Functions:**
$$\begin{aligned} 5x_1 + x_2 + 2x_4 &\leq 1000 \\ 4x_1 + 2x_2 + 2x_3 + x_4 &\leq 600 \\ x_1 + 2x_3 + x_4 &\leq 150 \end{aligned}$$

Input

- **C** = [6, 2, 2.5, 4]
- **A** = [[5, 1, 0, 2], [4, 2, 2, 1], [1, 0, 2, 1]]
- **b** = [1000, 600, 150]
- **Accuracy** = 0.00001

Output

- Maximum value of
$$Z = 1050$$
at
$$x_1 = 0, x_2 = 225, x_3 = 0, x_4 = 150$$

Problem 6

- **Maximization or Minimization?** Not Applicable
- **Objective Function:** Maximize
$$Z = 4x_1 + 5x_2 + 4x_3$$
- **Constraint Functions:**
$$\begin{aligned} 2x_1 + 3x_2 - 6x_3 &\leq 240 \\ 4x_1 + 2x_2 - 4x_3 &\leq 200 \\ 4x_1 + 6x_2 - 8x_3 &\leq 160 \end{aligned}$$

Input

- **C** = [4, 5, 4]
- **A** = [[2, 3, -6], [4, 2, -4], [4, 6, -8]]
- **b** = [240, 200, 160]
- **Accuracy** = 0.001

Output

- The problem is unsolvable!

Problem 7

- **Maximization or Minimization?** Minimization
- **Objective Function:** Minimize
$$Z = -x_1 - x_2$$
- **Constraint Functions:**
$$\begin{aligned} x_1 + x_2 &\leq 1 \\ -x_1 - x_2 &\leq -3 \end{aligned}$$

Input

- **C** = [-1, -1]
- **A** = [[1, 1], [-1, -1]]
- **b** = [1, -3]
- **Accuracy** = 0.001

Output

- The method is not applicable!

Problem 8

- **Maximization or Minimization?** Maximization
- **Objective Function:** Maximize

$$Z = 2x_1 + x_2$$

  - **Constraint Functions:**

$$-x_1 + x_2 \geq 1$$

Input

- **C** = [2, 1]
- **A** = [[-1, 1]]
- **b** = [1]
- **Accuracy** = 0.001

Output

- The problem is unsolvable!

Setup and Run

```
cd assignment_1
python main.py
```

Code

simplex.py

```
from typing import List, Tuple
import numpy as np


class Simplex:
    """
    The Simplex class implements the Simplex method for solving linear programm
    """

    def __init__(
        self, C: List[float], A: List[float], b: List[float], accuracy: float
    ) -> None:
        """
        Initializes the Simplex method with the following inputs:
        C: Coefficients of the objective function.
        A: Coefficients of the inequality constraints.
        b: Right-hand side values of the inequality constraints.
        accuracy: Precision for detecting optimality (helps handle floating-point erro
        """
        self.C_coef = np.array(C)  # Objective function coefficients
        self.A_coef = np.array(A)  # Coefficients of the constraints
        self.b_coef = np.array(b)  # Right-hand side values
        self.accuracy = accuracy  # Desired accuracy
        self.table = None  # Simplex table
        self.optimised = False  # Indicates whether the solution is optimized
        self.solvable = True  # Indicates whether the problem is solvable

    def check_infeasibility(self) -> bool:
        """
        If any value in the right-hand side vector b is negative
        and the corresponding row in the matrix A has no positive coefficients,
        the problem is infeasible.
        """
        for i in range(len(self.b_coef)):
            if self.b_coef[i] < 0 and all(
                self.A_coef[i][j] <= 0 for j in range(len(self.A_coef[i]))
            ):
                return True
        return False

    def check_unboudedness(self, ratios: np.ndarray) -> bool:
        """
        If the objective function can grow indefinitely in the direction
        of the feasible region, then the problem is unbounded.
        """
        if np.all(np.isinf(ratios)):
            print("The problem is unsolvable!")
            self.solvable = False
            return True
        return False

    def fill_initial_table(self) -> None:
        """
    Initializes the Simplex table by combining the constraint matrix A,
    the identity matrix (for slack variables), and the right-hand side vector b.
    Also appends the objective function row with negative coefficients of C.
    """
    self.table = np.hstack(
        (
            self.A_coef,  # Coefficients of the constraints
            np.eye(self.A_coef.shape[0]),  # Identity matrix for slack variables
            np.reshape(self.b_coef, (-1, 1)),  # Right-hand side vector b
        )
    )
    # Objective function row (negative coefficients of C)
    func = np.hstack((-self.C_coef, np.zeros(self.A_coef.shape[0] + 1)))
    # Add the objective function row at the bottom
    self.table = np.vstack((self.table, func))

def make_iteration(self) -> None:
    """
    Performs one iteration of the Simplex algorithm:
    1. Finds the pivot column.
    2. Checks for unboundedness.
    3. Performs the pivot operation to transform the table.
    """

    if self.table is None:
        print("Table was not initialized!")
        return

    # Find the most negative value in the objective row
    pivot_column = np.argmin(self.table[-1, :-1])

    # Check if the solution is already optimal
    if self.table[-1, :-1][pivot_column] >= -self.accuracy:
        self.optimised = True
        return

    # Compute the ratios for the ratio test
    ratios = np.divide(
            self.table[:-1, -1],  # Right-hand side values (b)
            self.table[:-1, pivot_column],  # Pivot column values
            out=np.full_like(
                self.table[:-1, -1], np.inf
            ),  # Fill with inf where division is not valid
            where=self.table[:-1, pivot_column]
            > 0,  # Only consider positive entries in the pivot column
        )

        if self.check_unboudedness(ratios):
            return

        # Select the pivot row
        pivot_row = np.argmin(ratios)
        # Normalize the pivot row
        self.table[pivot_row] = (
            self.table[pivot_row] / self.table[pivot_row][pivot_column]
        )
        # Make all other elements in the pivot column zero
        for row in range(self.table.shape[0]):
            if row != pivot_row:
                self.table[row] = (
                    self.table[row]
                    - self.table[row][pivot_column] * self.table[pivot_row]
                )

    def get_solution(self) -> Tuple[List[float], float]:
        """
        Returns the decision variables and the optimized objective function value if t
        """

        # Check if the problem is infeasible
        if self.check_infeasibility():
            print("The method is not applicable!")
            self.solvable = False

        # Perform iterations while the solution is not optimized
        while (not self.optimised) and self.solvable:
            self.make_iteration()

        # If the problem is unsolvable, return empty results
        if not self.solvable:
            return [], None

        # Initialize solution array (size of decision variables + slack variables)
        solution = np.zeros(self.C_coef.shape[0] + self.A_coef.shape[0])

        for row in range(self.A_coef.shape[0]):
            # Find the column index in this row where the value is 1
            for col in range(self.C_coef.shape[0] + self.A_coef.shape[0]):
                # Check if this column is a basic variable
                if self.table[row, col] == 1 and np.sum(self.table[:, col]) == 1:
                    # This is a basic variable column
                    solution[col] = self.table[row, -1]
                    break  # Move to the next row

        # Extract decision variables from the solution
        decision_vars = solution[: self.C_coef.shape[0]]

        # Round to 10 decimal places
        decision_vars = [round(var, 10) for var in decision_vars]
        max_value = round(self.table[-1, -1], 10)

        return decision_vars, max_value
```

main.py

```
from src.simplex import Simplex

command = ""

while command.lower() != "end":
    print("What a nice day to solve optimization with simplex! (enter end to finish)")
    print("Enter function coeficients: ")
    command = input()
    if command.lower() == "end":
```

```
    break
try:
    function_row = list(map(float, command.split(" ")))
except Exception:
    print("Invalid function coefficients. Please, try again.")
    break
print("Enter number of constraints and then coeficients of constraints: ")
try:
    n = int(input())
    constraint_coef = []
    for _ in range(n):
        constraint_coef.append(list(map(float, input().split(" "))))
except Exception:
    print("Invalid constraints. Please, try again.")
    break
print("Enter right hand side: ")
try:
    rhs = list(map(float, input().split()))
except Exception:
    print("Invalid right-hand side coefficients. Please, try again.")
    break
print("Enter accuracy: ")
try:
    acc = float(input())
except Exception:
    print("Invalid accuracy value. Please, try again.")
    break
try:
    simplex = Simplex(function_row, constraint_coef, rhs, acc)
    simplex.fill_initial_table()
    answer, max_value = simplex.get_solution()
except Exception:
    print("You entered invalid problem. Please, try again.")
    break
print("Solution: ")
for i in range(len(answer)):
    print(f"x{i + 1} = {answer[i]}")
print("Max value: ")
print(max_value)
```