Assignment 2

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

Programming Language

- **Programming Language:** Python

Linear Programming Problem

Problem 1

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

Input

- **C** = [40, 30]
- **A** = [[1, 1], [2, 1]]
- **b** = [12, 16]
- **Accuracy** = 0.001
- **Alpha** = 0.5 and 0.9

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:**
$$x_1 + 4x_2 \leq 24$$
$$3x_1 + 1x_2 \leq 21$$
$$x_1 + x_2 \leq 9$$

Input

- **C** = [2, 5]
- **A** = [[1, 4], [3, 1], [1, 1]]
- **b** = [24, 21, 9]
- **Accuracy** = 0.001
- **Alpha** = 0.5 and 0.9

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** = 0.001
- **Alpha** = 0.5 and 0.9

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** = 0.001
- **Alpha** = 0.5 and 0.9

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** = 0.001
- **Alpha** = 0.5 and 0.9

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:**
$$2x_1 + 3x_2 - 6x_3 \leq 240$$
$$4x_1 + 2x_2 - 4x_3 \leq 200$$
$$4x_1 + 6x_2 - 8x_3 \leq 160$$

Input

- **C** = [4, 5, 4]
- **A** = [[2, 3, -6], [4, 2, -4], [4, 6, -8]]
- **b** = [240, 200, 160]
- **Accuracy** = 0.001
- **Alpha** = 0.5 and 0.9

Output

- The problem is unsolvable!

Problem 7

- **Maximization or Minimization?** Minimization
- **Objective Function:** Minimize
$$Z = -x_1 - x_2$$
- **Constraint Functions:**
$$x_1 + x_2 \leq 1$$
$$-x_1 - x_2 \leq -3$$

Input

- **C** = [-1, -1]
- **A** = [[1, 1], [-1, -1]]
- **b** = [1, -3]
- **Accuracy** = 0.001
- **Alpha** = 0.5 and 0.9

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
- **Alpha** = 0.5 and 0.9

Output

- The problem is unsolvable!

Setup and Run

```
cd assignment_2
python main.py
```

Code

interior_point.py

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


class InteriorPoint:
    def __init__(
        self,
        C: List[float],
        A: List[List[float]],
        b: List[float],
        accuracy: float,
        alpha: float = 0.5,
        starting_point: List[float] = None,
    ) -> None:
        """
        Initializes the problem with the following inputs:
        C: adjficients of the objective function (for maximization).
        A: adjficients of the inequality constraints.
        b: Right-hand side values of the inequality constraints.
        accuracy: Precision for detecting optimality (helps handle floating-point error
        """

        self.A_origin: List[List[float]] = np.array(A)  # adjficients of the constraints
```

```python
        self.b: List[float] = np.array(b)  # Right-hand side values
        self.accuracy: float = accuracy  # Desired accuracy
        self.starting_point: List[float] = starting_point
        self.n: int = len(C)
        self.m: int = len(b)

        self.alpha: float = alpha  # Speed of convergence(step)
        self.is_converged: bool = False  # Indicates whether the solution is optimized
        self.solvable: bool = True  # Indicates whether the problem is solvable

        # Adjasting slack
        self.C_adj = np.hstack(
            (np.array(C), np.zeros(self.A_origin.shape[0]))
        )  # Objective function adjficients
        self.A_adj = np.hstack((self.A_origin, np.eye(self.A_origin.shape[0])))
        if starting_point is None:
            self.starting_point = self.generate_random_point()

    def make_iteration(
        self,
        c: np.ndarray,
        a: np.ndarray,
        x0: np.ndarray,
        alpha: float = 0.5,
    ) -> List[float]:
        x = x0.copy()
        D = np.diag(x)
        AA = np.dot(a, D)
        cc = np.dot(D, c)
        F = np.dot(AA, np.transpose(AA))
        FI = np.linalg.inv(F)
        H = np.dot(np.transpose(AA), FI)
        P = np.subtract(np.eye(len(c)), np.dot(H, AA))
        cp = np.dot(P, cc)
        nu = np.absolute(np.min(cp))
        y = np.add(np.ones(len(c), float), (alpha / nu) * cp)
        yy = np.dot(D, y)
        x = yy
```

```python
        return x

    def generate_random_point(self) -> List[float]:
        """
        Generate an initial feasible point x0 such that A_adj @ x0 = b and x0 > 0.
        """
        n_vars: int = self.A_adj.shape[1]
        x0 = np.ones(n_vars)  # Start with all ones

        for _ in range(100):
            residual = self.A_adj @ x0 - self.b
            if np.linalg.norm(residual) < self.accuracy:
                break

            delta_x = np.linalg.lstsq(self.A_adj, self.b - self.A_adj @ x0, rcond=None)[
                0
            ]
            x0 += delta_x

            x0 = np.maximum(x0, 1e-6)

            # Detect infeasibility
            if np.linalg.norm(residual) > 1e6:
                self.solvable = False
                return None

        return x0

    def solve(
        self, alpha: float = 0.5, max_iterations: int = 1000, epsilon: float = 1e-6
    ) -> Tuple[List[float], float]:
        """
        Solve the given problem and return the decision variables and optimized obje

        :param alpha: Convergence speed (step size)
        :param max_iterations: Maximum number of iterations
        :param epsilon: Precision for detecting optimality (helps handle floating-point
```

```
    :return: A tuple containing the optimized decision variables and the objective
    """
    if self.solvable is False:
        print("The problem does not have solution!")
        return None, None

    x = np.array(self.starting_point)

    for _ in range(max_iterations):
        x_new = self.make_iteration(self.C_adj, self.A_adj, x, alpha)

        if np.allclose(x, x_new, rtol=epsilon, atol=epsilon):
            self.is_converged = True
            break

        # Detect unboundedness
        if np.any(np.abs(x_new) > 1e6):
            print("The problem does not have a solution!")
            self.solvable = False
            return None, None

        x = x_new

    # Check convergence status
    if not self.is_converged:
        print(
            "Warning: The algorithm did not converge within the maximum iterations
        )

    # Extract decision variables (excluding slack variables)
    decision_variables = np.round(x[: self.n], 5)

    # Calculate the optimized objective value
    optimized_objective = np.round(self.C_adj[: self.n].dot(decision_variables))

    return decision_variables, optimized_objective
```

main.py

```
from src.interior_point import InteriorPoint

command = ""

while command.lower() != "end":
    print("\nWhat a nice day to solve optimization with this constrained gradient de
    print("Enter 'end' to exit the program")
    print("Enter function coefficients: ")
    command: str = input()
    if command.lower() == "end":
        break
    try:
        function_row = list(map(float, command.split()))
    except ValueError:
        print("Invalid function coefficients. Please try again.")
        continue

    print("Enter number of constraints and then coefficients of constraints: ")
    try:
        n = int(input())
        constraint_coef = []
        for _ in range(n):
            constraint_coef.append(list(map(float, input().split())))
    except ValueError:
        print("Invalid constraints. Please try again.")
        continue

    print("Enter right hand side: ")
    try:
        rhs = list(map(float, input().split()))
    except ValueError:
        print("Invalid right-hand side coefficients. Please try again.")
        continue

    print("Enter accuracy: ")
    try:
        acc = float(input())
```

```python
    except ValueError:
        print("Invalid accuracy value. Please try again.")
        continue

    try:
        ip = InteriorPoint(function_row, constraint_coef, rhs, acc)

        # Solve the problem, getting the decision variables and optimized objective value
        answer, max_value = ip.solve()

        if answer is None:  # No solution found
            break

        # Print decision variables
        print("Decision variables:")
        for i in range(len(answer)):
            print(f"x{i + 1} = {answer[i]}")

        # Print the optimized objective value
        print(f"Optimized objective function's value: {max_value}")
    except Exception as e:
        print(f"An error occurred: {str(e)}")
        print("You entered an invalid problem. Please try again.")
```