Report 2

Team information.

- Team leader: Ruslan Belkov
- Team member 1: Eldar Mametov
- Team member 2: Mukhammadrizo Maribjonov
- Team member 3: Olga Puzhalina
- Team member 4: Kuklin Pavel
- Team member 5: Ivan Smirnov

Link to the product.

- The product is available: GitHub - dantetemplar/simplex
- Look src/interior_point.py

Programming language.

- Programming language: Python version 3.11+

Linear programming problem.

- Interior-Point algorithm
- Objective function:

$$z = -x_1 - x_2 + 0x_3 + 0x_4$$

- Constraint functions:

$$\begin{cases} 2x_1 + 4x_2 + x_3 + 0x_4 = 16 \\ x_1 + 3x_2 + 0x_3 - x_4 = 9 \end{cases}$$

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.
- $\alpha = 0.5$
- The first point

Output/Results

The output contains:

- Step on each iteretion

- The answer to the problem solution

Code

```
import numpy as np

from src.lpp import Problem, Solution


def solve_using_interior_point_method(
    problem: Problem,
    max_iterations: int = 1000,
    xtol: float = 1e-6,
    ftol: float = 1e-6,
    alpha: float = 0.5,
    first_trial_solution: np.ndarray | None = None,
) -> Solution:
    if not problem.is_augmented:
        raise ValueError("Problem is not augmented.")

    number_of_elements = problem.number_of_targets + problem.number_of_constraints

    I = np.eye(number_of_elements)
    ones = np.ones(number_of_elements, dtype=float)

    A = problem.A.copy()
    C = problem.C.copy()

    if first_trial_solution is None:
        prev_trial_solution = ones.copy()
    else:
        prev_trial_solution = first_trial_solution.copy()

    while True:
        # check if solution is found
        print(*[f"{x:.2f}" for x in prev_trial_solution])
        # assert that solution is positive
        if np.any(prev_trial_solution <= 0):
            raise ValueError("Solution is not positive.")

        if np.any((constraints := A @ prev_trial_solution) > problem.b):
            if np.allclose(constraints, problem.b, atol=ftol):
                break
            raise ValueError("Solution does not satisfy constraints.")

        # scaling step
        D = np.diag(prev_trial_solution)

        scaled_A = A @ D
        scaled_C = D @ C

        # projection step
        pseudoinverse_scaled_A = np.linalg.pinv(scaled_A)

        P = I - pseudoinverse_scaled_A @ scaled_A

        projected_C = P @ scaled_C

        # absolute value of the negative component of Cp
        nu = np.min(projected_C)
```

```
        if nu > 0:
            raise ValueError("No negative component of Cp.")

        trial_solution_in_current_space = np.ones(number_of_elements, dtype=float) + (-alp

        # scale back to original space
        next_trial_solution = D.dot(trial_solution_in_current_space)

        # by X
        if np.allclose(
            prev_trial_solution,
            next_trial_solution,
            atol=xtol,
        ):
            break
        # by objective function
        if np.allclose(
            f_current := problem.C.dot(next_trial_solution),
            f_prev := problem.C.dot(prev_trial_solution),
            atol=ftol,
        ):
            break

        prev_trial_solution = next_trial_solution
    # check if solution is found
    print(*[f"{x:.2f}" for x in prev_trial_solution])
    print(f"{f_current=}")


if __name__ == "__main__":
    C = [-1, -1, 0, 0]
    A = [[2, 4, 1, 0], [1, 3, 0, -1]]
    b = [16, 9]

    problem = Problem(C=C, A=A, b=b, number_of_targets=2, number_of_constraints=2)
    problem.is_augmented = True
    first = np.array([1 / 2, 7 / 2, 1, 2])

    solution = solve_using_interior_point_method(problem, first_trial_solution=first)

    print(solution)
```