Report of Programming Task 1 of the course "Introduction to Optimization" - Fall 2024

Nikita Zagainov, Ilyas Galiev, Arthur Babkin, Nikita Menshikov, Sergey Aitov

September 2024

1 Team Information

Team leader: Nikita Zagainov - 5
Team member 1: Ilyas Galiev - 5
Team member 2: Arthur Babkin - 5
Team member 3: Nikita Menshikov - 5
Team member 4: Sergey Aitov - 5

2 Link to the product

Project source code

3 Programming language

Python

4 Linear programming problem

We aim to maximize nutritious value of salad given constraints on cost of its ingredients, maximum fats concentration, and weight of each individual component

Ingredient	Tomato	Cucumber	Bell Pepper	Lettuce Leaf	Onion
Cost, rub/kg	130	100	155	85	50
Nutritious value, ckal/kg	200	160	260	150	400
Max weight in salad, kg	0.6	0.6	0.6	0.2	0.05
Fats, proportion	0.004	0.005	0.006	0.003	0.004

- Maximization

- Objective function & constraints:

$$\text{maximize } c^T x$$

subject to

$$Ax \leq b$$

where:

$$A = \begin{bmatrix} 130 & 100 & 155 & 85 & 50 \\ 0.004 & 0.005 & 0.006 & 0.003 & 0.004 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} 200 \\ 1 \\ 0.6 \\ 0.6 \\ 0.6 \\ 0.2 \\ 0.05 \end{bmatrix}$$

$$c = \begin{bmatrix} 200 \\ 160 \\ 260 \\ 150 \\ 400 \end{bmatrix}$$

5 Output & Results

We tested our implementation of simplex method by comparing its outputs with scipy implementation, and all tests show that outputs of both methods are the same on multiple tests, including original problem.
The method is applicable to our problem:

Problem is bounded: True

x : [0.2115, 0.6, 0.6, 0.2, 0.05]

f : 344.3

6 Code

```python
import numpy as np
from typing import Tuple


def simplex(
    A: np.ndarray, b: np.ndarray, c: np.ndarray, tol: float = 1e-6
) -> Tuple[bool, np.ndarray, float]:
    """
    Solves a linear programming problem using the simplex method.

    Args:
    A: A numpy array representing the coefficients of the constraints.
    b: A numpy array representing the right-hand side of the constraints.
    c: A numpy array representing the coefficients of the objective
        function.
    tol: A tolerance value for the simplex method.

    Returns:
    A tuple containing three elements:
    - A boolean value indicating whether the simplex method was
        successful.
    - A numpy array representing the solution to the linear programming
        problem.
    - The value of the objective function at the solution.
    """
    m, n = A.shape

    A_eq = np.hstack([A, np.eye(m)])
    c_eq = np.concatenate([c, np.zeros(m)])
    B = list(range(n, n + m))
    tableau = np.hstack([A_eq, b.reshape(-1, 1)])
    tableau = np.vstack([tableau, np.concatenate([c_eq, [0]])])

    while True:
        col = pivot_col(tableau, tol)
        if col == -1:
            break
        row = pivot_row(tableau, tol, col)
        if row == -1:
            return False, None, None

        tableau[row, :] /= tableau[row, col]
        for i in range(len(tableau)):
            if i != row:
                tableau[i, :] -= tableau[i, col] * tableau[row, :]

        B[row] = col
```

```
45
46      x = np.zeros(n + m)
47      x[B] = tableau[:-1, -1]
48
49      return True, x[:n], c @ x[:n]
50
51
52  def pivot_col(tableau: np.ndarray, tol: float) -> int:
53      last_row = tableau[-1, :-1]
54      if np.all(last_row >= -tol):
55          return -1
56      return np.argmin(last_row)
57
58
59  def pivot_row(tableau: np.ndarray, tol: float, col: int) -> int:
60      rhs = tableau[:-1, -1]
61      lhs = tableau[:-1, col]
62      ratios = np.full_like(rhs, np.inf)
63      valid = lhs > tol
64      ratios[valid] = rhs[valid] / lhs[valid]
65      if np.all(ratios == np.inf):
66          return -1
67      return np.argmin(ratios)
68
69
70  def main():
71      A = np.array(
72          [
73              [130, 100, 155, 85, 50],
74              [0.004, 0.005, 0.006, 0.003, 0.004],
75              [1, 0, 0, 0, 0],
76              [0, 1, 0, 0, 0],
77              [0, 0, 1, 0, 0],
78              [0, 0, 0, 1, 0],
79              [0, 0, 0, 0, 1],
80          ],
81          dtype=np.float32,
82      )
83      b = np.array([200, 0.01, 0.6, 0.6, 0.6, 0.2, 0.05], dtype=np.float32)
84      c = -np.array([200, 160, 260, 150, 400], dtype=np.float32)
85      state, x, f = simplex(A, b, c)
86      print("Solver state:", "solved" if state else "not solved")
87      print("Optimal solution:", x)
88      print("Optimal value:", f)
89
90
91  if __name__ == "__main__":
92      main()
```