Linear Programming Solver: Simplex Method and Variations

Project Overview

This project implements a comprehensive linear programming (LP) solver with support for multiple solution methods:

- **Standard Simplex Method**
- **BIG-M Method**
- **Two-Phase Method** (not fully implemented)
- **Preemptive Goal Programming**

The implementation handles various constraint types (≤, ≥, =) and supports unrestricted variables through variable substitution.

Code Explanation

1. Standard Simplex Method (`simplex.py`)

The core simplex algorithm implementation:

```
def standard_simplex(tableau, basic, variables):
    iteration = 0
    while True:
        print(f"\nIteration {iteration}:")
        print(variables)
        print(tableau)
        print(f"basic : {basic}")

        pivot_col = get_pivot_col(tableau)
        if pivot_col is None:
            status = "Optimal"
            break

        pivot_row = get_pivot_row(tableau, pivot_col)
        if pivot_row is None:
            status = "Unbounded"
            break

        basic[pivot_row] = variables[int(pivot_col)]
        tableau = pivot_tableau(tableau, pivot_row, pivot_col)
        iteration += 1
```

Key components:

- `get_pivot_col()`: Identifies the entering variable (most negative coefficient in objective row)
- `get_pivot_row()`: Performs ratio test to find leaving variable

- `pivot_tableau()`: Performs the pivot operation to update the tableau
- Iterates until optimal solution found or problem is unbounded

---

2. BIG-M Method (`big_m.py`)

Handles ≥ and = constraints by introducing artificial variables with large penalty `M`:

```
def big_m(tableau, variables, basic, problem_type):
    M = 100  # Large penalty value
    for i in range(len(variables)):
        if variables[i][0] == 'A':
            tableau[-1][i] = M  # Add penalty for artificial variables

    # Adjust objective function to account for artificial variables
    for i in range(len(basic)):
        if basic[i][0] == 'A':
            tableau[-1] = tableau[i] * -M + tableau[-1]

    # Solve using standard simplex
    tableau, basic, variables, status = standard_simplex(tableau, basic,
variables)

    # Check feasibility
    if status == "Optimal":
        for i in range(len(basic)):
            if basic[i][0] == 'A' and tableau[i][-1] != 0:
                return tableau, basic, variables, "in-feasible"
    return tableau, basic, variables, status
```

---

3. Input Processing (`input_receiver.py`)

Handles problem input and constructs initial tableau:

```
def input_processor(obj, constraints_coeff, constraints_type, unr_vars,
problem_type):
    # Convert inputs to numpy arrays
    obj = np.array(obj, dtype=float)
    constraints_coeff = np.array(constraints_coeff, dtype=float)

    # Handle unrestricted variables by splitting into x⁺ and x⁻
    for i in range(num_of_decision_variables):
        if (i + 1) in unr_vars:
            variables[col_index] = f"x{i + 1}_plus"
            variables[col_index + 1] = f"x{i + 1}_minus"
            tableau[:-1, col_index] = constraints_coeff[:, i]  # x⁺
            tableau[:-1, col_index + 1] = -constraints_coeff[:, i]  # x⁻
            col_index += 2
```

```
    # Add slack/surplus/artificial variables based on constraint type
    for i in range(num_of_constraints):
        if constraints_type[i] == '<=':
            variables[slack_index] = f"S{i + 1}"  # Slack
            tableau[i, slack_index] = 1
        elif constraints_type[i] == '>=':
            variables[surplus_index] = f"E{i + 1}"  # Surplus
            tableau[i, surplus_index] = -1
            variables[artificial_index] = f"A{i + 1}"  # Artificial
            tableau[i, artificial_index] = 1
```

---

4. Goal Programming (`goal_programming.py`)

Implements preemptive goal programming with prioritized objectives:

```
class PreemptiveGoalProgramming:
    def add_goal(self, name, coefficients, rhs, inequality_type, priority):
        # Add deviation variables for each goal
        pos_dev_name = f"d+_{name}"
        neg_dev_name = f"d-_{name}"
        goal.pos_dev_index = len(self.tableau_variables)
        self.tableau_variables.append(pos_dev_name)
        goal.neg_dev_index = len(self.tableau_variables)
        self.tableau_variables.append(neg_dev_name)

    def solve(self):
        for priority in sorted(priority_levels):
            current_goals = [goal for goal in self.goals if goal.priority
== priority]

            for goal in current_goals:
                deviation_vars_to_minimize = []
                if goal.inequality_type == InequalityType.LESS_THAN_EQUAL:
                    deviation_vars_to_minimize.append(goal.neg_dev_index)
                elif goal.inequality_type ==
InequalityType.GREATER_THAN_EQUAL:
                    deviation_vars_to_minimize.append(goal.pos_dev_index)

                for dev_var_idx in deviation_vars_to_minimize:
                    while True:
                        entering_var_idx =
self._find_entering_variable(tableau, goal_index)
                        if entering_var_idx is None:
                            break

                        # Pivot and optimize
                        tableau = pivot_tableau(tableau, leaving_row_idx,
entering_var_idx)
```

Sample Problem and Solution

The code includes a test case in `input_receiver.py`:

```
obj = [4, 6, 3]
constraints_coeff = [
    [1, 2, 1, 8],    # x1 + 2x2 + x3 ≤ 8
    [2, 1, 3, 12],    # 2x1 + x2 + 3x3 = 12
    [3, -1, 2, 7]     # 3x1 - x2 + 2x3 ≥ 7
]
constraints_type = ['<=', '=', '>=']
unr_vars = [2]        # x3 is unrestricted
problem_type = 'max'
```

This problem is solved using the **BIG-M method**, with `x3` being treated as an unrestricted variable (split into `x3_plus` and `x3_minus`).

Implementation Notes

- **Numerical Stability**: Uses a tolerance (`1e-7`) for floating-point comparisons
- **Tableau Tracking**: Prints intermediate tableaus for debugging
- **Variable Handling**:
    - Splits unrestricted variables into positive and negative parts
    - Automatically adds slack/surplus/artificial variables
- **Solution Extraction**: Reconstructs original variable values from final tableau

---

Future Improvements

- Complete **Two-Phase Method** implementation
- Add more robust **input validation**
- Implement **sensitivity analysis**
- Develop **graphical user interface**
- Add support for **integer programming**

The code provides a solid foundation for solving linear programming problems with various constraint types and special requirements like unrestricted variables and multiple objectives.