GatorPy: A Custom Implemented Linear Programming Solver

Andres Espinosa
Industrial and Systems Engineering
University of Florida
andresespinosa@ufl.edu

***Abstract*—**

I. Introduction

GatorPy is a custom Linear Programming solver implemented entirely in Python. The purpose of this project is twofold: First, to serve as an educational tool as an example of a simple and custom implementation of an LP solver. Second, to establish a foundation for other students to build upon and contribute to a collaborative open-source University of Florida custom optimization solver.

A. Linear Programming

B. Simplex Algorithm

C. Available Solvers

There are numerous available optimization solvers, both commercial and open-source. One particular source of inspiration for this project is CVXPY, an open-source convex optimization solver [1].

II. Implementation

A. GatorPy Syntax

The overarching goal with the optimization modeling syntax is to maintain a healthy balance between a pythonic syntax and standard optimization linear algebra notation. GatorPy relies heavily on the NumPy numerical processing package in Python. The general structure of a GatorPy problem involves the following steps:

1) Create `Parameter` objects for each parameter in the problem. Each `Parameter` object takes in a `np.array` as the value.
2) Create `Variable` objects for each variable in the problem. Each `Variable` takes in an integer as the shape of the vector. *Note: Each variable must be a vector, this is left as a potential next step in section IV-A.*
3) Create a `Problem` object representing the overall problem. The `Problem` object expects a Python `dict` object with the following key-value pairs:
   - Either `"minimize"` or `"maximize"` as a key with a GatorPy `Expression` as the value.
   - Either `"subject to"` or `"constraints"` as a key with a list of GatorPy `Constraint` objects as the value.

The simple syntax of GatorPy can be best communicated with an example. Consider the following optimization problem with two variables and three constraints.

$$\begin{array}{ll} \text{maximize} & \mathbf{c}^\top \mathbf{y} \\ \text{subject to} & \mathbf{A}\mathbf{y} \preceq \mathbf{b} \\ & \mathbf{y} \preceq \mathbf{1} \\ & \mathbf{y} \succeq \mathbf{0} \end{array}$$

where

$$\mathbf{c} = \begin{bmatrix} 1.2 \\ 0.5 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1.2 & 0.5 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

This above optimization problem can be expressed in GatorPy as:

```python
# Parameters
A = Parameter(np.array([[1,1],[1.2,0.5]]))
b = Parameter(np.array([1,1]))
c = Parameter(np.array([1.2,1]))

# Variables
y = Variable(2)

# Problem
problem = Problem({
    'maximize': c.T @ y,
    'subject to': [
        A @ y <= b,
        y <= 1,
        y >= 0
    ]
})

solution = problem.solve()
print(solution)
>>> (1.14, [0.71, 0.29], True)
```

Listing 1. Solving a Linear Program Symbolically

B. Simplex Implementation

C. Python Objects

D. LP Reductions

III. Results

A. Testing Framework

B. Testing Results

IV. Discussion

A. Future Work

B. Conclusion

References

[1] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *Journal of Machine Learning Research*, vol. 17, no. 83, pp. 1–5, 2016.