

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

The implementation of GatorPy can be sectioned into four parts: First, the algebraic modeling language syntax that was created by GatorPy, this implementation is available in section II-A. Second, the implementation of the Simplex algorithm that receives in a slack form LP and outputs the optimal solution (or an infeasibility/unbounded certificate if applicable). This simplex implementation is available in section II-B. Third, the Python objects that are created to facilitate the GatorPy modeling language. These objects and their overall purpose to the modeling language are available for viewing in section II-C. Finally, a large portion of the work in this project involved a series of reductions that turn any LP problem into an equivalent form that can be processed by the simplex algorithm.

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:

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

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:

```
1 # Parameters
2 A = Parameter(np.array([[1,1],[1.2,0.5]]))
3 b = Parameter(np.array([1,1]))
4 c = Parameter(np.array([1.2,1]))
5
6 # Variables
7 y = Variable(2)
8
9 # Problem
10 problem = Problem({
11     'maximize': c.T @ y,
12     'subject to': [
13         A @ y <= b,
14         y <= 1,
15         y >= 0
16     ]
17 })
```

```

18
19 solution = problem.solve()
20 print(solution)
21 >>> (1.14, [0.71, 0.29], True)

```

Listing 1. Solving a Linear Program Symbolically

B. Simplex Implementation

The implementation of the Simplex algorithm in GatorPy uses the two-phase simplex method, which ensures feasibility and optimality of the solution. This version of the simplex algorithm is divided into two main phases: an initial feasibility search phase (Phase 1), and an optimality algorithm (Phase 2). The following defines the overarching purpose and goal of each section in the implementation. The pseudo-code algorithm is available in Algorithm ??

1) *Phase 1: Finding a Feasible Solution:* The goal of Phase 1 is to find a basic feasible solution (BFS) for the given linear programming problem. If the problem is infeasible, Phase 1 will detect this and terminate. The process involves:

- 1) Augmenting the constraint matrix A with artificial variables to form an auxiliary problem.
- 2) Minimizing the sum of the artificial variables to determine feasibility.
- 3) Using Bland's rule to prevent cycling during pivot selection.
- 4) Removing artificial variables from the basis if a feasible solution is found.

2) *Phase 2: Optimizing the Objective Function:* Once a feasible solution is obtained, Phase 2 optimizes the objective function. The process involves:

- 1) Constructing the simplex tableau using the feasible basis obtained from Phase 1.
- 2) Iteratively selecting entering and leaving variables based on Bland's rule and the minimum ratio test.
- 3) Performing pivot operations to update the tableau and improve the objective value.
- 4) Terminating when no entering variable exists, indicating optimality.

3) *Key Functions in the Implementation:* The implementation relies on six key pieces of the full algorithm.

- `pivot`: Performs the pivot operation to update the simplex tableau. This algorithm is summarized in Algorithm 1
- `find_entering_variable`: Determines the entering variable based on the specified rule (e.g., Bland's rule). This algorithm is summarized in Algorithm 2
- `find_leaving_variable`: Identifies the leaving variable using the minimum ratio test. This algorithm is summarized in Algorithm 3
- `simplex_phase_1`: Implements Phase 1 of the simplex algorithm. This algorithm is summarized in Algorithm 4
- `simplex_phase_2`: Implements Phase 2 of the simplex algorithm. This algorithm is summarized in Algorithm 5

- `two_phase_simplex`: Combines Phase 1 and Phase 2 to solve the linear programming problem. This algorithm is summarized in Algorithm ??

Algorithm 1 Pivot Operation

Input: tableau $T \in \mathbb{R}^{m \times n}$, indices r and c

Output: tableau T

- 1: $T_{r,:} \leftarrow T_{r,:}/T_{r,c}$ ▷ normalize pivot row
 - 2: **for** each row $i \neq r$ **do**
 - 3: $T_{i,:} \leftarrow T_{i,:} - T_{i,c} \cdot T_{r,:}$ ▷ update rows
 - 4: **end for**
 - 5: **return** T ▷ return updated tableau
-

Algorithm 2 Find Entering Variable (Bland's Rule)

Input: tableau $T \in \mathbb{R}^{m \times n}$

Output: index j , or None

- 1: **for** $j = 1$ to $n - 1$ **do**
 - 2: **if** $T_{m,j} < -\epsilon$ **then**
 - 3: **return** j ▷ return entering index
 - 4: **end if**
 - 5: **end for**
 - 6: **return** none ▷ no variables found
-

Algorithm 3 Find Leaving Variable

Input: tableau $T \in \mathbb{R}^{m \times n}$, index c

Output: index r , or None

- 1: Initialize empty list R
 - 2: **for** $i = 1$ to $m - 1$ **do**
 - 3: **if** $T_{i,c} > \epsilon$ **then**
 - 4: Append $(T_{i,n}/T_{i,c}, i)$ to R
 - 5: **end if**
 - 6: **end for**
 - 7: **if** R is empty **then**
 - 8: **return** None ▷ no variables found
 - 9: **end if**
 - 10: $r \leftarrow \arg \min_R q$ ▷ update row index
 - 11: **return** r ▷ return leaving index
-

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.

Algorithm 4 Simplex Phase 1

Input: Matrix \mathbf{A} , vector \mathbf{b} **Output:** Matrix \mathbf{A} , vector \mathbf{b} , list B , or None

```
1:  $\mathbf{A}_{aux} := [\mathbf{A} \ \mathbf{I}]$ 
2:  $\mathbf{T} := \begin{bmatrix} \mathbf{A} & \mathbf{I} & \mathbf{b} \\ -\mathbf{1}^\top \mathbf{A} & -\mathbf{1}^\top & -\mathbf{1}^\top \mathbf{b} \end{bmatrix}$ 
3:  $B := [\dim(\mathbf{A})_2, \dots, \dim(\mathbf{A}_{aux})_2]$ 
4: while true do
5:    $c \leftarrow \text{find\_entering\_variable}(\mathbf{T})$ 
6:   if  $c$  is None then
7:     break
8:   end if
9:    $r \leftarrow \text{find\_leaving\_variable}(\mathbf{T}, c)$ 
10:  if none found then
11:    return None ▷ return infeasible
12:  end if
13:   $\mathbf{T} \leftarrow \text{pivot}(\mathbf{T}, c, r)$ 
14: end while
15: if  $T_{-1,-1} > 0$  then ▷ positive objective value
16:  return None ▷ return infeasible
17: end if
18:  $\mathbf{T} \leftarrow \text{pivot}(\mathbf{T}, c, r)$  ▷ remove aux vars
19:  $\mathbf{A}, \mathbf{b} \leftarrow \mathbf{T}$ 
20: return  $\mathbf{A}, \mathbf{b}, B$ 
```

Algorithm 5 Simplex Phase 2

Input: Matrix \mathbf{A} , vector \mathbf{b} , cost vector \mathbf{c} , basis B **Output:** Optimal value f^* , solution \mathbf{x}^* , or Unbounded

```
1:  $\mathbf{T} := \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ -\mathbf{c}^\top & 0 \end{bmatrix}$ 
2: Adjust cost row from  $B$ 
3: while true do
4:    $c \leftarrow \text{find\_entering\_variable}(\mathbf{T})$ 
5:   if  $c$  is None then
6:     break ▷ optimality reached
7:   end if
8:    $r \leftarrow \text{find\_leaving\_variable}(\mathbf{T}, c)$ 
9:   if  $r$  is None then
10:    return Unbounded
11:  end if
12:   $\mathbf{T} \leftarrow \text{pivot}(\mathbf{T}, r, c)$ 
13:  Update basis  $B$  with  $B[r] \leftarrow c$ 
14: end while
15: Extract solution  $\mathbf{x}^*$  from  $\mathbf{T}$  using basis  $B$ 
16:  $f^* \leftarrow -\mathbf{T}_{-1,-1}$  ▷ optimal value from cost row
17: return  $f^*, \mathbf{x}^*$ , feasible = true
```

Algorithm 6 Two-Phase Simplex Method

Input: matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, vector $\mathbf{b} \in \mathbb{R}^m$, cost vector $\mathbf{c} \in \mathbb{R}^n$ **Output:** optimal value f^* , solution $\mathbf{x}^* \in \mathbb{R}^n$, or Infeasible

```
1:  $(\mathbf{A}', \mathbf{b}', B) \leftarrow \text{simplex\_phase\_1}(\mathbf{A}, \mathbf{b})$ 
2: if result is None then
3:   return Infeasible
4: end if
5:  $(f^*, \mathbf{x}^*) \leftarrow \text{simplex\_phase\_2}(\mathbf{A}', \mathbf{b}', \mathbf{c}, B)$ 
6: return  $f^*, \mathbf{x}^*$ 
```
