Revised Simplex Method for Solving Linear Programming Problems

Amr Yasser Anwar[*] Omar Hazem Ahmed[†]

November 12, 2025[‡]

Abstract

This report studies the *Revised Simplex Method* for linear programming. We provide a compact but rigorous presentation: motivation, mathematical theory (including the classical tableau connection), algorithmic pseudocode (Phase I + Phase II), implementation choices and numerical safeguards, and a set of experiments illustrating regular and pathological cases. We compare our implementation to a modern Python solver (SciPy's `linprog`) and discuss numerical behaviour and complexity trade-offs.

Contents

[*]Dept. of CSAI, Zewail City, University of Science and Technology, Egypt.
[†]Dept. of CSAI, Zewail City, University of Science and Technology, Egypt.

1 Motivation

Linear programming (LP) is a cornerstone of optimization with broad applications in operations research, economics, engineering design, and machine learning. The classical Simplex method (Dantzig) remains a fundamental algorithm because it provides exact combinatorial certificates (optimal basis, unboundedness, or infeasibility) and is very efficient in practice. However, the classical tableau-based Simplex stores and manipulates an entire tableau, which is memory-inefficient and costly for large or sparse problems.

The *Revised Simplex Method* addresses these issues by maintaining only the basis matrix B (or its factorization) and a few auxiliary vectors. This reduces memory usage, allows efficient linear algebra (LU factorization and updates), and separates numeric linear algebra from combinatorial pivot selection.

Key reasons to implement the revised variant:

- Better memory scaling for large / sparse A.
- Numerically more stable if LU factorization with pivoting is used.
- Easier to integrate with high-performance sparse linear algebra.

2 Theory

2.1 Standard form and notation

We work with the standard (maximization) form:

$$\begin{aligned} \text{maximize} \quad & c^\mathsf{T} x \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0, \end{aligned}$$

where $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, typically $m < n$. A *basis* is an index set B of size m whose columns A_B are linearly independent. We denote the basic and nonbasic variables by x_B and x_N respectively. The basic feasible solution (BFS) corresponding to B is $x_B = B^{-1}b,\ x_N = 0$.

Dual variables and reduced costs. Let c_B, c_N be the cost subvectors. Define the simplex dual vector $y = B^{-\mathsf{T}} c_B$. The reduced costs for nonbasic indices are

$$\bar{c}_N = c_N - A_N^\mathsf{T} y.$$

For a maximization problem the current BFS is optimal iff $\bar{c}_N \leq 0$ (componentwise). A positive reduced cost indicates a profitable nonbasic variable to enter.

2.2 Tableau connection

The classical simplex tableau encodes $B^{-1}A$ and $B^{-1}b$; its rows correspond to basic variables. The revised simplex computes exactly the same quantities (reduced costs, directions, ratios) but avoids forming the full tableau $B^{-1}A$, instead computing quantities column-by-column via linear solves with B or via factorized solves (LU).

To illustrate, the tableau (partial) has rows:

basic	$B^{-1}A_N$	$B^{-1}b$
x_B	D	x_B

where column j of D is the solution $d = B^{-1}a_j$.

2.3 Certificates and special cases

- **Optimality certificate:** $\bar{c}_N \le 0$ and $x_B \ge 0$.
- **Unboundedness certificate:** for entering column q, if $B^{-1}a_q \le 0$ then the objective is unbounded along that direction.
- **Infeasibility:** no BFS exists — detected via Phase I (auxiliary LP) if Phase I optimal value > 0.
- **Degeneracy and cycling:** degenerate pivots (some $x_{B_i} = 0$) can cause cycling; use Bland's rule or lexicographic perturbation.

3 Algorithm

This section presents concise and practical pseudocode for both phases. Phase I constructs a feasible basis when none is available; Phase II optimizes the original objective starting from a feasible basis.

3.1 Phase I (auxiliary feasibility problem)

Given A, b with possibly no obvious BFS, form the auxiliary (minimization) LP by adding artificial variables $r \in \mathbb{R}^m$:

$$\min\ \mathbf{1}^\mathsf{T} r \quad \text{s.t.} \quad Ax + Ir = b,\ x \ge 0,\ r \ge 0.$$

Initialize basis B as the artificial columns (so $r = b$ initially if $b \ge 0$) and run the revised simplex on ($A_{\text{aux}} = [A \mid I],\ b,\ c_{\text{aux}}$). If the optimal Phase I objective $> \varepsilon$ (ε small tolerance) then the original LP is infeasible. Otherwise remove artificial variables from the basis (pivot them out) to obtain a feasible basis for the original LP.

Algorithm 1 Phase I (sketch)

```
Require: $A, b$
1: Build $A_{\text{aux}} = [A \mid I_m]$,   $c_{\text{aux}} = [0_n; 1_m]$
2: Initialize basis $B \leftarrow$ indices of artificial columns
3: Run RevisedSimplex($A_{\text{aux}}$, $b$, $c_{\text{aux}}$, $B$) until optimal
4: if optimal objective $> \varepsilon$ then
5:     return INFEASIBLE
6: else
7:     Remove artificial columns from basis (pivot out or drop redundant rows)
8:     return feasible basis $B_{\text{feasible}}$
9: end if
```

3.2 Phase II (optimization using Revised Simplex)

Assume a feasible basis B is available. We keep an LU factorization of B and use it for the linear solves needed.

Algorithm 2 Revised Simplex (Phase II) — LU-based

```
Require: $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, feasible basis $B$
Ensure: Optimal basis and solution or certificate of unboundedness
 1: Compute LU factorization of $B$ (with pivoting)
 2: Compute $x_B \leftarrow B^{-1}b$ (solve via LU)
 3: for iteration = 1 to max_iters do
 4:     Compute $y \leftarrow B^{-\top}c_B$ (solve LU-transpose)
 5:     For each $j \in N$: compute reduced cost $\bar{c}_j = c_j - a_j^\top y$
 6:     if all $\bar{c}_j \le$ tol for $j \in N$ then
 7:         return (optimal, $x$)
 8:     end if
 9:     Choose entering index $q \in N$ (pricing rule; e.g., largest $\bar{c}_j$ or Bland)
10:     Solve $Bd = a_q$ for direction $d$ (LU solve)
11:     if $d \le$ tol componentwise then
12:         return (unbounded)
13:     end if
14:     Compute step length $\theta^* = \min\{x_{B_i}/d_i : d_i > \text{tol}\}$ and leaving index $p$
15:     Pivot: replace $p$ by $q$ in basis indices
16:     Update LU factorization (rebuild or use efficient update)
17:     Update $x_B \leftarrow x_B - \theta^* d$ and set $x_q = \theta^*$
18: end for
19: return (max_iters_reached)
```

Pivot / pricing choices. For coursework, full pricing (compute all reduced costs) is simple and sufficient. For larger problems consider partial pricing or steepest-edge heuristics. Use Bland's rule to avoid cycling.

LU updates. Two pragmatic choices:

- Recompute LU after each pivot (robust, simple; cost $O(m^3)$ per rebuild).
- Use sparse LU updates / ETA updates and rebuild periodically (faster in practice for large m).

4 Implementation

This section records the concrete choices made in the implementation and practical tips to reproduce results.

4.1 Language, libraries and environment

Implementation used: Python 3.9+. Main libraries:

- `numpy` and `scipy.linalg` for linear algebra (`lu_factor/lu_solve`).
- `scipy.optimize.linprog` (method=`highs`) for baseline comparisons.
- Use a `requirements.txt` to pin versions; record the git commit hash.

4.2 Data structures and API

- Store A as `numpy.ndarray`. For larger work use SciPy sparse.
- Basis indices as integer arrays/lists (B and N).
- Public API: `solve(A,b,c, B0=None, tol=1e-9, max_iters=10000)` returning status, solution x, objective, and iteration log.

4.3 Numerical safeguards and tolerances

- Use tolerance $\text{tol} = 10^{-9}$ for zero checks (reduced costs, pivot denominators).
- Use LU with partial pivoting for numerical stability.
- Rebuild LU factorization if condition numbers grow or after a fixed number of pivots.
- Implement Bland's anti-cycling rule as a fallback.

5 Examples and Experiments

We run several small, illustrative experiments. Each example includes the standard-form matrices, the basis chosen, the algebraic steps performed by one or two revised-simplex iterations (reduced costs, duals, directions, ratio tests), and a short discussion. The goal is to make the textbook mechanics explicit so you can reproduce the iteration logs in `results/`.

5.1 Example 1 — Regular feasible LP (worked through)

Maximize $3x_1 + 2x_2$ subject to

$$\begin{aligned} x_1 + x_2 &\le 4, \\ x_1 + 3x_2 &\le 6, \\ x_1, x_2 &\ge 0. \end{aligned}$$

Introduce slack variables $s_1, s_2 \ge 0$ to convert to equalities. Using the variable ordering (x_1, x_2, s_1, s_2) the equality system is

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 3 & 0 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 4 \\ 6 \end{bmatrix}, \qquad c = \begin{bmatrix} 3 \\ 2 \\ 0 \\ 0 \end{bmatrix}.$$

We start with the obvious feasible basis $B = \{s_1, s_2\}$ (columns 3 and 4).

Initial basic solution. Since $B = I_2$,

$$x_B = B^{-1}b = b = \begin{bmatrix} 4 \\ 6 \end{bmatrix}, \qquad x_N = 0.$$

The basis costs are $c_B = [0, 0]^\mathsf{T}$.

Iteration 0 (compute reduced costs). Compute the simplex dual $y = B^{-\mathsf{T}} c_B = 0$. The reduced costs for the nonbasic columns (here x_1, x_2) are

$$\bar{c} = c_N - A_N^\mathsf{T} y = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

Because we are maximizing, any positive reduced cost indicates a profitable entering variable. Choose x_1 (largest reduced cost 3) to enter.

Direction and ratio test. Solve $Bd = a_{x_1}$; with $B = I$ we get

$$d = a_{x_1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Form the ratio test for components with $d_i > 0$:

$$\theta_i = x_{B_i}/d_i = \{4/1,\ 6/1\} = \{4, 6\}.$$

Minimum ratio is $\theta^* = 4$ leaving variable s_1 (row 1). We update

$$x_B \leftarrow x_B - 4d = \begin{bmatrix} 4 \\ 6 \end{bmatrix} - 4 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \end{bmatrix},$$

and set $x_1 = \theta^* = 4$. Objective increases by $4 \cdot 3 = 12$.

New basis and optimality check. New basis index set: $B = \{x_1, s_2\}$ (columns 1 and 4). The basis matrix is

$$B = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \qquad B^{-1} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}.$$

Compute the new dual

$$y = B^{-\mathsf{T}} c_B, \qquad c_B = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \Rightarrow B^{-\mathsf{T}} = (B^{-1})^{\mathsf{T}} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix},$$

so

$$y = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}.$$

Reduced cost for x_2 is

$$\bar{c}_{x_2} = c_{x_2} - a_{x_2}^{\mathsf{T}} y = 2 - \begin{bmatrix} 1 \\ 3 \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} 3 \\ 0 \end{bmatrix} = 2 - 3 = -1 \leq 0.$$

All reduced costs for nonbasics are nonpositive, so the current BFS is optimal.

Result. Optimal basic feasible solution:

$$x = (x_1, x_2, s_1, s_2) = (4, 0, 0, 2), \qquad \text{objective } = 12.$$

Iteration log (compact).

Iter	Entering	Leaving	Objective	Basic solution
0	–	–	0	$(s_1, s_2) = (4, 6)$
1	x_1	s_1	12	$(x_1, x_2) = (4, 0)$

This matches the simple handwritten simplex; the revised-simplex implementation should produce the same pivots and objective values (up to tolerances).

5.2 Example 2 — Phase I required (worked example)

We now present an LP that requires an explicit Phase I because the natural slack/surplus choice does not immediately provide a feasible basis.

Consider the maximization problem

$$\max\ 5x_1 + 4x_2 \quad \text{subject to} \quad \begin{aligned} x_1 + x_2 &\geq 4, \\ x_1 + 2x_2 &\geq 5, \\ x_1, x_2 &\geq 0. \end{aligned}$$

Convert $\geq$ constraints to equalities by subtracting surplus variables $s_1, s_2 \geq 0$ and adding artificial variables $a_1, a_2 \geq 0$:

$$x_1 + x_2 - s_1 + a_1 = 4,$$
$$x_1 + 2x_2 - s_2 + a_2 = 5.$$

Use the variable ordering $(x_1, x_2, s_1, s_2, a_1, a_2)$. The augmented matrix and right-hand side are

$$A_{\text{aug}} = \begin{bmatrix} 1 & 1 & -1 & 0 & 1 & 0 \\ 1 & 2 & 0 & -1 & 0 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 4 \\ 5 \end{bmatrix}.$$

Phase I formulation. The Phase I objective is min $a_1 + a_2$, i.e. the auxiliary cost vector is $c_{\text{aux}} = [0\ 0\ 0\ 0\ 1\ 1]^{\mathsf{T}}$. Initialize the basis to the artificial columns $B = \{a_1, a_2\}$ (indices 5 and 6). With this choice $B = I$ and the initial basic solution is

$$a = b = \begin{bmatrix} 4 \\ 5 \end{bmatrix}, \quad x_1 = x_2 = s_1 = s_2 = 0,$$

so the initial Phase I objective equals $4 + 5 = 9$.

Below we show two simplex pivots that drive the Phase I objective down to zero and remove the artificial variables from the basis. These algebraic steps are chosen to illustrate the mechanics of revised simplex in Phase I (not the only possible pivot sequence).

Phase I — iteration 0 (initial reduced costs). With $B = I$ and $c_B = [1, 1]^{\mathsf{T}}$ the dual equals $y = c_B$. Reduced costs for nonbasics (x_1, x_2, s_1, s_2) are

$$\bar{c}_{x_1} = 0 - [1; 1]^{\mathsf{T}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -2,$$

$$\bar{c}_{x_2} = 0 - [1; 2]^{\mathsf{T}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -3,$$

$$\bar{c}_{s_1} = 0 - [-1; 0]^{\mathsf{T}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = +1,$$

$$\bar{c}_{s_2} = 0 - [0; -1]^{\mathsf{T}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = +1.$$

Because Phase I is a minimization of the sum of artificials, a negative reduced cost indicates a direction that reduces the Phase I objective. We therefore pick x_2 (most negative reduced cost -3) to enter.

Solve $Bd = a_{x_2}$: with $B = I$ we have $d = a_{x_2} = [1\ 2]^{\mathsf{T}}$. Ratio test:

$$\theta = \min\{4/1,\ 5/2\} = \min\{4,\ 2.5\} = 2.5,$$

so the leaving variable is a_2 (row 2). After the pivot the new basis is $B = \{a_1, x_2\}$. Compute the updated basic solution by forming the new basis inverse (here small so we compute it explicitly):

$$B = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}, \qquad B^{-1} = \begin{bmatrix} 1 & -\frac{1}{2} \\ 0 & \frac{1}{2} \end{bmatrix}.$$

Thus the new basic vector is

$$x_B = B^{-1}b = \begin{bmatrix} 1 & -\frac{1}{2} \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 2.5 \end{bmatrix},$$

so the artificial $a_1 = 1.5$, $x_2 = 2.5$, and the Phase I objective has fallen from 9 to 1.5 (since a_2 left the basis and became zero).

Phase I — iteration 1 (remove remaining artificial). With the new basis $B = \{a_1, x_2\}$ the basis costs are $c_B = [1, 0]^\mathsf{T}$. The new dual is

$$y = B^{-\mathsf{T}} c_B = (B^{-1})^\mathsf{T} c_B = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -\frac{1}{2} \end{bmatrix}.$$

Reduced costs for the remaining nonbasics (including x_1) are:

$$\bar{c}_{x_1} = 0 - \begin{bmatrix} 1 \\ 1 \end{bmatrix}^\mathsf{T} y = 0 - (1 \cdot 1 + 1 \cdot (-\tfrac{1}{2})) = -\tfrac{1}{2},$$

$$\bar{c}_{s_1} = 0 - \begin{bmatrix} -1 \\ 0 \end{bmatrix}^\mathsf{T} y = 0 - (-1 \cdot 1 + 0 \cdot (-\tfrac{1}{2})) = +1,$$

$$\bar{c}_{a_2} = 1 - \begin{bmatrix} 0 \\ 1 \end{bmatrix}^\mathsf{T} y = 1 - (0 \cdot 1 + 1 \cdot (-\tfrac{1}{2})) = 1.5.$$

The most negative reduced cost is $\bar{c}_{x_1} = -\frac{1}{2}$, so choose x_1 to enter. Solve $Bd = a_{x_1}$ with $a_{x_1} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, and

$$d = B^{-1} a_{x_1} = \begin{bmatrix} 1 & -\frac{1}{2} \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}.$$

Ratio test over positive components of d:

$$\theta_i = x_{B_i}/d_i = \{1.5/(\tfrac{1}{2}),\ 2.5/(\tfrac{1}{2})\} = \{3,\ 5\},$$

so $\theta^* = 3$ and leaving index is the artificial a_1. After the pivot the basis becomes $B = \{x_1, x_2\}$ and the basic solution is

$$x_B = B^{-1}b = \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}.$$

All artificial variables are now nonbasic (zero). The Phase I objective is exactly zero, confirming feasibility of the original LP. We may now drop a_1, a_2 and start Phase II from the feasible basis $B = \{x_1, x_2\}$ with initial feasible solution $x = (3, 1)$.

Remarks on Phase I. This worked example demonstrates the usual pattern: start with artificial columns as the identity, use Phase I minimization to drive the artificial variables to zero (if feasible), then remove them and continue in Phase II from the resulting feasible basis. The same sequence of dual solves, reduced-cost computations, linear solves for directions, and ratio tests is used in both phases; only the objective vector changes.

5.3 Degeneracy, cycling and Bland's rule

Degeneracy occurs when a basic variable is exactly zero at a BFS (within the numerical tolerance). Degeneracy can cause the objective to remain unchanged across a pivot (a *degenerate pivot*), and—if poorly handled—can lead to cycling.

A small degenerate example. Consider

$$\max\ x_1 + x_2 \quad \text{subject to} \quad \begin{aligned} x_1 + x_2 &\le 1, \\ x_1 + 2x_2 &\le 1, \\ x_1, x_2 &\ge 0. \end{aligned}$$

Add slacks s_1, s_2 and use the ordering (x_1, x_2, s_1, s_2). With the initial basis $B = \{s_1, s_2\}$ we have $x_B = [1, 1]^\mathsf{T}$ and objective 0.

If x_1 is chosen to enter, the direction is $d = [1, 1]^\mathsf{T}$ and the ratio test gives ties $1/1 = 1$ and $1/1 = 1$. If the algorithm picks the first row to leave the basis we obtain the new basic solution $(x_1, s_2) = (1, 0)$ and $x_2, s_1 = (0, 0)$; note that the new BFS has one or more basic variables equal to zero (degeneracy). A subsequent pivot may not increase the objective (degenerate pivot). Repeated degenerate pivots with unwise tie-breaking are the mechanism that can produce cycling.

Bland's anti-cycling rule. Bland's rule prescribes a deterministic tie-breaking: always choose the entering (and leaving) variable with the smallest index among those that are eligible. Bland's rule guarantees finite termination (no cycling). For coursework or small experiments we recommend implementing Bland's rule as a fallback: use aggressive pricing (largest reduced cost) for speed, but if a pivot repeats a basis exactly (or if a maximum number of degenerate pivots is observed) switch to Bland's rule for safety.

5.4 Unboundedness test (simple analytic check)

A textbook unbounded example (explicit and easy to analyze) is

$$\max\ x_1 + x_2 \quad \text{subject to } x_1 - x_2 = 1, \quad x_1, x_2 \ge 0.$$

Write the equality in standard-form with variable ordering (x_1, x_2). The single-row constraint matrix is $A = [1\ \ -1]$ and $b = 1$. A feasible basis is $B = \{x_1\}$ with $x_1 = b = 1$ and $x_2 = 0$.

Compute the direction for the nonbasic column x_2:

$$d = B^{-1} a_{x_2} = 1 \cdot (-1) = -1 \le 0.$$

Because $d \le 0$ componentwise, increasing x_2 does not force any basic variable below zero; hence the objective can be increased arbitrarily along that direction and the LP is unbounded. The revised simplex detects this in one step (when the direction vector has no positive components the ratio test fails and the solver returns an unbounded certificate).

5.5 Suggestions for "hard" / stress tests

To exercise a revised-simplex implementation (numerical stability, LU updates, degeneracy handling), try the following types of problems:

- **Nearly singular bases:** construct A with columns that are nearly linearly dependent (e.g., add a small multiple ε of one column to another). This stresses LU factorization pivoting and condition-number checks.

- **Degenerate networks:** small network-flow instances where many basic variables are zero; these provoke many degenerate pivots.
- **Phase I-heavy cases:** chains of `>=` and equality constraints that force many artificials initially; validate that your Phase I objective reliably reaches (near) zero for feasible problems and that artificials are removed correctly.
- **Unbounded directions:** include a simple equality that leaves a free positive ray (as in the unbounded example above) to check the solver's unbounded certificate.

Remark on logging and reproducibility For each experiment save an iteration log (entering/leaving indices, objective, condition number of B, LU rebuild events) to `results/<problem>.log`. This makes it straightforward to fill the tables in Section **??** with measured numbers and to reproduce pathological behaviour.

6 Comparison with SciPy

To validate our implementation, we compared it against SciPy's HiGHS backend using the two worked examples from Section 5. Each experiment was executed with the script `run_experiments.py` (defaults: 6 repeats, 1 warmup → 5 measured trials). Timing was recorded using `time.perf_counter()`, and reported times are the median values with sample standard deviations over the measured trials. The revised solver provides simplex-specific diagnostics (simplex pivots, LU rebuilds, final basis indices), whereas HiGHS reports internal iteration counts (`res.nit`) that do not correspond to simplex pivots. The basis condition number, Cond(B), was computed using `numpy.linalg.cond` on the final basis matrix, when available.

Problem	Solver	Status	Obj	Residual ($\|Ax-b\|_\infty$)	Iter	LU rebuilds	Cond(B)
example1	revised	Optimal	12.0	0.0	1	2	2.6180
example1	SciPy (HiGHS)	Optimal	12.0	0.0	2[a]	—	—
example2_phaseI	revised	Optimal	0.0	0.0	2	3	6.8541
example2_phaseI	SciPy (HiGHS)	Optimal	0.0	0.0	2[a]	—	—

Problem	Solver	Time (median, s)	Time (std, s)	Trials
example1	revised	1.1578×10^{-4}	2.3582×10^{-5}	5
example1	SciPy (HiGHS)	8.4918×10^{-4}	8.9171×10^{-4}	5
example2_phaseI	revised	1.3740×10^{-4}	1.6711×10^{-5}	5
example2_phaseI	SciPy (HiGHS)	8.7754×10^{-4}	1.5965×10^{-4}	5

Table 1: Comparison summary. For the revised solver, 'Iter' indicates the number of simplex pivots; HiGHS reports internal iterations. '—' indicates metrics not available. Times are measured with `time.perf_counter()` and represent median ± sample standard deviation over measured trials.

[a] HiGHS internal iterations (`res.nit`), not simplex pivots.

Key Observations

- **Correctness:** Both solvers yield identical objective values and feasibility residuals for the test problems (example1 optimum = 12.0; example2_phaseI Phase I optimum = 0.0). The revised implementation is therefore numerically consistent with HiGHS on these examples.

- **Diagnostics:** The revised solver provides detailed basis-level information (simplex pivots, LU rebuilds, final basis, and condition number). HiGHS does not expose these quantities through `linprog`, hence the '—' entries in Table 1.
- **Performance on small problems:** On these tiny examples the Python implementation is substantially faster than SciPy/HiGHS: the revised solver's median time is $\approx 1.16 \times 10^{-4}$ s on `example1` (about 7.3× faster than HiGHS's $\approx 8.49 \times 10^{-4}$ s) and $\approx 1.37 \times 10^{-4}$ s on `example2_phaseI` (about 6.4× faster than HiGHS's $\approx 8.78 \times 10^{-4}$ s). This behaviour is expected for very small problems: our lightweight Python implementation avoids HiGHS startup/presolve overhead and per-call fixed costs dominate. For medium-to-large or highly sparse problems, HiGHS and other highly-optimized solvers typically outperform a simple Python solver.
- **Numerical behaviour:** Final basis condition numbers are small to moderate (≈ 2.62 and 6.85), and feasibility residuals are at machine precision ($< 10^{-12}$). Larger and numerically challenging instances should be tested to fully exercise LU pivoting and numerical safeguards (see Section 4).

Reproducibility All per-run JSON logs and the aggregated `results/summary.csv` used to build Table 1 are saved in the `results/` directory (e.g., `example1_revised_run1.json`, `example1_scipy_run.json`, `summary.csv`). Re-run `code/run_experiments.py` to refresh these numbers if you modify the solver or experiment settings.

7 Conclusion

This report has presented the Revised Simplex method from theory to implementation and demonstrated its behaviour on several illustrative examples. Concretely, we:

- Derived the dual and reduced-cost formulas used by the revised algorithm and explained the connection to the classical tableau;
- Described a practical LU-based implementation supporting Phase I feasibility recovery, pivoting strategies, and anti-cycling measures;
- Demonstrated correctness and basic performance on worked examples, and compared diagnostics and timings against SciPy/HiGHS (Table 1).

Our experiments confirm that the revised implementation reproduces the expected simplex pivots and returns the same optima and feasibility residuals as HiGHS on the small test problems, while providing richer basis-level diagnostics. The implementation remains primarily educational: for medium-to-large or highly sparse instances, performance and numerical robustness will benefit from specialised sparse factorizations, more sophisticated pricing (partial or steepest-edge), and systematic LU-rebuild policies. We therefore recommend these extensions as priorities for future work.

All scripts, per-run logs, and aggregated results are saved under `results/` to facilitate reproducibility. Releasing the implementation and test-suite (with pinned dependencies) will further aid validation and comparative studies.

References

[1] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali. *Linear Programming and Network Flows*. 4th ed. John Wiley & Sons, 2009. ISBN: 978-0470462720. URL: https://www.wiley.com.

[2] Robert E. Bixby. "Solving real-world linear programs: A decade and more of progress". In: *Operations Research* 50.1 (2002). Survey of practical LP solver developments and implementations, pp. 3–15. DOI: 10.1287/opre.50.1.3.17780. URL: https://doi.org/10.1287/opre.50.1.3.17780.

[3] R. G. Bland. "New finite pivoting rules for the simplex method". In: *Mathematics of Operations Research* 2.2 (1977). Introduces Bland's anti-cycling rule; guarantees finite termination, pp. 103–107. DOI: 10.1287/moor.2.2.103. URL: https://pubsonline.informs.org/doi/10.1287/moor.2.2.103.

[4] George B. Dantzig. *Linear Programming and Extensions*. Tech. rep. R-366. Classic monograph; foundational reference on linear programming. The RAND Corporation, 1963. URL: https://www.rand.org/pubs/reports/R366.html.

[5] ERGO-Code/HiGHS. *HiGHS: Linear optimization software (GitHub repository)*. https://github.com/ERGO-Code/HiGHS. Open-source code; cite when comparing implementations or reproducing experiments. 2017–.

[6] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. 4th ed. Baltimore, MD: Johns Hopkins University Press, 2013. ISBN: 978-1421407944. URL: http://www.cs.cornell.edu/cv/GVL4/.

[7] Julian Hall. *HiGHS: a high-performance linear optimizer*. Tech. rep. Overview of HiGHS design and features. Proceedings of the 8th International Conference on Continuous Optimization (ICCOPT) - invited talk, 2019. URL: https://webhomes.maths.ed.ac.uk/hall/ICCOPT19/ICCOPT19.pdf.

[8] Julian Hall and HiGHS contributors. *HiGHS: A high-performance linear optimization software*. https://highs.dev/. Project website and documentation; HiGHS contains high-performance revised-simplex and interior-point solvers. 2019–.

[9] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*. 4th ed. Cham: Springer, 2014. ISBN: 978-3319138415. URL: https://link.springer.com/book/10.1007/978-3-319-13415-8.

[10] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020). Use when referencing SciPy / `linprog` comparisons, pp. 261–272. DOI: 10.1038/s41592-019-0686-2. URL: https://doi.org/10.1038/s41592-019-0686-2.

License