Mathematical Programming
Simplex Implementation

Paolo Lammens and Luis Sierra Muntané

GM – FME – UPC

November 15, 2018

Contents

1 Introduction

In the field of linear programming, one of the most well-known optimization algorithms, if not the most famous, is the Simplex algorithm. Developed by the American mathematical scientist George Dantzig in the 1940s, the Simplex algorithm works by switching between a polyhedron's vertices, known as basic feasible solutions, via vectors known as basic feasible directions. By doing so algorithmically, due to the convexity of polyhedra, when certain conditions are met, the algorithm can be shown to converge. For this assignment, a naive implementation of the simplex method was coded using Python to solve linear programming problems.

2 Data

The problem sets provided to us for this assignment are the following.

- Paolo Lammens – 03494734: problem set no. 41
- Luis Sierra Muntané – 24491604: problem set no. 70

3 Implementation

3.1 Quick overview of the Simplex method

The simplex method uses an iterative approach to find the optimal solution of a linear programming problem. It consists of two phases, in each of which the same core algorithm is used, which will referred as the "simplex algorithm".

Let us consider a linear programming problem in standard form. The simplex algorithm takes in a constraint matrix A, a cost function c and an initial basic feasible solution x (along with its corresponding basis). Each iteration consists of four main steps:

1) First, an optimality test is run: if the current BFS is optimal, the algorithm terminates; otherwise, a nonbasic variable is selected to enter the basis. The optimality condition is based upon the reduced costs of every nonbasic variable x_q (for $q \in \mathcal{N}$), calculated as

$$r_q = c_q - c'_{\mathcal{B}} B^{-1} A_q \,. \tag{1}$$

   A negative reduced cost implies a possible improvement in the value of the cost function, but if all such reduced costs are non-negative, a minimum (and thus an optimum) has been attained and the algorithm terminates. Otherwise, one of the indices q is chosen such that $r_q < 0$, according to a certain rule (e.g. Bland's rule).

2) Next, the basic feasible direction associated to the current BFS (d) is computed.

$$d_{\mathcal{B}} = -B^{-1} A_q \,. \tag{2}$$

3) After that, the maximum step length θ^* is calculated. If d has no negative component, the problem is declared as unbounded and thus the algorithm terminates.

$$\theta^* = \min_{\{i=1,\dots,m \mid d_{\mathcal{B}(i)} < 0\}} \left(\frac{-x_{\mathcal{B}(i)}}{d_{\mathcal{B}(i)}} \right) \tag{3}$$

4) Finally, the values of the variables along with the cost function and basis are updated as follows:

$$\begin{gathered} x_{\mathcal{B}} := x_{\mathcal{B}} + \theta^* d_{\mathcal{B}}, \quad x_q := \theta^*, \quad z := z + \theta^* r_q, \\ \mathcal{B} := \mathcal{B} \setminus \{B(p)\} \cup \{q\}, \quad \mathcal{N} := \mathcal{N} \setminus \{q\} \cup \{B(p)\} \end{gathered} \tag{4}$$

As previously mentioned, the whole simplex method consists of two phases, the first of which serves to find an initial BFS for the original problem. Supposing the original problem is of the form

$$\begin{cases} \min\limits_{x \in \mathbb{R}^n} & c'x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{cases},$$

where $A \in \mathcal{M}_{m \times n}(\mathbb{R})$, this is obtained by applying the simplex algorithm to the so-called "phase I problem"

$$\begin{cases} \min\limits_{x \in \mathbb{R}^n,\ y \in \mathbb{R}^m} & \sum\limits_{i=1}^{m} y_i \\ \text{subject to} & Ax + \mathrm{Id}\, y = b \\ & x \geq 0 \end{cases}$$

with the idea that an initial BFS is trivially $y = b$ (with basic matrix is Id). If such a problem didn't have an optimal value for the cost function at $z* = 0$, then the original problem would be unfeasible.

3.2 The actual implementation

As previously mentioned, the simplex algorithm was brought to life by means of a Python 3.7 program. To do so, the steps described previously were implemented point by point, integrating Phase I into the code. Most features or functions declared to fulfill any of the previous steps are thoroughly commented in the code itself, as they are simply translations of the expressions illustrated in the previous subsection.

Pivoting rules An avid and discerning reader may have noticed that in the very first step of a simplex iteration, the choice of which nonbasic variable enters the basis is not clarified. This is because many choices are possible, and different choices of variables lead to different outcomes in the path followed by the algorithm. It is for this reason that in the implementation of the algorithm, more than one criterion was used to determine the entering variable. These pivoting rules were as follows.

On the one hand, Bland's rule is implemented as the standard rule to guarantee the convergence of the algorithm, since in the case of the existence of degenerate solutions it will cycle following the lexicographical order of the indices of the basic variables.

On the other hand, another implementation, which in our experience performs faster, is that of choosing the most negative reduced cost—equivalent, in an intuitive phrasing, to taking the path of steepest descent. This need not translate to the greatest decrease in the cost function since that also depends on the step length θ^*, but it generally did converge in fewer iterations for the problems provided. The main problem with this rule is the risk of endless cycling when treating

degenerate problems. Due to that, we've introduced a precautionary limit of 500 iterations, after which the function raises an exception. Nevertheless, we didn't encounter any problem in which that occurred.

Inverse update To make our algorithm more efficient, rather than computing the inverse matrix B^{-1} in every single iteration, the matrix is updated by using the previous matrix and changing the column corresponding with the variable that entered/exited the basis. That is achieved through a sequence of elementary row operations on the previous inverse matrix:

$$\beta'_i \coloneqq \beta_i - \frac{d_{\mathcal{B}(i)}}{d_{\mathcal{B}(p)}} \qquad \forall i \neq p$$
$$\beta'_p \coloneqq \frac{\beta_p}{-d_{\mathcal{B}(p)}},$$

where β_i and β'_i denote the ith row of the previous and next inverse matrix, respectively. In this way, we avoid calculating it from scratch, which is computationally expensive and so can unnecessarily lengthen the run time of the program.

Numerical error Furthermore, to avoid floating point arithmetic errors, an epsilon parameter was initialized at the beginning of the code so that very small numbers were rounded and did not cause errors to propagate and drastically affect the final results. More details are given below.

Design The overall design is a main "utilities" module, with all of the relevant functions, and an auxiliary script (see appendix A for further details). The following outline indicates the structure and design (i.e., main functions and global variables) of the main module, `simplex.py`.

- `simplex_core(A: matrix, c: np.array, x: np.array, basic: set, rule: int = 0) \ -> (int, np.array, set, float, np.array)`

  This function is the implementation of what we were referring to previously as the "simplex algorithm" (to distinguish it from the overall two-phase method). It is intended for internal use only (although in Python we're all consenting adults, so we need not make things "private").

  It basically contains some setup and then a loop that iterates through BFSs until either an optimum or a feasible ray is found. As we justify later in th text, we enforced a limit of 500 iterations. The return values are the exit code (`0` means an optimum has been attained, `1` means the problem is unbounded), the exiting BFS x^*, the basis set $\mathcal{B}$, the value of z^* (if an optimum has been attained), and a feasible ray d (if the problem is unbounded). At each iteration, it prints a status update.

  An overview of the main local varibles:

  - `m, n`: no. of rows and columns of A, respectively.
  - `B`: list of basic indices. It is stored as a `list` and not as a `set` (unlike the parameter `basic` and the local variable `N`) because we want to keep the order of the indices as is, so that when updating the inverse matrix we don't need to find out where the column corresponding to $\mathcal{B}(p)$ has to go—i.e., if we kept a strict order, we would first have to find the correct spot for $A_{\mathcal{B}(p)}$, according to the lexicographical order of the rest of basic indices, and then move the columns $A_{\mathcal{B}(i)}$ with $\mathcal{B}(i) > \mathcal{B}(p)$ one place to the right.

We update it at the end of each iteration by assigning `B[p] = q`.

- `N`: set of nonbasic indices. In this case, we do want to keep strict lexicographical order, since when using Bland's rule, we need to traverse the nonbasic indices in order. Using the built-in `set` type ensures that.

  It is updated as `N = N - {q} | {B[p]}`; note that it needs to be updated before `B`.
- `B_inv`: inverse of the basic matrix. It is calculated once in the initialization phase of `simplex_core` and then updated at each iteration as described above.
- `prices`: variable corresponding to the marginal costs (also known as shadow prices), calculated as $c_{\mathcal{B}}B^{-1}$, which was ubiquitously used throughout the code and so is stored in a separate variable for convenience and efficiency.
- `q`: index of entering nonbasic variable.
- `r_q`: reduced cost corresponding to the qth variable (r_q).
- `p`: index p within $\{1, \ldots, m\}$ such that $\mathcal{B}(p)$ is the exiting basic variable.
- `d`: basic feasible direction for current iteration.
- `theta`: step length for current iteration.
- `z`: value of the objective function.

- `simplex(A: matrix, b: np.array, c: np.array, rule: int = 0) -> (int, np.array, float, np.array)`

  This function is the "public" method of this module—i.e., the interface for external use. It takes the data of a problem (in standard form) and the pivoting rule to be used (`0` is Bland's, `1` is minimal reduced cost) as parameters. It does a bit of error checking and also enforces the condition that the constraint matrix has to be full-rank—if it is not, by means of a QR factorization, it removes the redundant rows.

  Then, it proceeds to the setup and execution of phase I—the latter by calling `simplex_core`. If it terminates with $z^* > 0$, it declares the problem as unfeasible and terminates. Otherwise, it continues to the execution of phase II—again, by calling `simplex_core`. The return values—although useless right now, since all results are printed to `stdout`—are the exit code (`0` means an optimum has been attained, `1` means the problem is unbounded), the resulting BFS x^*, the value of z^* (if an optimum has been attained), and a feasible ray d (if the problem is unbounded).

- `trunc(x: float) -> float`

  A function for rounding numbers to 0 when their absolute value is less than `epsilon`. Useful for avoiding numerical catastrophes such as failing to identify an unbounded problem because some of the components of d are along the lines of $-1.2\text{e}-18$ (when it ought to be 0)[1].

- `epsilon = 10 ** (-10)`

  A global, constant numerical threshold for truncating numbers to 0 (with `trunc`).

[1] We encountered this problem in the form of a singular basic matrix.

4 Results

The results obtained when solving the problems with the simplex implementation are presented in two groups corresponding to problem set 41 and problem set 70. Due to their length, the iterations are included in a separate text file (batch_solve.txt), but their overall breakdown is presented here.

4.1 Problem set 41

Here are the final results the algorithm produced for each problem. There are two such solutions for each problem, one corresponding to the algorithm using Bland's rule and a second one using the minimal reduced cost to determine the entering variable. The breakdown of these problems was: Problem 1 has an optimum $z^* \approx -279.29$, Problem 2 was an unbounded problem, Problem 3 had an optimum at $z^* \approx -1210.72$ and Problem 4 was found to be unfeasable in phase I.

- Problem 1:

```
Solving problem set 41, problem number 1, with Bland's rule...

-------------------------------------------------------------------------
| Found optimal solution at x =                                         |
| [  0.      2.507   0.      0.768 ...  49.137 136.908 392.496  17.49 ]. |
|                                                                       |
| Basic indices: {1, 3, 5, 6, 7, 13, 16, 17, 18, 19}                    |
| Nonbasic indices: {0, 2, 4, 8, 9, 10, 11, 12, 14, 15}                 |
|                                                                       |
| Optimal cost: -279.29040729186545.                                    |
-------------------------------------------------------------------------
19 iterations in phase I, 19 iterations in phase II (38 total).


Solving problem set 41, problem number 1, with minimal reduced cost rule...

-------------------------------------------------------------------------
| Found optimal solution at x =                                         |
| [  0.      2.507   0.      0.768 ...  49.137 136.908 392.496  17.49 ]. |
|                                                                       |
| Basic indices: {1, 3, 5, 6, 7, 13, 16, 17, 18, 19}                    |
| Nonbasic indices: {0, 2, 4, 8, 9, 10, 11, 12, 14, 15}                 |
|                                                                       |
| Optimal cost: -279.2904072759005.                                     |
-------------------------------------------------------------------------
12 iterations in phase I, 10 iterations in phase II (22 total).
```

- Problem 2:

```
Solving problem set 41, problem number 2, with Bland's rule...

--------------------------------------------------------------------------
| unbounded problem. Found feasible ray d =                              |
| [ 0.    0.    0.    0.    ...  4.667  8.667 14.333  1.   ]            |
| from x =                                                               |
| [ 0.  0.  0.  0.    ... 3370.039 5518.51  8575.667    0.   ].         |
--------------------------------------------------------------------------
17 iterations in phase I, 30 iterations in phase II (47 total).


Solving problem set 41, problem number 2, with minimal reduced cost rule...

--------------------------------------------------------------------------
| unbounded problem. Found feasible ray d =                              |
| [ 0.   0.   0.    0.    ...  4.667  8.667 14.333  1.   ]              |
| from x =                                                               |
| [ 0.  0.  0.  0.    ... 3370.039 5518.51  8575.667    0.   ].         |
--------------------------------------------------------------------------
11 iterations in phase I, 16 iterations in phase II (27 total).
```

- Problem 3:

```
Solving problem set 41, problem number 3, with Bland's rule...

--------------------------------------------------------------------------
| Found optimal solution at x =                                          |
| [7.325e-02 7.947e-01 2.326e+00 0. ... 0. 0. 0. 2.470e+02].             |
|                                                                        |
| Basic indices: {0, 1, 2, 8, 9, 10, 11, 12, 15, 19}                     |
| Nonbasic indices: {3, 4, 5, 6, 7, 13, 14, 16, 17, 18}                  |
|                                                                        |
| Optimal cost: -1210.7235588791593.                                     |
--------------------------------------------------------------------------
21 iterations in phase I, 10 iterations in phase II (31 total).
```

```
Solving problem set 41, problem number 3, with minimal reduced cost rule...

------------------------------------------------------------------------
| Found optimal solution at x =                                        |
| [7.325e-02 7.947e-01 2.326e+00 0. ... 0. 0. 0. 2.470e+02].           |
|                                                                      |
| Basic indices: {0, 1, 2, 8, 9, 10, 11, 12, 15, 19}                   |
| Nonbasic indices: {3, 4, 5, 6, 7, 13, 14, 16, 17, 18}                |
|                                                                      |
| Optimal cost: -1210.7235588158378.                                   |
------------------------------------------------------------------------
12 iterations in phase I, 7 iterations in phase II (19 total).
```

- Problem 4:

```
Solving problem set 41, problem number 4, with Bland's rule...

------------------------------------------
| Unfeasible problem (z_I = 824.25 > 0). |
------------------------------------------
14 iterations in phase I.


Solving problem set 41, problem number 4, with minimal reduced cost rule...

------------------------------------------
| Unfeasible problem (z_I = 824.25 > 0). |
------------------------------------------
10 iterations in phase I.
```

4.2 Problem set 70

Similarly, here are the final outcomes of the simplex iterations for problem set 70, in each case first for Bland's rule and then for the minimum reduced cost rule. The breakdown of the four problems in this set was: Problem 1 has an optimum $z^* \approx -471.53$, Problem 2 had an optimum at $z^* \approx -794.24$, Problem 3 was found to be unfeasable in phase I and finally Problem 4 is an unbounded problem.

- Problem 1:

```
Solving problem set 70, problem number 1, with Bland's rule...

------------------------------------------------------------------------
| Found optimal solution at x =                                        |
| [  4.239   0.      0.      2.625 ... 302.724   0.    100.705 479.73 ]. |
|                                                                      |
| Basic indices: {0, 3, 4, 5, 7, 9, 12, 16, 18, 19}                    |
```

```
| Nonbasic indices: {1, 2, 6, 8, 10, 11, 13, 14, 15, 17}                  |
|                                                                          |
| Optimal cost: -471.5280570075495.                                        |
----------------------------------------------------------------------------
19 iterations in phase I, 11 iterations in phase II (30 total).


Solving problem set 70, problem number 1, with minimal reduced cost rule...

----------------------------------------------------------------------------
| Found optimal solution at x =                                            |
| [  4.239   0.      0.      2.625 ... 302.724   0.    100.705 479.73 ]. |
|                                                                          |
| Basic indices: {0, 3, 4, 5, 7, 9, 12, 16, 18, 19}                       |
| Nonbasic indices: {1, 2, 6, 8, 10, 11, 13, 14, 15, 17}                  |
|                                                                          |
| Optimal cost: -471.528057003315.                                         |
----------------------------------------------------------------------------
15 iterations in phase I, 13 iterations in phase II (28 total).
```

- Problem 2:

```
Solving problem set 70, problem number 2, with Bland's rule...

----------------------------------------------------------------------------
| Found optimal solution at x =                                            |
| [  1.119   4.485   0.      0.    ... 251.853 268.039   0.     22.272]. |
|                                                                          |
| Basic indices: {0, 1, 7, 10, 11, 13, 15, 16, 17, 19}                    |
| Nonbasic indices: {2, 3, 4, 5, 6, 8, 9, 12, 14, 18}                     |
|                                                                          |
| Optimal cost: -794.2416133763497.                                        |
----------------------------------------------------------------------------


Solving problem set 70, problem number 2, with minimal reduced cost rule...

----------------------------------------------------------------------------
| Found optimal solution at x =                                            |
| [  1.119   4.485   0.      0.    ... 251.853 268.039   0.     22.272]. |
|                                                                          |
| Basic indices: {0, 1, 7, 10, 11, 13, 15, 16, 17, 19}                    |
| Nonbasic indices: {2, 3, 4, 5, 6, 8, 9, 12, 14, 18}                     |
|                                                                          |
| Optimal cost: -794.2416134520921.                                        |
----------------------------------------------------------------------------
12 iterations in phase I, 19 iterations in phase II (31 total).
```

- Problem 3:

```
Solving problem set 70, problem number 3, with Bland's rule...

-------------------------------------------
| Unfeasible problem (z_I = 553.699 > 0). |
-------------------------------------------
15 iterations in phase I.


Solving problem set 70, problem number 3, with minimal reduced cost rule...

-------------------------------------------
| Unfeasible problem (z_I = 553.699 > 0). |
-------------------------------------------
12 iterations in phase I.
```

- Problem 4:

```
Solving problem set 70, problem number 4, with Bland's rule...

-------------------------------------------------------------------------
| unbounded problem. Found feasible ray d =                             |
| [0.    0.    0.    0.    ... 3.111 9.222 9.111 1.   ]                |
| from x =                                                              |
| [0.   0.    0.    0.    ... 1901.889 6732.778 6899.889    0.   ].    |
-------------------------------------------------------------------------
14 iterations in phase I, 35 iterations in phase II (49 total).


Solving problem set 70, problem number 4, with minimal reduced cost rule...

-------------------------------------------------------------------
| unbounded problem. Found feasible ray d =                       |
| [ 0.  0.  1.  0. ... 21. 95. 15. 63.]                           |
| from x =                                                        |
| [    0.     0.   815.     0. ... 16416. 76448. 11508. 50509.]. |
-------------------------------------------------------------------
12 iterations in phase I, 23 iterations in phase II (35 total).
```

Appendix A Repository index

- simplex.py: Python module with all of the core implementation of the simplex method.
- solve.py: Python script for parsing and converting data from the database file (pm18_exercici_simplex_dades.txt) and calling the simplex function from the simplex.py module with those data.

  Usage:

```
$ python solve.py --help
usage: solve.py [-h] [--rule {bland,minrc}] num prob

positional arguments:
  num                   problem set number within 1-79
  prob                  problem number within 1-4

optional arguments:
  -h, --help            show this help message and exit
  --rule {bland,minrc}  pivoting rule for simplex algorithm
```

- batch_solve.bat: Windows bash file for automating solving our problems (problem sets 41 and 70, problems 1 through 4).
- batch_solve.txt: text file with the output of each of the solve.py calls' output when batch_solve.bat is run.
- pm18_exercici_simplex_dades.txt: given database file.
- simplex_implementation_report.pdf: this pdf file.