# The Simplex Method

Corey R. Randall

## I. Introduction

This document provides directions for how to use the accompanying Python files to solve a linear optimization model. All files were written from scratch by the author, Corey R. Randall, during the fall of 2018 as a project for the course MEGN 586 at Colorado School of Mines. The current iteration of the code utilizes the Simplex Method and has the ability to run a sensitivity analysis for the right-hand-side (RHS) constraint values.

## II. Background

The Simplex Method is an algorithm that exploits the form of linear models to arrive at an optimal solution. Using the fact that all linear programs have convex structures, the algorithm attempts to find an initial feasible solution and then proceeds to check adjacent extreme points so long as they improve the defined objective [1]. Convexity allows an optimal solution to be found since the structure dictates that local optima are also global optima.

## III. Installation

All Python files can be downloaded from GitHub and must be saved into the same folder. The syntax used is for Python 3 so an appropriate version should be downloaded and installed. The only external module that must also be installed is numpy which is used for array manipulation. A simple way to get this set up is to install Anaconda and use its graphical user interface (GUI) to add numpy to the base environment [2].

## IV. Format

For ease of use, the user only needs to make changes to the "LP_Runner.py" file within the section titled "User Inputs" in order to solve a linear optimization problem.

User inputs are set up in such a way that a non-experienced user can still utilize the program. This means that the formulation does not need to be converted into standard form prior to the use of the algorithm.

The objective can be defined as either 'minimize' or 'maximize' and must be a string. Regardless of this specification, within the "Standard_Form.py" file the problem is converted to a minimization for generalizing the algorithm to work for any program.

The "c_coeff" input represents the coefficients of each variable in the objective function. It is important to note that this vector must contain a value for each variable even if its coefficient is zero. The user should also note the order of the variables they are using for this list of coefficients since the constraints will need to have coefficients listed in the same order.

As many constraints as needed can be defined by adding to the constraint dictionary. The formulation must be done such that all variables have a non-negativity constraint; however, the user does not need to explicitly specify non-negativity in the constraint dictionary. Each constraint should have a unique index in increasing order, starting with '1' for the first. The constraint is then set equal to a list containing its name as a string, the set of coefficients (including zeros) for each variable in the same order as mentioned above, the inequality/equality used stated as a string, and the RHS value of the constraint. And example is shown below for a problem consisting of three variables:

$$constraint[1] = ['name',\ [1, 1, 0],\ '<=',\ 4]$$

The remaining user inputs are mostly optional and do not need to be changed to simply solve a problem. A sensitivity analysis for the RHS value of each constraint can be turned 'on' or 'off' since it is not required to solve the problem. Running this report requires an extra function call and can slow down the total run time for large problems. Since the Simplex Method does not specify a preference for variables entering the basis [3], the user can specify 'first' or 'last' for the incoming variable. Once the reduced costs are calculated, the incoming variable is chosen from the defined position considering only the negative entries. Changing this input can help to reduce the number of iterations that are performed while solving the problem. If multiple optima exist, this will not change the optimal objective function value, but will report different solutions for the variable set.

Due to the computational restrictions in numerical calculations in Python, a tolerance can be set by the user so that any positive or negative values within the specified magnitude will be treated as a zero. This means that the tolerance should in general be set to a very small number. The decimals input allows for a cleaner report to be generated. A full solution containing a higher numerical accuracy is still saved in the report; however, the initial print out is limited to the number of decimals defined by the user. It is important to note that the rounding of the solution only takes place once during the printing of the solution so that no loss of accuracy occurs during each iteration of the algorithm.

## V. Examples

A "Sample_Problems.py" file has been supplied which contains a few sample inputs as well as their outputs. These problems were chosen for their simplicity as well as to provide a range of tests for the algorithm. All outputs from these sample problems match solutions obtained from AMPL for the same formulations.

It is important to note that not all linear formulations have solutions. The Simplex Method will report when a formulation is either infeasible or unbounded. The code supplied here is set up to identify when these conditions exist and to print a statement to inform the user.

## VI. Outputs

Default outputs of the "LP_Runner.py" file will only display the stopping criterion, variable values, and objective function value. Additional information is still calculated and stored for the user to check if desired. After running the solution type: "conversion", "initial_solution", "solution", or "rhs_sensitivity_report" in the command line to display extra information about the problem and solution.

Each of these additional output prompts display information from the subfunction of each step of the Simplex Method. The "conversion" output displays the problem in standard form, "initial_solution" displays the state of the solution after Phase I, "solution" will give the same general output information while also including slack values, dual values, and more. Printing the "rhs_sensitivity_report" will only work if the user specified the sensitivity to be turned on as an input. When generated, it displays the minimum and maximum values that the RHS value of each constraint can take in order to keep the basis the same and not affect the optimal objective function value. Note that if there is a specific output that would like to be used from any of these reports, it can be specified inside brackets as a string. An example of this would be determining the slack values or dual values for each constraint using the statements below:

$$solution['slacks'] \text{ or } solution['duals']$$

For a list of all specific outputs and spellings, the bottom of each subfunction can be checked. All vector outputs associated with variables are in the same order that the user specified during the inputs of coefficients for the objective function and constraints. Likewise, all vector outputs associated with constraints (e.g. slack and dual values) are in the same order that the user specified using indices [1]...[m] for the set of m constraints.

## VII. Evaluation

In its current form, the files utilize a minimal number of loops and conditional statements in order to provide a timely solution. The main inefficiency that still exists is in Phase I. An initial feasible solution full of nothing but artificial variables starts Phase I even if another existing set of variables could have been chosen. This was done in order to get the code to function without taking extra time to code in a search algorithm that looks for existing identity matrix columns. If desired, an experienced coder may choose to edit the beginning of "Phase_I.py" to have a for loop that searches for these existing columns and only adds artificial variables as needed. Adding this structure has the possibility to dramatically improve the efficiency of the initial solution. In order to move to Phase II, an initial solution without any artificial variables must be found. Currently, the minimum number of iterations that are needed in order to move to that next phase is 'm', the number of constraints in the system. It is possible for systems to have enough slack variables such that no artificial variables would need to be added meaning that the minimum number of iterations that Phase I may need to run could be reduced down to one.

It is important to note that Python is not the most efficient language and that the code will take more time to run than professional software packages written in languages that take less time to compile and utilize less RAM such as C+ or Fortran; however, as an open-source development language, having this code in Python can help expand access to this algorithm to help further research in communities or groups that are not paying to license other software.

## VIII. Future Work

Multiple systems and processes can benefit from being optimized. While not all problems are fully linear, there are ways to linearize them that allow for a solution to be obtained. Current work is being done as part of a thesis project to explore transport properties in thin film nafion for use in fuel cell and battery modeling. Nafion is an electrolyte that allows for ion transport, but not for electron transport. As nafion becomes very thin, less than 60nm, the transport of ions and molecules behave much different than what has been observed in thicker bulk layers of the material [4].

An experiment for in-situ transport measurements via neutron reflectometry is in development that would allow for a 1-D profile of the material to be observed under various conditions. Neutron reflectometry is an optical technique that measures the intensity of a beam of neutrons reflecting off multi-layer samples. As the beam and detector rotate through small angles, less than 5, the reflected neutrons create constructive and destructive signals when reflecting off each layer in the sample. Using analytical solutions, the data can be fit and information about the thickness (sensitive to 1nm), composition, roughness, and more can be obtained.

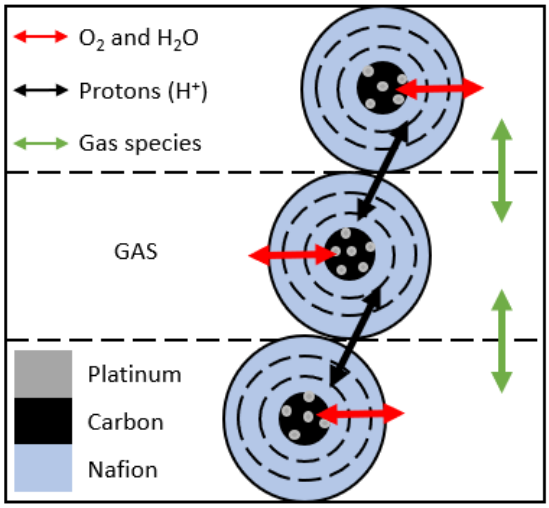


Fig. 1. Discretized cathode layer for pseudo-2D transport analysis in PEM cells using a particle/shell model approach.

For the novel experiments in development, this technique will assist in defining transport properties in much thinner films of nafion which are typically seen in polymer-electrolyte-membrane (PEM) fuel cells. A pseudo-2D model has been prepared to take in the mathematical relationships for these transport conditions as a function of relative humidity and thickness. A simplified graphic for this model is shown in Figure 1. Dashed lines in this figure show how the two dimensions can be discretized. Although they are not explicitly shown, reactions take place at the nafion and platinum surfaces and gas flows into the cathode from an external channel above. Reaction rates and gas compositions control only a portion of the transport that takes place in this system. Conductivity and diffusion properties are that affect transport also change because nafion swells to become thicker when in a hydrated environment. With water inside the membrane, transport should become faster due to water being a polar molecule to shield ions from each other; however, swelling also creates a longer distance through which ions and molecules must be transported in order to reach a reaction site. Ideally there exists some optimal thickness and humidity conditions at which transport is maximized. Depending on the ability to linearize the found relationships, this Simplex Model could be revisited in the future in order to find these conditions. For convenience, the PEM cell model has also been written in Python so that integration between the two can be done more easily. Prior to any optimization of the nafion thickness, the PEM cell model will need to be validated.

## REFERENCES

[1] Input citation
[2] Input citation
[3] Input citation
[4] Input citation