Tutorial 9: Simplex Explained

"Does an 'explanation' make it any less impressive?"

— Ludwig Wittgenstein

This tutorial explains how the simplex algorithm solves the LP model, i.e., how the simplex algorithm takes us from the mathematical model of the electricity market to the prices and quantities that appear on the display.

Demonstrating the simplex algorithm

Objective and Constraints

As discussed in Tutorial 1: Explaining Prices, the simplex algorithm solves a Linear Programming (LP) model, which is a mathematical problem specified in terms of:

- An *objective* equation
- *Constraint* equations that must be obeyed while achieving the objective.

The objective of the electricity market model is to maximize the objective value which is calculated as the difference between the value of the load that is supplied and the cost of the generation offers that must be cleared in order to supply the load.

The constraints ensure that the mathematical model behaves in the same was as the physical electricity network, e.g., there is a constraint that models how power flows along a transmission circuit.

Small electricity market model

The details of the simplex algorithm can be demonstrated by means of a small electricity market model consisting of a generator and a load connected via a bus, as shown in Figure 195. Build the model by tapping the Bus, Gen, Load buttons on the Build toolbar.

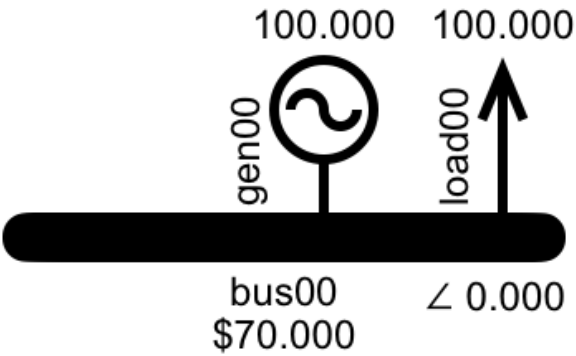

Figure 195: The small model

Constraints for the small model

The constraints that model the physical reality of the small model are listed in Table 12.

Table 12: Constraints in the small model

Component	Purpose of associated constraint
Bus00	Node balance constraint ensures that the quantity of electricity that enters the bus is equal to quantity that leaves
Gen00	Limit the cleared offer quantity to no more than the quantity that was offered
Load00	Limit the cleared bid quantity to no more than the quantity that was bid

The constraints listed in Table 12 are modelled by the equations shown in Equation 25, Equation 26 and Equation 27.

$$genOffer_{Cleared} - loadBid_{Cleared} = 0$$

Equation 25: Node balance equation

$$loadBid_{Cleared} \leq loadBid_{Max}$$

Equation 26: Max cleared bids

$$genOffer_{Cleared} \leq genOffer_{Max}$$

Equation 27: Max cleared offers

Objective function for the small model

The objective function for the small model is represented by Equation 28.

Maximize:

$$loadBid_{Cleared} \times loadBid_{Price}$$
$$-genOffer_{Cleared} \times genOffer_{Price}$$

Equation 28: Objective function

Simplex standard form

The simplex algorithm requires that the equations be expressed in a standard form:

- All equations expressed as $ax \leq b$ where $b \geq 0$, i.e., the RHS values are all ≥ 0
- Objective function expressed as a value to be maximized.

The electricity market model in standard form

Our objective function already has maximization as its requirement, which matches standard form.

All the constraints except for the node balance constraint meet the $ax \leq b$ requirement. The node balance constraint is an equality constraint so we replace the = with a $\geq$ and a $\leq$.

The $\leq$ is already in standard form. The $\geq$ equation is not so we multiply both sides by -1 to turn it into a $\leq$ equation. We can do this because the RHS is zero; hence after multiplying it by -1 the RHS is still zero and we still meet the $b \geq 0$ requirement.

After this adjustment the complete set of constraints for the model, expressed in the form required by the simplex algorithm, are as shown in Equation 29.

$$-loadBid_{Cleared} + genOffer_{Cleared} \leq 0$$

$$loadBid_{Cleared} - genOffer_{Cleared} \leq 0$$

$$loadBid_{Cleared} \leq loadBid_{Max}$$

$$genOffer_{Cleared} \leq genOffer_{Max}$$

Equation 29: Equation constraints for the small model, expressed in standard form

Viewing the constraints

To view the constraints for the small model, first you need to solve the model by tapping the Solve button, which takes you to the Solve Settings display, make sure all the options are selected OFF and then tap the Solve Now button.

When the solver has completed, tap the Results button, then tap the Constraints row, which takes you to the constraints display shown in Figure 196.

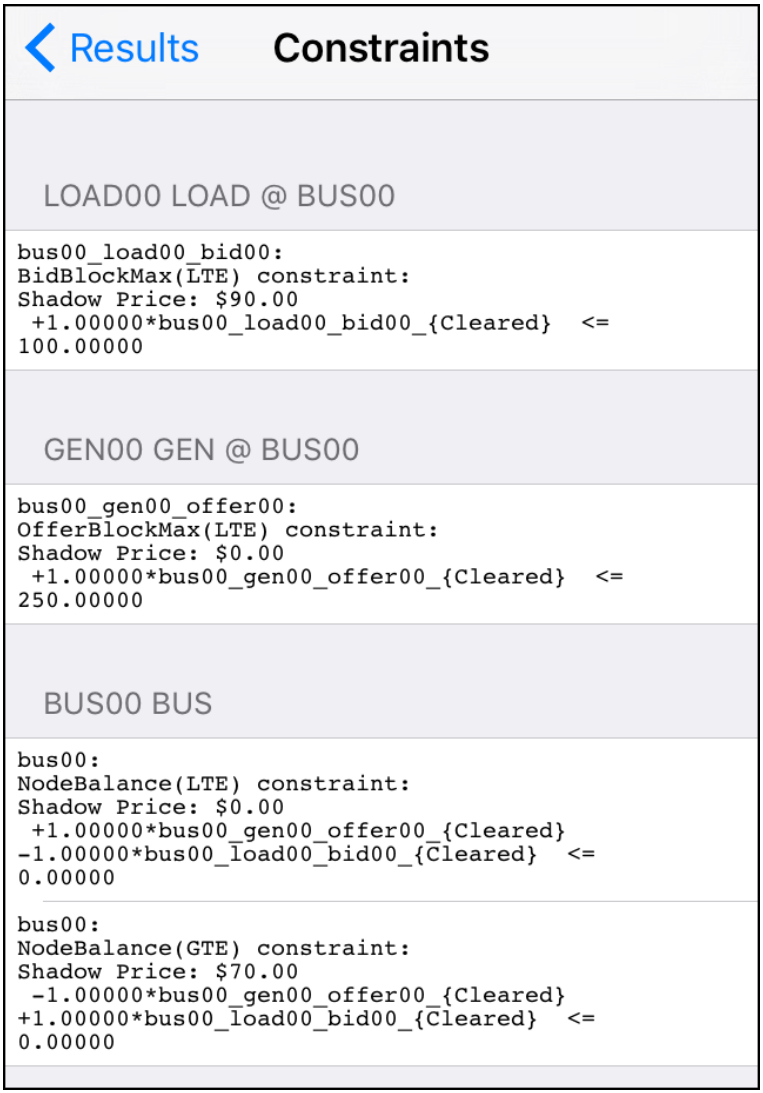

Figure 196: Constraint equations for the small model

Parameters and Variables

The elements of the constraint equations are *variables* and *parameters*.

Parameters are the fixed amounts:

$loadBid_{Max} = 100$

$genOffer_{Max} = 250$

Variables are the values that will be determined by the simplex algorithm:

$loadBid_{Cleared}$

$genOffer_{Cleared}$

Electricity market equations represented in tableau form

The simplex algorithm is easier to follow if the equations that form the mathematical model are represented as a tableau.

In the tableau the constraints are the rows, the variables are the columns, and the number at the intersection of the rows and columns represents the factor that the variable has in the constraint.

The following sections describe the actions that the simplex algorithm takes in order to maximize the objective value. You can track these actions via the tableaux.

The tableaux for the small model are shown in Figure 197 and Figure 198 (which are two halves of one spreadsheet, presented as two figures because it is too wide for this page).

###TABLEAU[0001]### col[1] will enter via row[2] and col[4] will leave			
Variables->	RHS	[1]bid00_{Cleared}	[2]offer00_{Cleared}
[1]_{NodeBalanceLTE} basic col[3]	0	-1	1
[2]_{NodeBalanceGTE} basic col[4]	0	1	-1
[3]bid00_{BidBlockMaxLTE} basic col[5]	100	1	0
[4]offer00_{OfferBlockMaxLTE} basic col[6]	250	0	1
[5]ObjectiveFn	0	-160	70
###TABLEAU[0002]### col[2] will enter via row[3] and col[5] will leave			
Variables->	RHS	[1]bid00_{Cleared}	[2]offer00_{Cleared}
[1]_{NodeBalanceLTE} basic col[3]	0	0	0
[2]_{NodeBalanceGTE} basic col[1]	0	1	-1
[3]bid00_{BidBlockMaxLTE} basic col[5]	100	0	1
[4]offer00_{OfferBlockMaxLTE} basic col[6]	250	0	1
[5]ObjectiveFn	0	0	-90
###TABLEAU[0003]### FINAL			
Variables->	RHS	[1]bid00_{Cleared}	[2]offer00_{Cleared}
[1]_{NodeBalanceLTE} basic col[3]	0	0	0
[2]_{NodeBalanceGTE} basic col[1]	100	1	0
[3]bid00_{BidBlockMaxLTE} basic col[2]	100	0	1
[4]offer00_{OfferBlockMaxLTE} basic col[6]	150	0	0
[5]ObjectiveFn	9000	0	0

Figure 197: Left side of the tableaux

[3]slack{NodeBalanceLTE}	[4]slack{NodeBalanceGTE}	[5]slack_bid00_{BidBlockMaxLTE}	[6]slack_offer00_{OfferBlockMaxLTE}
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0

[3]slack{NodeBalanceLTE}	[4]slack{NodeBalanceGTE}	[5]slack_bid00_{BidBlockMaxLTE}	[6]slack_offer00_{OfferBlockMaxLTE}
1	1	0	0
0	1	0	0
0	-1	1	0
0	0	0	1
0	160	0	0

[3]slack{NodeBalanceLTE}	[4]slack{NodeBalanceGTE}	[5]slack_bid00_{BidBlockMaxLTE}	[6]slack_offer00_{OfferBlockMaxLTE}
1	1	0	0
0	0	1	0
0	-1	1	0
0	1	-1	1
0	70	90	0

Figure 198: Right side of the tableaux

Tracking the simplex solve

The app allows you to track the actions of the simplex algorithm by saving the tableaux that the simplex algorithm uses. After the simplex algorithm has finished solving, the tableaux can be exported to a csv file, e.g., as shown in Figure 197 and Figure 198.

We are going to export the tableaux for the small model. Before we click the Solve Now button the app needs to know that we want to save the tableaus. By default the tableaux are not saved because for larger models the size of the tableaux grows quite quickly and writing them out will slow

down the time that it takes to solve the model (or if they are very large, the app will run out of memory).

Choosing to save tableaux

Tapping the Solve button takes you to the Solve Options display, which includes the "Save Tableaux" option shown in Figure 199. The "Some" option will save the first, second and last tableaux. Select the "All" option, then solve.

Figure 199: "Save Tableaux" on the Solve Settings display

Accessing the saved tableaux

To view the saved tableaux, after the solve has completed, you will need to export them as a csv file via email. Go to the Results display and then tap the "Import Export" button, indicated in Figure 200.

Figure 200: The Import Export button

The "Import Export" icon leads to the "Import Export" display shown in Figure 201. To export the tableaux, tap the "Email Results" button (the other options on this display are described in the Controls and Displays section).

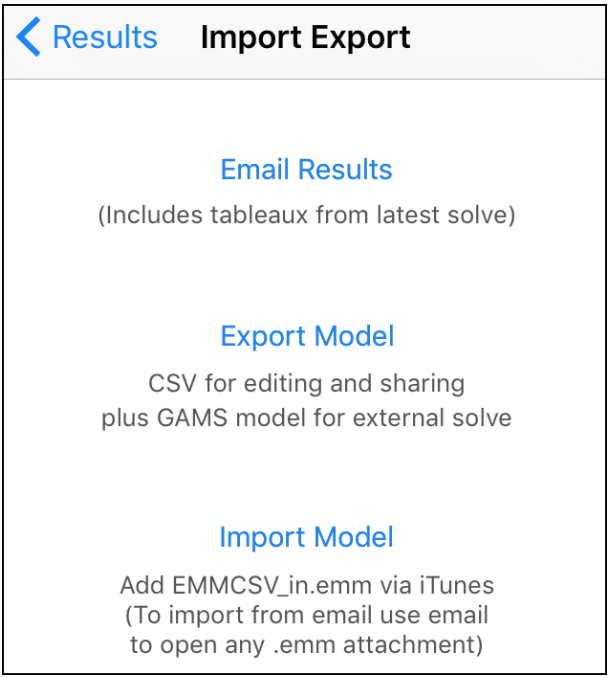

Figure 201: The Import Export display… use Email Results to export the tableaux

The "Email Results" email will contain several attachments; a screenshot of the model, a csv file containing the results of the latest solve and (assuming that there are saved tableaux) a csv file containing the saved tableaux from the solve. You can open the tableaux csv in a spreadsheet.

Layout of the tableau

The heading for each tableau describes the actions that the algorithm will apply to that tableau in order

to improve the objective value and thereby arrive at the next tableau, e.g., for Tableau00 the heading is "col[1] will enter via row[2] and col[4] will leave". What this means is explained below, as we track the progress of the solve.

Each variable has its own column in the tableau. Each constraint has its own row. The label for each row includes the name of the constraint, the row number, and the column number of the *basic variable* (explained below) for that constraint.

Initial Tableau

The simplex algorithm works to maximise the objective value while meeting the requirements of the constraints. To do this the simplex algorithm converts the equations, which are supplied to the algorithm in the standard form described above, into a set of *equality* constraints that represent a feasible solution. These are the equations represented by the tableau.

Slack variables

Each tableau represents a valid solution that meets the requirements of the constraints. To create the initial tableau the first thing that the simplex algorithm does is to convert the <= constraints to equality constraints (constraints that have an =

sign) by adding slack variables. Starting with the inequality constraint, e.g.,

$$-loadBid_{Cleared} + genOffer_{Cleared} \leq 0$$

…the simplex algorithm adds a slack variable, s_1. The slack variable can take any non-negative value and therefore allows the inequality ≤ in the original equation to be replaced with an equality, to create a new constraint:

$$-loadBid_{Cleared} + genOffer_{Cleared} + s_1 = 0$$

This meets the requirements of the original constraint, provided that the RHS is non-negative and all variables are ≥ 0. These requirements will be enforced by the rules of the algorithm.

Adding slack variables

Applying slack variables produces an equivalent set of equality constraints:

$$-loadBid_{Cleared} + genOffer_{Cleared} + s_1 = 0$$

$$loadBid_{Cleared} - genOffer_{Cleared} + s_2 = 0$$

$$loadBid_{Cleared} + s_3 = 100$$

$$genOffer_{Cleared} + s_4 = 250$$

Basic Feasible Solution (BFS)

From the equations shown above, the simplex algorithm creates the initial feasible solution by

setting all the slack variables to the RHS value and all other variables to zero:

$$s_1 = 0$$

$$s_2 = 0$$

$$\text{loadBid}_{\text{Cleared}} = 0$$

$$s_3 = 100$$

$$genOffer_{Cleared} = 0$$

$$s_4 = 250$$

This is represented by the initial tableau from the csv, i.e., Tableau01, reproduced in Table 13.

Table 13: Constraints in the initial tableau

Basic	$loadBid_{Clrd}$ [1]	$genOffer_{Clrd}$ [2]	s_1 [3]	s_2 [4]	s_3 [5]	s_4 [6]	RHS
[3]	-1	1	1				0
[4]	1	-1		1			0
[5]	1				1		100
[6]		1				1	250

Basic variables

In this first Basic Feasible Solution (BFS) all the variables have their *value* set to zero except for the slack variables. The solution is feasible because each slack variable only has a non-zero *factor* in one

of the constraints, and each constraint only has one variable that is not set to zero, i.e., the slack variable.

The variables that are set to zero are referred to as the non-basic variables. The remaining variables, currently the slack variables, constitute the basis of the solution and are referred to as the basic variables.

In this tableau and all that follow, the basic variables have a factor of zero in all constraints except one. This constraint is the constraint where they are the basic variable, and each constraint only has one basic variable. In the constraint where it is non-zero, the basic variable has a factor of one.

In the tableaux file each row records the column that contains the basic variable for that row's constraint, e.g., for row [1] the basic column is column [3].

Note that while the non-basic variables are assigned a *value* of zero, they can still have a non-zero *factor*. For example, the LoadBidCleared variable has non-zero factors in three of the equations, but it is non-basic so the LoadBidCleared variable has a *value* of zero by definition.

Values assigned by algorithm or equation

This is a re-statement of what we have just covered, but is repeated because it is important to note the distinction between the value assigned to a variable by the algorithm and the value enforced by the equations.

A non-basic variable has its value assigned by the algorithm. This value is zero. Regardless of the factor that this variable has in any equations, the value of a non-basic variable is zero.

A basic variable has its value enforced by the equations. The algorithm only allows a basic variable to have a non-zero factor in one equation, and this non-zero factor must be 1.0. Further, each equation is only allowed one basic variable. Therefore, the RHS of each equation sets the value of that equation's basic variable.

The objective function including slack variables

While this initial solution is feasible, the objective value as calculated by Equation 30 is zero because it was defined in terms of the original variables, which are currently all non-basic and therefore have values of zero.

Equation 30: The objective, incorporating the slack variables

Maximize:

$$150 \times loadBid_{Cleared}$$
$$-100 \times genOffer_{Cleared}$$
$$+ 0 \times s_1 + 0 \times S_2 + 0 \times s_3 + 0 \times S_4$$

Improving the objective value

Starting with the initial Basic Feasible Solution (BFS), the simplex algorithm moves to another BFS that improves the objective value. Improving the objective value is achieved by choosing the non-basic variable whose increase would most improve the objective value, and then making it a basic variable, which may allow it to be non-zero.

Keeping track of the objective value

To keep track of the objective value, the objective function is re-written as a constraint. Using z to represent the objective value, the objective function as a constraint is shown in Equation 31.

$$z - loadBid_{Cleared} \times loadBid_{Price}$$
$$+ genOffer_{Cleared} \times genOffer_{Price} = 0$$

Equation 31: Writing the objective function as a constraint with the objective value as variable z

Table 14 shows the initial tableau with the objective function added as a constraint.

Table 14: Constraints and objective as a tableau

z	$loadBid_{Clrd}$	$genOffer_{Clrd}$	s_1	s_2	s_3	s_4	RHS
	-1	1				1	0
	1	-1			1		0
	1			1			100
		1	1				250
1	-160	70					0

The progress of the algorithm will modify the tableau via a series of row operations. This will maintain the relationship between the variables as enforced by the original set of constraints. The row operations will also update the objective.

It turns out that we don't need to display the z column, because other than z there are no other values in this column so row operations won't change the z factor, and we won't be scaling the objective row, hence the factor of z will always be one.

The initial tableau can now be presented as shown in Table 15, which lines up with the csv export shown in Figure 197 and Figure 198.

Table 15: The initial tableau, incorporating the objective

Basic	$loadBid_{Clrd}$ [1]	$genOffer_{Clrd}$ [2]	s_1 [3]	s_2 [4]	s_3 [5]	s_4 [6]	RHS
[3]	-1	1	1				0
[4]	1	-1		1			0
[5]	1				1		100
[6]		1				1	250
	-160	70					0

In the objective function constraint the basic variables have *factors* of zero and the non-basic variables have *values* of zero… the only variable that is not set to zero and has a non-zero factor is the objective value z, which is therefore equal to the RHS. As we shall see, this will always be the case.

Currently the RHS of the objective row is zero and therefore the objective value is zero. The algorithm will now work to improve the objective value.

Reduced costs

In the objective function constraint the only variables that have non-zero *factors* are the non-basic variables.

The non-basic variables have *values* of zero. If one of these variables became non-zero then the objective value, z, (hidden off to the left with its factor always

one) would need to change, in order to keep the constraint balanced.

For example, if the $loadBid_{Clrd}$ variable, which has a factor of -150 in the objective function constraint, were to take on a value of 1.0 then the objective value would need to take on a value of 150 in order to keep the objective constraint valid as follows: 150 - 150 = 0.

Likewise, if the $genOffer_{Clrd}$ variable with a factor of 70 were to take on a value of 1.0 then the objective value would need to be -70.

Hence, the *factors* in the objective function constraint indicate which variable is the best to increase in order to improve the objective value. These factors are referred to as the *reduced costs*.

In this case only the $loadBid_{Clrd}$ variable can improve the objective value. Because it has a reduced cost of -150, each unit of increase by the $loadBid_{Clrd}$ variable improves the objective value by 150.

The entering variable and the leaving variable

To attempt to increase the objective value, the simplex algorithm chooses the variable with the most negative reduced cost, in this case $loadBid_{Clrd}$, and then increases the value of that

variable as much as possible. The only variables with non-zero reduced costs are the non-basic variables, so for the value of this variable to be increased, it needs to become a basic variable.

However, the tableau is feasible by virtue of the fact that there is only one basic variable for each constraint, so in order for the non-basic variable to become basic, i.e., to *enter the basis*, one of the basic variables will need to become non-basic, i.e., *leave the basis*.

The non-basic variable will become the basic variable for one of the constraints, i.e., the entering constraint… the tableau will be manipulated so that canonical form is maintained, i.e., so that the newly basic variable has a factor of 1.0 in the entering constraint and zero in all other constraints, including the objective function constraint. Also each constraint can only have on basic variable, hence in the entering constraint the variable that is currently basic will become non-basic.

Determining the entering constraint

The entering constraint is selected by finding the row that *allows* the factor of the entering variable to be set to 1.0 in that row and zero in all other rows.

What drives this selection is not what happens in the entering row, because there it is a simple division to obtain a factor of one, but rather what happens to the other rows in order to set the entering variable's factor to zero in those rows.

The first step to making the entering variable basic is to divide the entering row by the factor of the entering variable in that row, so that its factor becomes 1.0. Then the row operations will subtract a multiple of the entering row from the other rows so that the entering variable's factor becomes zero in those rows.

Row operations example on a dummy tableau

For this subsection we will use the dummy tableau in Table 16 to demonstrate how and why the algorithm determines the entering row. We are using a dummy tableau because the numbers in our actual model don't lend themselves to explaining the issue.

Selecting the entering row is decided by the requirement that variables must be non-negative (remember that this requirement arose when the original inequality constraints were replaced with slack variables and equality constraints).

The algorithm cannot select an entering row that would cause any of the basic variables to be assigned a negative value (the *non*-basic variables have values of zero regardless of the equations, so we don't have to worry about them).

Table 16: Dummy tableau to demonstrate entering and leaving variables

	Col#1	2	3	4	5	RHS
Row#1	-7	1				1
2	5		1			20
3	10			1		1
4	40		1		1	8
5	-150	70				0

In the dummy example the entering variable is in column one. When selecting the entering row we obviously can't use the row with the -7 factor because to make the factor 1 would result in a RHS of -1/7. Therefore the value of the basic variable would be -1/7, which cannot happen because all variables must be >= 0.

If row 2 was selected, then in order for the entering variable to take on a factor of 1.0 in that row, the row would be divide by 5.0, and the RHS becomes 4.0. Row 2 and 3 would then be as follows:

	Col#1	2	3	4	5	RHS
Row#2	1		1/5			4
Row#3	10			1		1

Row 2 is OK, but now to give the entering variable a factor of zero in row 3 we need to subtract 10 x row 2 from row 3. This will result in a RHS of -39 for row 3; hence whichever variable is basic in row 3 would become negative. So we can't do this.

Therefore, in order to prevent a negative RHS for any of the other rows, the entering row must be the row with the smallest positive ratio of RHS to entering variable factor… so that when any other row subtracts the entering row (in order to zero the entering variable in that row), the RHS of that row will remain positive.

In our dummy example, the row with the smallest positive ratio of RHS to entering factor is row 3, where the ration is 1/10. After the entering variable has entered via row 3, the tableau is shown in Table 17.

Table 17: Dummy tableau after pivot

	Col#1	2	3	4	5	RHS
Row#1	0	1		0.7		1.7
#2	0		1	-0.5		19.5

#3	1			0.1		0.1
#4	0		1	-4	1	4
Obj	0	70		15		15

The new tableau after the pivot

Back to our small model, the column with the smallest ratio of RHS to entering factor is row 2, with a ratio of zero. This is the same as the ratio for row 1; for consistency if more than one row has the same ratio then we always use the last row that we find, i.e., row 2 in this case.

After row operations have been performed to allow $loadBid_{Clrd}$ to become a basic variable, the tableau appears as shown in Table 19.

The entering variable is $loadBid_{Clrd}$. The leaving variable is the variable that was previously basic in the entering row. In this case the leaving variable is the slack variable *s2,* which becomes a non -basic variable.

Now that $loadBid_{Clrd}$ is a basic variable it is no longer set to a value of zero by the algorithm. However, the equation where $loadBid_{Clrd}$is basic has a RHS of zero, therefore the $loadBid_{Clrd}$ variable is assigned a value of zero by the equation.

We can also see from the RHS of the objective constraint that the objective value is still zero.

Table 18: Initial tableau (again)

Basic	$loadBid_{Clrd}$ [1]	$genOffer_{Clrd}$ [2]	s_1 [3]	s_2 [4]	s_3 [5]	s_4 [6]	RHS
[3]	-1	1	1				0
[4]	1	-1		1			0
[5]	1				1		100
[6]		1				1	250
	-160	70					0

Table 19: Second tableau

Basic	$loadBid_{Clrd}$ [1]	$genOffer_{Clrd}$ [2]	s_1 [3]	s_2 [4]	s_3 [5]	s_4 [6]	RHS
[3]			1	1			0
[1]	1	-1		1			0
[5]		1		-1	1		100
[6]		1				1	250
		-90		160			0

Updating the objective value

Bringing the entering variable into the basis included a row operation to set its reduced cost to

zero. The reduced cost of zero indicates that the variable has been increased as much as possible and cannot improve the objective value any further. The row update that set the reduced cost to zero also had the impact of updating the objective value.

The reduced cost of the entering variable was -150, hence to zero the reduced cost, the row operation added 150 times the entering row to the objective row. This has the effect of adding 150 times the RHS of the entering row to the RHS of the objective row. Because the RHS of the entering row *is* the value of the entering variable this has the effect of adding 150 times the value of the entering variable to the objective value… which was the intent of the iteration in the first place.

Adjusting the reduced costs

The first step was to scale the entering row so that the entering variable has a factor of one. This also scaled all other factors in that row.

Apart from the entering variable, the only other variables with non-zero factors in the entering row are non-basic variables. After scaling, the entering variable has a factor of one, and the factor each non-basic variables now indicates, if they were to become non-zero, how much a one unit change in

their value would force the entering variable to change.

After we scaled the entering row to make the entering factor one, we added that row, multiplied by the reduced cost of the entering variable, to the objective function, in order to zero the reduced cost of the entering variable there. What this also did was… for all the non-basic variables in the entering row take the rate at which they would allow the entering variable to change, multiply this rate by the reduced cost of the entering variable, and add the result to their original reduced costs.

Thus the reduced costs of the non-basic variables have been updated to reflect their value in terms of of how much freedom they can provide the entering variable, i.e., by the ratio of their factor in the entering row to that of the entering variable, multiplied by the value of the entering variable.

The non-basic variables in the entering row only have the potential to allow the entering variable to *increase* if their factor in the constraint is *negative*. If their factor is negative then the update of the objective function row results in an increase in the negativity of their reduced cost. If this increased negativity has caused their reduced cost to become negative then it means that any cost that they

previously presented has been overcome and they now represent a potential benefit.

Looking at tableau 2 in Figure 202 and Figure 203 (the tableaux csv again) we can see that this is what has happened; the potential of the cleared offer to allow the cleared bid to increase in value has overcome the cost of the offer… the reduced cost of the cleared offer variable was positive and now it is negative.

###TABLEAU[0001]### col[1] will enter via row[2] and col[4] will leave			
Variables->	RHS	[1]bid00_{Cleared}	[2]offer00_{Cleared}
[1]_{NodeBalanceLTE} basic col[3]	0	-1	1
[2]_{NodeBalanceGTE} basic col[4]	0	1	-1
[3]bid00_{BidBlockMaxLTE} basic col[5]	100	1	0
[4]offer00_{OfferBlockMaxLTE} basic col[6]	250	0	1
[5]ObjectiveFn	0	-160	70
###TABLEAU[0002]### col[2] will enter via row[3] and col[5] will leave			
Variables->	RHS	[1]bid00_{Cleared}	[2]offer00_{Cleared}
[1]_{NodeBalanceLTE} basic col[3]	0	0	0
[2]_{NodeBalanceGTE} basic col[1]	0	1	-1
[3]bid00_{BidBlockMaxLTE} basic col[5]	100	0	1
[4]offer00_{OfferBlockMaxLTE} basic col[6]	250	0	1
[5]ObjectiveFn	0	0	-90
###TABLEAU[0003]### FINAL			
Variables->	RHS	[1]bid00_{Cleared}	[2]offer00_{Cleared}
[1]_{NodeBalanceLTE} basic col[3]	0	0	0
[2]_{NodeBalanceGTE} basic col[1]	100	1	0
[3]bid00_{BidBlockMaxLTE} basic col[2]	100	0	1
[4]offer00_{OfferBlockMaxLTE} basic col[6]	150	0	0
[5]ObjectiveFn	9000	0	0

Figure 202: Left side of the tableaux csv

[3]slack{NodeBalanceLTE}	[4]slack{NodeBalanceGTE}	[5]slack_bid00_{BidBlockMaxLTE}	[6]slack_offer00_{OfferBlockMaxLTE}
1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1
0	0	0	0
[3]slack{NodeBalanceLTE}	[4]slack{NodeBalanceGTE}	[5]slack_bid00_{BidBlockMaxLTE}	[6]slack_offer00_{OfferBlockMaxLTE}
1	1	0	0
0	1	0	0
0	-1	1	0
0	0	0	1
0	160	0	0
[3]slack{NodeBalanceLTE}	[4]slack{NodeBalanceGTE}	[5]slack_bid00_{BidBlockMaxLTE}	[6]slack_offer00_{OfferBlockMaxLTE}
1	1	0	0
0	0	1	0
0	-1	1	0
0	1	-1	1
0	70	90	0

Figure 203: Right side of the tableaux csv

Why the objective value is still zero

Now that the cleared bid is a basic variable it can take a non-zero value but because the RHS of its basic constraint is zero the cleared bid is constrained to zero, and therefore the objective value is still zero.

The next entering variable

Now that the cleared bid variable is basic it is constrained by the RHS of its basic. The cleared bid can only increase if enabled to do so by another variable in this constraint. The other variable would need to have a negative factor in the constraint…

increasing the value of this other variable would allow the cleared bid variable to increase.

We know the contents of the node balance constraint in this model, i.e., cleared bids out of the bus must be matched by cleared offers in, so it is no surprise that the variable that will allow the cleared bid variable to increase is the cleared offer variable.

The reduced cost for cleared offers is now -$90, i.e., the difference between the $70 cost of the cleared offer (i.e., its original reduced cost) and the $160/MW benefit resulting from the increase in cleared bids that the cleared offer would enable.

This difference was calculated when the row operations zeroed the reduced cost of the cleared bid, passing the per MW benefit of the cleared bid along to the cleared offer, offsetting the existing cost of the offer.

The cleared offer variable is the variable with the most negative reduced cost; hence it will be the next entering variable. The next (and, in this small model, final) step is for the cleared offer variable to enter the basis. This will follow the same procedure as above; find the entering row (the row which will allow the necessary row operations without making any basic variables negative) and then perform row operations in order to allow the entering variable to

become basic (by meeting the requirement that it has a factor of 1 in the entering row and zero in all others).

A closer look at the entering row

The entering variable becomes the basic variable in the row with the lowest ratio of RHS to factor, because choosing any other row would break constraints by forcing their RHS negative.

Looking at the numbers we can relate this to the reality of our model. One of the constraints in tableau02 is the offer max constraint, which prevents the offer from clearing more than its offered quantity. The other constraint originally limited the cleared bid to be no more than the bid max… but after the pivot it now acts to limit the cleared *offer* to be no more than the bid max.

The row operations of the pivot effectively moved the relationship between the bids and offers from the node balance constraint (the entering row) to the bid max constraint.

We can see that this "new" constraint, which limits the cleared offer to be no more that the bid max, is the over-riding constraint on the cleared offer. If we allowed the cleared offer to enter via the other constraint, i.e., the offer max constraint, then the

cleared offer quantity would be more than the cleared bid quantity, breaking the node balance.

The algorithm can't see this, but it selects the bid max row as the entering row for the cleared offer because it is the constraint with the lowest ratio of factor to RHS, which means the same thing as our observation; it is the constraint that will bind first as the entering variable, i.e., the cleared offer, is increased.

Solve complete

When the cleared offer enters the basis, the RHS of the bid max constraint sets the value for the cleared offer. Because there are now no negative reduced costs there is no way to further improve the objective; hence the solve is complete.

"The tremendous power of the simplex method is a constant surprise to me"

- George Dantzig (inventor of the simplex algorithm)

http://www-history.mcs.st-and.ac.uk/~history/Biographies/Dantzig_George.html

Extracting the results from the final tableau

Quantities from RHS values

In the final tableau shown in Table 20 the cleared offer and cleared bid variables are both basic, i.e., able to be non-zero.

The cleared bid (column 1) is basic in row 2, with a RHS of 100, the cleared offer (column 2) is basic in row 3, also with a RHS of 100. These RHS provide the value of these variables.

Table 20: The final tableau

Basic	$loadBid_{Clrd}$ [1]	$genOffer_{Clrd}$ [2]	s_1 [3]	s_2 [4]	s_3 [5]	s_4 [6]	RHS
[3]			1	1			
[1]	1				1		100
[2]		1		-1	1		100
[6]				1	-1	1	150
				70	90		9000

Shadow price of constraints

As discussed in Tutorial 1: Explaining Prices, the shadow price of a constraint is the rate of change of the objective value due to relaxing that constraint. The shadow price therefore indicates the value of the quantity that the constraint is constraining.

The bus price, i.e., the value of power at a bus, is the shadow price of the node balance constraint; the node balance constraint constrains the flow of power into and out of the bus, therefore the value of relaxing the node balance constraint tells us the value of power at the bus.

Shadow price from final tableau

The shadow prices can be extracted from the final tableau.

The values in the final tableau produce the optimum objective value and also meet the requirements of the original constraints. For example, given the original set of constraints, shown in Equation 32, we can plug in the values from the final tableau and the equations will solve.

Equation 32: Constraints of the linear programme (again)

$$-loadBid_{Cleared} + genOffer_{Cleared} + s_1 = 0$$

$$loadBid_{Cleared} - genOffer_{Cleared} \quad + s_2 = 0$$

$$loadBid_{Cleared} + s_3 = 10$$

$$genOffer_{Cleared} + s_4 = 20$$

Using the values from the final tableau, the second node balance constraint is shown in Equation 33.

$$loadBid_{Cleared} - genOffer_{Cleared} + s_2 = 0$$
$$100 - 100 + 0 = 0$$

Equation 33: Node balance with final tableau values

We can see from Equation 33 that decreasing the value of slack variable s_2 relaxes the node balance constraint; therefore the impact on the objective value of decreasing s_2 in the final tableau will provide the shadow price of the node balance constraint in the final tableau, and therefore the bus price.

To find the impact on the objective value of decreasing s_2 remember that the progress of the algorithm was driven by looking for negative reduced costs because they indicate the rate at which the objective value will improve if we increase the associated variable.

From this it follows that a positive reduced cost indicates the rate at which the objective will *decrease* if we could increase the variable. And also, the observation that helps us here, a positive reduced cost indicates the rate at which the objective value will increase if we could *decrease* the variable.

The positive reduced cost of s_2 provides the rate of objective value increase that would result from

decreasing s_2. Decreasing s_2 relaxes the node balance constraint that originally contained s_2. Therefore, the positive reduced cost of s_2 provides the improvement to the objective value due to relaxing the node balance constraint.

From the final tableau we can see that the reduced cost of s_2 is \$70/MW, which is therefore the shadow price of the node balance constraint.

The above is true of any of the constraints. The positive reduced cost, in the final tableau, of the constraint's original slack variable, will provide the shadow price of the constraint.

Viewing the simplex iterations

As well as exporting the tableaux to see what the simplex algorithm did, you can also view details of the simplex solve from within the app.

Iteration details

On the Results display select the Iterations row… the iteration details are displayed as shown in Figure 204 (these are the iterations from the small model that we have just studied).

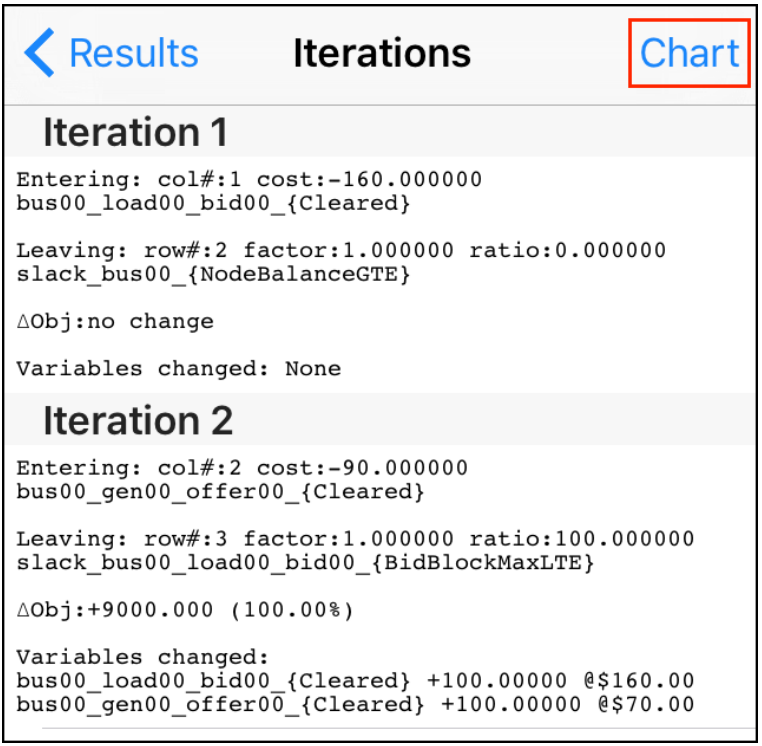

Figure 204: Iteration details display

Graphical view of iterations

The simplex iterations can also be viewed as a chart by tapping the Chart button on the Iterations display (this chart is also displayed while the solver is solving). The chart plots iterations vs objective, as shown in Figure 205. The chart shown in this example is from the Hawkes Bay 033 sample case presented in Tutorial 8: Actual Market Data.

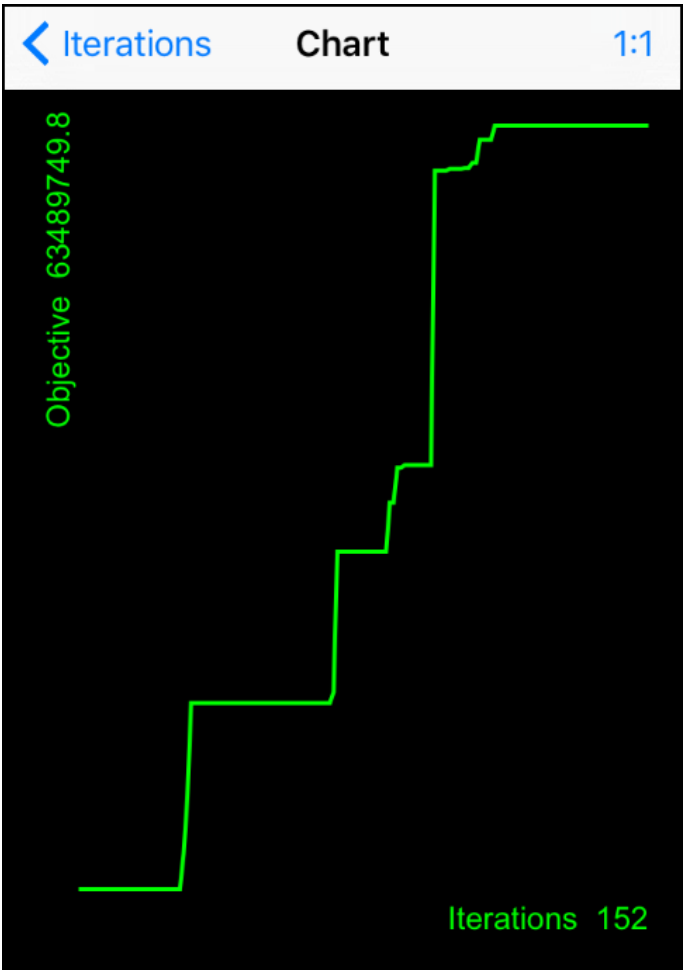

Figure 205: Iterations vs objective (shown here for the Hawkes Bay 033 model)

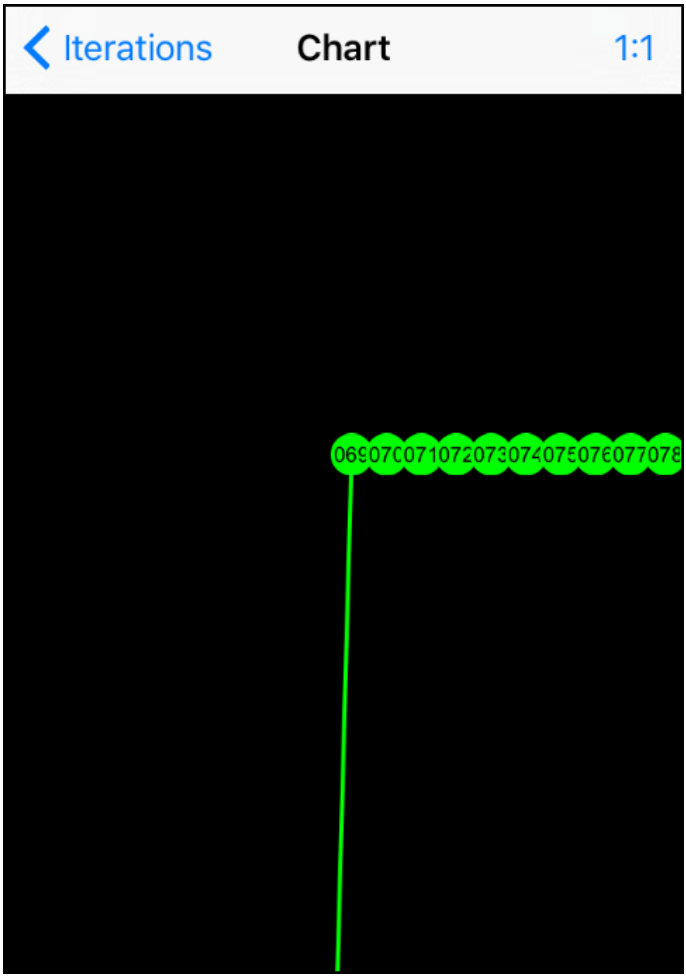

Figure 206: Zoom in to view individual iterations

Use the pinch gesture to zoom in and view the individual iteration points, as shown in Figure 206. Tapping on an iteration point will present the

simplex actions that produced this result, see the example in Figure 207.

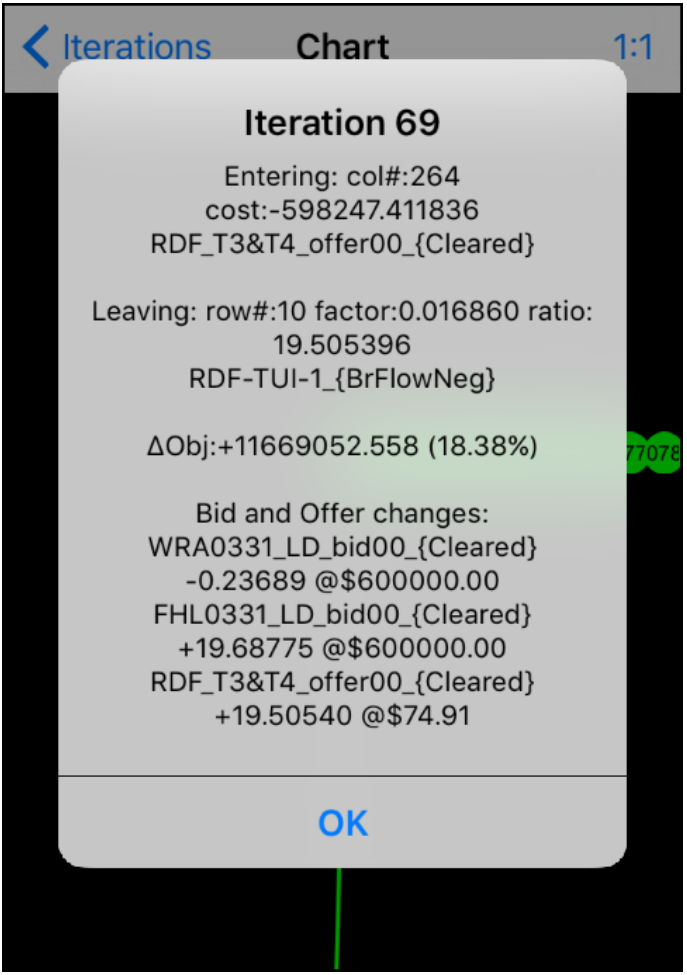

Figure 207: Tap iteration point to view simplex actions

Summary

In this section we built and solved a small electricity model then exported its tableaux and tracked in detail how the simplex algorithm arrived at the result. Along the way we explained how and why the simplex algorithm makes the decisions that it does, and why it works.

When the result was complete we saw how the prices and quantities are extracted from the final tableau.

We also saw how to view the simplex iterations in the app, either via a description of the iterations, or graphically as a chart of objective value vs iteration count.