Linear Programming

November 12, 2009

Parts of this introduction to linear programming were adapted from Chapter 29 of *Introduction to Algorithms*, Second Edition, by Cormen, Leiserson, Rivest and Stein.

1 What is Linear Programming?

The first thing to know about linear programming is that it is "programming" in the sense of trying to establish a plan or a schedule for something, rather than programming in the sense of trying to write some code to solve a problem. For example, a TV station has to solve the problem of deciding which shows to air at which times. This is a programming problem.

The kind of programming we are going to study is linear. "Linear" indicates that the constraints on the cost or profit of a decision are expressed as linear combinations of variables. A linear combination is a sum of terms, each of which is a constant-coefficient multiplied by a variable. The variables are raised to the first power. In each constraint, the linear combination is less than, greater than or equal to a constant.

In most cases, a linear programming problem will look something like this:

$$
\begin{aligned}
&\text{Maximize} \\
&x_1 - 5x_2 + 3x_3 \\
&\text{Subject to} \\
&x_1 - x_2 > 3 \\
&x_2 + 2x_3 < 2 \\
&5x_1 - 2x_2 + x_3 > 7
\end{aligned}
$$

The first equation is the **objective function** (which term should be familiar to you from greedy algorithms) and the other inequalities are the **constraints**. Another way to think about linear programming is as a constraint satisfaction and optimization problem.

objective function
constraints

It is not hard to make up simple examples of linear programming problems based on business problems. For example, suppose a furniture production company makes tables, chairs and window frames. The company can ask a price of 100 dollars for a table, 30 dollars for a chair and 35 dollars for selling a window frame. The production of furniture requires paying a cost for supplies and labor.

The problem here is to decide how many tables, chairs and window frames to make given the cost of each item. More concretely, if we let t = the number of tables produced, c = the number of chairs produced, w = the number of window frames produced, s = the total cost of supplies and l = the total cost of labor then we want to maximize the income, less expenses:

$$100t + 30c + 35w - s - l$$

subject to a set of equations that describe the cost of producing each item and equations that describe how much money we have on hand for buying labor and supplies. Suppose a table costs 20 dollars in supplies and 30 dollars in labor to produce; then we would add the constraints

$$\begin{aligned} 30t - l_t &= 0 \\ 20t - s_t &= 0 \end{aligned}$$

which require us to spend 30 dollars on labor for each table built and 20 dollars on supplies for each table built. Similarly, suppose it costs 10 dollars in supplies and 10 dollars in labor to make a chair and that it costs 15 dollars in supplies and 5 dollars in labor to make a window frame. We would add the following constraints:

$$\begin{aligned} 10c - l_c &= 0 \\ 10c - s_c &= 0 \\ 15w - l_w &= 0 \\ 5w - s_w &= 0 \end{aligned}$$

We needed different cost variables for each item, so we go back and add them in to the objective function and the constraint equations to get

$$\begin{aligned} &\text{Maximize} \\ &100t + 30c + 35w - s_t - s_c - s_w - l_t - l_c - l_w \\ &\text{Subject to} \\ &30t - l_t = 0 \\ &20t - s_t = 0 \\ &10c - l_c = 0 \\ &10c - s_c = 0 \\ &15w - l_w = 0 \\ &5w - s_w = 0 \end{aligned}$$

The final constraints are that we only have 400 dollars on hand to spend on labor and 300 on supplies. Those constraints are easily expressed as

$$\begin{aligned} l_t + l_c + l_w &\leq 400 \\ s_t + s_c + s_w &\leq 300 \end{aligned}$$

We can now pass our linear programming problem off to our favorite solver and find the most profitable way to program our furniture shop so as to maximize profits. The most profitable way is represented by assignments of values to the variables in the problem.

In the furniture problems, we tacitly assumed that t, c and w were integers. That is, we assumed we could only make an integer number of tables, chairs and window frames. This assumption makes sense for this problem. But for most problems we will assume that all variables can have arbitrary real values.

One interesting thing about linear programming is that, compared to graph formulations of problems, it is an entirely different way to think about a problem. In the graph representation, the challenge is to define the contents of each node in the graph, describe how child nodes are related to their parents and describe an algorithm for exploring the graph. In linear programming, the challenge is to formulate the problem as a set of equations and then solve them.

2 Solving Linear Programming Problems

In the next section, we will learn how to solve linear programming problems using the simplex method. The simplex method is a greedy algorithm. Interestingly, we can formulate the continuous (divisible) knapsack problem as a linear programming problem and use the simplex method to derive an algorithm that looks exactly like the greedy knapsack algorithm. First, we need some standardized forms for linear programming problems, so we examine those in this section. This will simplify our discussion.

2.1 Standard Form Step #1

The **standard form (after step #1)** for a linear program is

standard form (after step #1)

$$\begin{array}{ll} \text{Maximize} & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} & \\ & \sum_{j=1}^{n} a_{i,j} x_j \leq b_i \text{ for } i = 1, 2, \ldots m \\ & x_j \geq 0 \text{ for } j = 1, 2, \ldots n \end{array}$$

in which all variables range over the real numbers. Given a linear programming problem, it is always possible to add variables and constraints to get the problem into standard form (after step #1). We will assume all problems are in standard form (after step #1) henceforth.

The standard form (after step #1) includes the optimization of the objective function with n variables $(x_1 \ldots x_n)$ using m constraint equations of n terms each and a set of n non-negativity equations. It is often convenient (especially when implementing a linear program solver) to represent the variables for the linear programming problem in standard form (after step #1) as a matrix and a pair of vectors. The standard form (after step #1) includes an $m \times n$ matrix A containing the values of the a coefficients in the constraint equations, an n-dimensional vector b contains the b_i entries from the constraint equations, and an n-dimensional vector c contains the c_i entries from the objective function.

If we use the matrix and vectors representation, then the standard form

(after step #1) is

$$\begin{array}{ll} \text{Maximize} & c^T x \\ \text{subject to} & \\ & Ax \leq b \\ & x \geq 0 \end{array}$$

in which c^T is the transpose of c (to get a single value for the objective function, you just rotate c 90 degrees and multiply the resulting 1 x n matrix by x (which is an n by 1 matrix) to get a single scalar value) and 0 is the n-dimensional 0 vector (which is a vector containing 0 in every entry).

2.2 Standard Form Step #2

Standard form (after step #2) will allow us to assume that all linear programming problems are in the same form. We care about standard form because it is the form used in the simplex method. In standard form, the only inequalities allowed are non-negativity constraints; the other constraints must be equalities.

Completing the conversion to standard form requires two additional changes. First, a new variable, z is introduced to store the value of the objective function. Second, all of the constraint equations in standard form are rewritten as equalities, rather than inequalities, with a new non-negativity constraint. So the constraint

$$\sum_{j=1}^{n} a_{i,j} x_j \leq b_i$$

becomes the pair of constraints

$$\begin{array}{l} s = b_i - \sum_{j=1}^{n} a_{i,j} x_j \\ s \geq 0. \end{array}$$

The important thing here is that the new pair of constraints is satisfied if and only if the original constraint is satisfied. The variable s is called a **slack variable** because it measures the difference, or slack, between the left and right sides of the original constraint.

slack variable

It is convenient to assume that the slack variable for constraint i is called x_{n+i} instead of s. So that the standard form is

$$\begin{array}{l} x_{n+i} = b_i - \sum_{j=1}^{n} a_{i,j} x_j \\ x_{n+i} \geq 0. \end{array}$$

Initially, the original variables participating in the (linear combinations of the) constraints and the objective function are called the **non-basic variables**. The slack variables comprise the **basic variables**. They are "basic" because they stand alone on one side (the left-hand side in these notes) of each constraint.

non-basic variables

basic variables

At this point, it is a good idea to do an example. We will begin with the linear programming problem

$$\begin{array}{ll} \text{Maximize} & 2x_1 - 3x_2 + 3x_3 \\ \text{subject to} & \\ & x_1 + x_2 - x_3 \leq 7 \\ & -x_1 - x_2 + x_3 \leq -7 \\ & x_1 - 2x_2 + 2x_3 \leq 4 \\ & x_1, x_2, x_3 \geq 0 \end{array}$$

The first thing to notice is that this linear programming problem has been standardized up through standard form step #1. We want to convert it to standard form. First, we will add the slack variables x_4, x_5, x_6 and change the constraints to equalities.

$$\begin{array}{ll} \text{Maximize} & 2x_1 - 3x_2 + 3x_3 \\ \text{subject to} & \\ & x_4 = 7 - x_1 - x_2 + x_3 \\ & x_5 = -7 + x_1 + x_2 - x_3 \\ & x_6 = 4 - x_1 + 2x_2 - 2x_3 \\ & x_1, x_2, x_3, x_4, x_5, x_6 \geq 0 \end{array}$$

Finally, we add the z variable to track the value of the objective function.

$$\begin{array}{l} z = 2x_1 - 3x_2 + 3x_3 \\ x_4 = 7 - x_1 - x_2 + x_3 \\ x_5 = -7 + x_1 + x_2 - x_3 \\ x_6 = 4 - x_1 + 2x_2 - 2x_3 \\ x_1, x_2, x_3, x_4, x_5, x_6, z \geq 0 \end{array}$$

concise representation of standard form

For later use in the Simplex method, we will want a more concise representation of a linear programming problem in standard form. Just like before, we will use the matrix A and the vectors b and c to keep track of the coefficients in the constraints (using A), the constant terms in the slack equations (using b) and the coefficients in the objective function (using c). But, we will also need to know the indices for the basic variables and the indices for the non-basic variables[1]. The set B will contain the indices of the basic variables, and the set N will contain the indices of the non-basic variables. Finally, the value v will be an optional constant term in the objective function. The compact representation for the above linear program in standard form is:

$$\begin{array}{rcl} B & = & \{4, 5, 6\} \\ N & = & \{1, 2, 3\} \\ c & = & \begin{pmatrix} 2 & -3 & 3 \end{pmatrix}^T \\ b & = & \begin{pmatrix} 7 \\ -7 \\ 4 \end{pmatrix} \end{array}$$

[1] During execution of the algorithm, the basic and non-basic variable sets will evolve

$$
\begin{aligned}
A &= \begin{pmatrix} 1 & 1 & -1 \\ -1 & -1 & 1 \\ 1 & -2 & 2 \end{pmatrix} \\
v &= 0
\end{aligned}
$$

Notice that the coefficients of A are the additive inverse (i.e., 5 becomes -5) of their representation in standard form.. In this example, there is no constant term in the objective function so v is set to 0.

3 Simplex Algorithm

A simplex is a convex polygon and the simplex algorithm greedily visits the vertices of the polygon. A convex polygon is a polygon in which the line segment connecting any two points in the polygon is also contained in the polygon. For example, a stop sign is a convex polygon while the ASB (an X-shaped building) is not. For a specific linear programming problem, the simplex is formed by the constraints.

The simplex algorithm operates in a manner similar to solving a linear system using Gaussian elimination. Just in case you haven't taken linear algebra or don't remember: in Gaussian elimination the system is repeatedly transformed into an equivalent system until the structure of the resulting system yields an easily extracted solution. The simplex algorithm iterates in a similar manner.

We associate each iteration of the algorithm with a *basic solution*. A basic solution is obtained from the standard form of the linear programming problem by setting each non-basic variable to 0 and then computing the values of the basic variables from the remaining equality constraints. A basic solution corresponds to a vertex of the simplex. On each iteration of the algorithm, we are going to convert the current standard form into a new equivalent standard form whose basic solution is a different vertex of the simplex. The goal is to move through the vertices to find the one that maximizes the objective function. This is done by finding a non-basic variable that, when increased from 0, causes an increase in the objective value. The amount by which we can increase the non-basic variable is limited by the basic variables. In particular, we increase the non-basic variable until one of the basic variables become zero. This indicates that there is no more slack in the system (i.e., the difference between the function and its constraining inequality is 0). At this point we rewrite the standard form to exchange the chosen non-basic variable with the basic variable that is now 0. This is a new basic solution and we repeat the process until we cannot find a non-basic variable to increase that gives an increase in the objective function.

3.1 Simplex Algorithm Example

An extended example will help see the process. Consider the following linear program in standard form:

$$\begin{array}{lrcrcrcr} \text{maximize} & 3x_1 & + & x_2 & + & 2x_3 & & \\ \text{subject to} & x_1 & + & x_2 & + & 3x_3 & \leq & 30 \\ & 2x_1 & + & 2x_2 & + & 5x_3 & \leq & 24 \\ & 4x_1 & + & x_2 & + & 2x_3 & \leq & 36 \\ & & & & & x_1, x_2, x_3 & \geq & 0 \end{array}$$

We begin by converting this into its equivalent standard from. Standard form is useful for algebraic as well as algorithmic manipulation. We say that a constraint is *tight* for a particular setting of its non-basic variables if they cause the constraint's basic variable to become 0. Similarly, a setting of the non-basic variables that would make a basic variable become negative *violates* that constraint. The slack variable thus represents how far away a constraint is from being tight, and they help us determine the limit to which we can increase non-basic variables without violating any constraints.

We convert to standard form by creating a slack variable for each constraint, converting the inequality to equality constraints, and creating a variable for the objective function. Note that we drop the non-negativity constraints since they are implied by the standard form. The standard form of the problem is:

$$\begin{array}{rcrcrcrcr} z & = & & & 3x_1 & + & x_2 & + & 2x_3 \\ x_4 & = & 30 & - & x_1 & - & x_2 & - & 3x_3 \\ x_5 & = & 24 & - & 2x_1 & - & 2x_2 & - & 5x_3 \\ x_6 & = & 36 & - & 4x_1 & - & x_2 & - & 2x_3 \\ & & x_1, x_2, x_3 x_4, x_5, x_6 & & & & \geq & & 0 \end{array}$$

A **feasible solution** to this system is any assignment of positive values to x_1, x_2, and x_3 that yields positive values for x_4, x_5, and x_6; thus, there are an infinite number of feasible solutions to this system. We are interested in the *basic solution* obtained by setting all the non-basic variables to zero (i.e., the variables on the right-hand side of the equalities). The basic solution for our example is $(\bar{x_1}, \bar{x_2}, \bar{x_3}, \bar{x_4}, \bar{x_5}, \bar{x_6}) = (0, 0, 0, 30, 24, 36)$, and it has an objective value $z = (3 \cdot 0) + (1 \cdot 0) + (2 \cdot 0) = 0$. Note that the basic solution sets $\bar{x_i} = b_i$ for each $i \in B$. An iteration of the simplex algorithm rewrites the objective function and constraint equations to create a different set of non-basic variables. This in turn creates a new basic solution. The rewrite in no way changes the underlying linear program that is being solved. Rather, it represents the movement from one vertex to another in exploring the simplex. Note that it is possible that the basic solution is not feasible during the first few iterations of the algorithm. This is OK.

feasible solution

We reformulate the linear programming problem in an iteration of the algorithm by picking a non-basic variable x_e and increasing its value to see if it increases the objective function value. If this is the case, then we continue to

increase x_e until a basic variable x_l becomes 0. At this point, x_e becomes basic (i.e., it has a nonzero value) and x_l becomes non-basic (i.e., it has a zero value). We choose our non-basic value to increase by looking at the objective function. If a non-basic variable exists in the objective function with a positive coefficient, then we can select that variable as x_e in our algorithm. Keep in mind that as we increase x_e other basic variables and our objective functions may also change. The first basic variable to reach 0 becomes x_l.

To continue with our example, notice that x_1 is a non-basic variable with a positive coefficient in the objective function. We select this as the variable to increase. As we increase x_1, the variables x_4, x_5, and x_6 decrease. The non-negativity constraint prevents us from increasing x_1 above 9 since x_6 becomes negative at that point; thus, x_6 is our tightest constraint since it limits how much we can increase x_1. We now switch the roles of x_1 and x_6 by solving the third constraint for x_1:

$$\begin{aligned} x_6 &= 36 - 4x_1 - x_2 - 2x_3 \\ 4x_1 &= 36 - x_2 - 2x_3 - x_6 \\ x_1 &= 9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4} \end{aligned}$$

We now rewrite the other equations by writing x_1 in terms of x_2, x_3, and x_6 using the above equation. Doing this for x_4 gives us

$$\begin{aligned} x_4 &= 30 - x_1 - x_2 - 3x_3 \\ &= 30 - \left(9 - \frac{x_2}{4} - \frac{x_3}{2} - \frac{x_6}{4}\right) - x_2 - 3x_3 \\ &= 21 - \frac{3x_2}{4} - \frac{5x_3}{2} + \frac{x_6}{4} \end{aligned}$$

We repeat this procedure for the remaining constraint and objective function to rewrite our linear program in the following form:

$$\begin{array}{rcrcrcrcr} z &=& 27 &+& \frac{x_2}{4} &+& \frac{x_3}{2} &-& \frac{3x_6}{4} \\ x_1 &=& 9 &-& \frac{x_2}{4} &-& \frac{x_3}{2} &-& \frac{x_6}{4} \\ x_4 &=& 21 &-& \frac{3x_2}{4} &-& \frac{5x_3}{2} &+& \frac{x_6}{4} \\ x_5 &=& 6 &-& \frac{3x_2}{2} &-& 4x_3 &+& \frac{x_6}{2} \end{array}$$

The rewrite operation as shown above is called a *pivot*. A pivot takes the non-basic variable x_e called the *entering variable* and the basic variable x_l called the *leaving variable* and exchanges their roles in the linear program.

Pivoting rewrites that linear program into an equivalent form. The original basic solution to our linear program was $(0, 0, 0, 30, 24, 36)$ with an objective value of 0. The new basic solution to the linear program after the pivot is $(9, 0, 0, 21, 6, 0)$ with an objective value of 27. This solution is feasible in our linear program before the pivot, and it yields the same objective value.

Continuing the example, we now find a new non-basic variable to increase. We do not want to increase x_6 since it has a negative coefficient. We can attempt to increase either x_2 or x_3. Let us use x_3. We can increase x_3 to

$\frac{3}{2}$ before the third constraint becomes negative, so the third constraint is the tightest constraint. We pivot on x_3 and x_5 by solving for x_3 on the right-hand side of the third constraint and substituting it into the other equations to rewrite the linear program to

$$\begin{array}{rcrcrcrcr} z & = & \frac{111}{4} & + & \frac{x_2}{16} & - & \frac{x_5}{8} & - & \frac{11x_6}{16} \\ x_1 & = & \frac{33}{4} & - & \frac{x_2}{16} & + & \frac{x_5}{8} & - & \frac{5x_6}{16} \\ x_3 & = & \frac{3}{2} & - & \frac{3x_2}{8} & - & \frac{x_5}{4} & + & \frac{x_6}{8} \\ x_4 & = & \frac{69}{4} & + & \frac{3x_2}{16} & + & \frac{5x_5}{8} & - & \frac{x_6}{16} \end{array}$$

There is still a non-basic variable in the objective function with a positive coefficient. We increase this variable and pivot again to rewrite the linear program as:

$$\begin{array}{rcrcrcrcr} z & = & 28 & - & \frac{x_3}{6} & - & \frac{x_5}{6} & - & \frac{2x_6}{3} \\ x_1 & = & 8 & + & \frac{x_3}{6} & + & \frac{x_5}{6} & - & \frac{x_6}{3} \\ x_2 & = & 4 & - & \frac{8x_3}{3} & - & \frac{2x_5}{3} & + & \frac{x_6}{3} \\ x_4 & = & 18 & - & \frac{x_3}{2} & + & \frac{x_5}{2} & & \end{array}$$

There are no other variables we can change to increase the objective value. The basic solution to the linear program is $(8, 4, 0, 18, 0, 0)$. The objective value from this solution is 28. We can now return to our original linear programming. The only variables in the original program are x_1, x_2, and x_3. Using our basic solution these variables are $x_1 = 8$, $x_2 = 4$, and $x_3 = 0$. Notice that these values give an objective value of 28 as expected. Note that our final solution assigns integers to every variable; however, this will not always be the case.

3.2 Pivoting

Pivoting is a key, perhaps the key, operation in the simplex algorithm. The pseudocode for the pivoting algorithm is given in Figure 1. The parameters passed to the algorithm are the matrices, vectors and integers in the concise representation of the standard form (the definition of the concise representation is on page 5).

The pseudocode for the simplex algorithm is given in Figure 2. The basic idea of the simplex algorithm is to choose a pair of entering and leaving variables. The entering variable, e which is chosen on line 7, is a non-basic variable (i.e., its index is in N) with a positive coefficient in the objective function (i.e., $c_e > 0$). The significance of the entering variable is that if we increase the value of the entering variable then we can increase the value of the objective function and that was our goal in the first place. Note that the algorithm, on line 7, doesn't give you any guidance on how to pick the entering variable. You just have to pick one, of possibly many, non-basic variables in the objective function with a positive coefficient. You may decide to pick the non-basic variable with the *biggest* positive coefficient, or you might use some other method.

The for-loop in lines 8 through 12 determines how much we can change the value of the entering variable while allowing the basic variables (i.e., variables with index in B) for each constraint to remain positive. In line 12, we pick the

```
1   Pivot (N, B, A, b, c, v, l, e)
2   // Compute coefficients for equation for the new basic variable x_e
3   // The variables with hats, like b̂_e, will be the return values
4   b̂_e = b_l / a_{l,e}
5   for each j ∈ N − {e}
6       do â_{e,j} = a_{l,j} / a_{l,e}
7   â_{e,l} = 1 / a_{l,e}
8   // Compute coefficients for the other constraints
9   for each i ∈ B − {l}
10      do b̂_i = b_i − a_{i,e} b̂_e
11          for each j ∈ N − {e}
12              do â_{i,j} = a_{i,j} − a_{i,e} â_{e,j}
13          â_{i,l} = −a_{i,e} â_{e,l}
14  // Compute the objective function
15  v̂ = v + c_e b̂_e
16  for each j ∈ N − {e}
17      do ĉ_j = c_j − c_e â_{e,j}
18  ĉ_l = −c_e â_{e,l}
19  // Compute new basic and non-basic variable sets
20  N̂ = (N − {e}) ∪ {l}
21  B̂ = (B − {l}) ∪ {e}
22  return (N̂, B̂, Â, b̂, ĉ, v̂)
```

Figure 1: The Pivot algorithm.

index of the basic variable with the tightest constraint (i.e., let $l = i$ such that δ_i is the smallest of the δs).

We repeat that process until either there are no more non-basic variables with positive coefficients or we discover that the value of the objective function is unbounded (line 14). After completing the pivot operations, the only remaining task is to return the values of the nonzero basic variables.

The initial assumptions are made to simplify the project. We could have InitializeSimplex put an arbitrary LP in standard form, but we will just assume that this is done previously. In general, an LP may not have the origin as a vertex. The origin could be inside or outside of the feasible region rather than at a vertex. The more general approach of implementing the InitializeSimplex algorithm would change the basis to make the origin a vertex if necessary. If would also ensure that the first basic solution is feasible. These can be done by just using another LP as a preprocess, so we will skip it – you can refer to Section 7.6.3 of the text for more details on this issue.

```
1  // Previously convert the problem to standard form.
2  // We assume that the first basic solution is at the origin and is feasible.
3  // This is not true in general but makes our problem easier.
4  Simplex (A, b, c)
5  (N, B, A, b, c, v) = InitializeSimplex(A, b, c) //Initialize data structures
6  while there exists some j ∈ N such that c_j > 0
7      do choose an index e ∈ N such that c_e > 0 // e is entering variable
8          for each index i ∈ B
9              do if a_{i,e} > 0
10                 then δ_i = b_i / a_{i,e}
11                 else δ_i = ∞
12         choose l ∈ B such that δ_i is minimized // l is leaving variable
13         if δ_l = ∞
14             then return "unbounded"
15             else (N, B, A, b, c, v) = Pivot (N, B, A, b, c, v, l, e)
16 // set the non-basic variables to 0 and everything else to the optimal solution
17 for i = 1 to n
18     do if i ∈ B
19         then x̄_i = b_i
20         else x̄_i = 0
21 return (x̄_1, x̄_2, ... x̄_n)
```

Figure 2: The simplex algorithm.

4 Implementation Notes

There are a few small, but important details, in the implementation of the simplex algorithm in Figure 2.

Recall that we decided that we could represent a linear programming problem in concise matrix form (defined on page 5) with the matrix A and the vectors b and c to keep track of the coefficients in the constraints (using A), the constant terms in the slack equations (using b) and the coefficients in the objective function (using c). But, we will also need to know the indices for the basic variables and the indices for the non-basic variables. The set B will contain the indices of the basic variables and the set N will contain the indices of the non-basic variables. Finally, the value v will be an optional constant term in the objective function.

The first detail is that the $a_{i,j}$ entries in the A coefficient matrix are actually the negated values of the coefficients as they appear in the standard form equations. Note that we did this in the example on page 6. This is because the matrices work out to be

$$z = v + \sum_{j \in N} c_j x_j$$
$$x_i = b_i - \sum_{j \in N} a_{i,j} x_j \text{ for } i \in B$$

and the $a_{i,j}$ terms are subtracted from the b_i term in each equation.

The second detail is that the A array in the simplex algorithm should be of size $(n+m) \times (n+m)$ rather than $m \times n$. Recall that n is the number of non-basic variables and m is the number of basic variables. Since each basic variable is the left side of a constraint equation, and there is one row in A per constraint, then you'd expect that there would only be m rows in A. But there are n rows in A because any basic or non-basic variable can be pivoted into a basic variable. When a formerly non-basic variable is pivoted into becoming a basic variable, then the new basic variable becomes the left side of a new constraint equation. In the algorithm, you need a simple and easy way to find the new constraint equation for the new basic variable. The way this is done is to keep the constraint equation for the basic variable with index i at row i of A. Since any of the $n+m$ variables can be a basic variable during the run of the algorithm, you need $n+m$ rows. Initially pad the extra cells with 0's.

You also need $n+m$ columns because the coefficient for variable x_j (which might be 0 or might not be) in the constraint for variable x_i needs a place to stay which is also easy to find later. A pivot operation allows any variable to end up in the constraints for any other variable so you need all $n+m$ columns in addition to all $n+m$ rows.

Similarly, the c and b vectors will contain $n+m$ entries as well.

If you don't get this right now, then you should probably work out an example on paper before trying to code up the algorithm. Doing so will save you a lot of time since you will be more likely to generate correct code.