Optimization Theory (DS2) Lecture #4
The Heart of the Simplex Algorithm

January 17, 2017

Abstract

Today we are going to look at the simplex algorithm, as developed by George Dantzig (1947). We will cover Secs. 2.4 and 2.5 of the textbook. *In my opinion, this is the hardest week of the semester, stick with me here!*

Text and examples adapted from A Gentle Introduction to Optimization.

1 Review

1.1 Standard Equality Form (SEF)

An LP is in *standard equality form* if:

1. it is a maximization problem;
2. other than the non-negativity constraints, all constraints are *equalities*; and
3. every variable has a non-negativity constraint.

That is, it can be written in the form

$$\max\{z(\vec{x}) = \vec{c}^{\intercal}\vec{x} : A\vec{x} = \vec{b}, \vec{x} \geq \vec{0}\}. \tag{1}$$

In this formulation, $\vec{c}$ is the vector of coefficients of each of our variables in the objective function. It will have two entries for each free variable we adapted and will have zeroes for each slack variable we added. A and $\vec{b}$ have one line for each constraint in our problem.

1.2 A Simplex Iteration

The basic idea is to increase one of our variables x_i until it hits a "stop" against one of the constraints. In the main body of the simplex algorithm, this will involve running along an edge of the *polytope* (see below) until we reach a vertex.

Consider the following LP in SEF:

$$\max z(\vec{x}) = (2,3,0,0,0)(x_1,x_2,x_3,x_4,x_5)^\intercal \tag{2}$$

subject to

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 & 1 \end{pmatrix} \vec{x} = \begin{pmatrix} 6 \\ 10 \\ 4 \end{pmatrix} \tag{3}$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0. \tag{4}$$

Given the solution $\vec{x_a} = (0,0,6,10,4)^\intercal$ (easy to see that's a solution – look at the right side of the array), $z(\vec{x_a}) = 0$.

Now try increasing x_1, choosing $x_1 = t$. A little algebra gives

$$\begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 6 \\ 10 \\ 4 \end{pmatrix} - t \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} \tag{5}$$

from which we get

$$t \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} \leq \begin{pmatrix} 6 \\ 10 \\ 4 \end{pmatrix} \tag{6}$$

which gives us the limiting inequality $t \leq 10/2 = 5$. Setting $t = 5$, $\vec{x'} = (5,0,1,0,9)^\intercal$ and $z(\vec{x'}) = 10$.

Unfortunately, we can't yet apply the same trick to x_2, so that's the topic for this week.

2 Bases

(Adapted from Sec. 2.4.1 in the textbook.)

Before we get to the algorithm itself, we need to discuss the idea of a *basis* for a set of variables (dimensions), and *canonical form*.

Recall that the *inverse* of a square matrix A is the matrix A^{-1} such that $AA^{-1} = I$, where I is the identity matrix. A *singular* matrix has no inverse, or equivalently, its determinant is 0. A *nonsingular* matrix has an inverse and a non-zero determinant. (Question: is the identity matrix singular or nonsingular?)

A is the $m \times n$ matrix with m linearly independent rows that comprise our constraints. We need to partition our columns into two sets, one that comprises a *basis*, which we will call B, and the rest, which we will call N.

Consider

$$A = \begin{pmatrix} 2 & 1 & 2 & -1 & 0 & 0 \\ 1 & 0 & -1 & 2 & 1 & 0 \\ 3 & 0 & 3 & 1 & 0 & 1 \end{pmatrix}. \tag{7}$$

The set of columns $B = \{2, 5, 6\}$ is a basis, which we can see easily by taking

$$A_B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{8}$$

which, if you think about it as a three-dimensional space, is the vector composed of the x, y and z unit vectors.

Question: For the simplex algorithm, does our basis set have to be orthonormal?

The set of columns $B = \{1, 5, 6\}$ is also a basis:

$$A_B = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix}. \tag{9}$$

You can see pretty easily how to turn that into the identity matrix by substracting 0.5 times the first row from the second, and 1.5 times the first row from the third, after which our basis is orthonormal.

We will need a *basic solution* $\vec{x_a}$ for our problem $A\vec{x} = b$. $\vec{x_a}$ is a basic solution iff:

1. $A\vec{x_a} = b$, and
2. $x_N = \vec{0}$.

Every basis has a unique basic solution, which we are going to need. For $B = \{1, 5, 6\}$, the basic solution is $\vec{x_a} = (1, 0, 0, 0, 0, -2)^\intercal$.

3 Canonical Forms

(Adapted from Sec. 2.4.2 in the textbook.)

Consider an LP (P) in SEF:

$$\max\{\vec{c}^\intercal \vec{x} + \vec{z_a} : A\vec{x} = b, \vec{x} \geq \vec{0}\} \tag{10}$$

where $\vec{z_a}$ is a constant. Let B be a basis of A. (P) is in *canonical form for* B if:

1. A_B is an identity matrix, and
2. $\vec{c_B} = \vec{0}$.

Let's return to the LP in Eqs. 2-4 above. Obviously, $B = \{3, 4, 5\}$ is a basis, but that's boring, it's already in canonical form. $B = \{1, 2, 4\}$ is also a basis, so let's rewrite our program in canonical form for this basis:

$$\max z(\vec{x}) = 17 + (0, 0, -5/2, 0, -1/2)(x_1, x_2, x_3, x_4, x_5)^\intercal \tag{11}$$

subject to

$$\begin{pmatrix} 1 & 0 & 1/2 & 0 & -1/2 \\ 0 & 1 & 1/2 & 0 & 1/2 \\ 0 & 0 & -3/2 & 1 & 1/2 \end{pmatrix} \vec{x} = \begin{pmatrix} 1 \\ 5 \\ 3 \end{pmatrix} \tag{12}$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0. \tag{13}$$

(Notice that now columns 1, 2, 4 form an identity matrix.) So how did we get here?

3.1 Interlude: calculating the inverse of a matrix using Octave

The following ad hoc Octave code will create a matrix for A_B, take its inverse, and left multiply it with the full constraint matrix A.

```
octave:1> AB = [ 1, 1, 0; 2, 1, 1; -1, 1, 0]
AB =

   1   1   0
   2   1   1
  -1   1   0

octave:2> A = [1, 1, 1, 0, 0; 2, 1, 0, 1, 0; -1, 1, 0, 0, 1]
A =

   1   1   1   0   0
   2   1   0   1   0
  -1   1   0   0   1

octave:3> ABinv = inv(AB)
ABinv =

   0.50000   0.00000  -0.50000
   0.50000   0.00000   0.50000
  -1.50000   1.00000   0.50000

octave:4> ABinv*A
ans =

   1.00000   0.00000   0.50000   0.00000  -0.50000
   0.00000   1.00000   0.50000   0.00000   0.50000
   0.00000   0.00000  -1.50000   1.00000   0.50000
```

3.2 Interlude: calculating the inverse of a matrix using R

The following ad hoc R code will create a matrix for A_B, take its inverse, and left multiply it with the full constraint matrix A. Note the use of `%*%` to do the matrix

multiplication; simply using $\star$ does element-by-element multiplication. The function `solve()`, for our purposes here, inverts the matrix.

```
> AB = t(matrix(
+ c(1, 1, 0,  2, 1, 1,  -1, 1, 0),
+ nrow = 3,
+ ncol = 3))
> AB
     [,1] [,2] [,3]
[1,]    1    1    0
[2,]    2    1    1
[3,]   -1    1    0
> A = t(matrix(
+ c(1, 1, 1, 0, 0,  2, 1, 0, 1, 0,  -1, 1, 0, 0, 1),
+ nrow=5, ncol=3))
> A
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    1    1    0    0
[2,]    2    1    0    1    0
[3,]   -1    1    0    0    1
> ABinv <- solve(AB)
> ABinv
     [,1] [,2] [,3]
[1,]  0.5    0 -0.5
[2,]  0.5    0  0.5
[3,] -1.5    1  0.5
> ABinv %*% A
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0  0.5    0 -0.5
[2,]    0    1  0.5    0  0.5
[3,]    0    0 -1.5    1  0.5
```

3.3 Now Back to Our Regularly Scheduled Program

So how did we get that canonical form in Eqs. 11 to 13? There's about 3.5 pages of linear algebra in the textbook...

Fundamentally, take the program (P):

$$\max\{z(\vec{x}) = \vec{c}^{\intercal}\vec{x} + \vec{z_a}\} \tag{14}$$

subject to

$$A\vec{x} = \vec{b} \tag{15}$$

$$\vec{x} \geq \vec{0} \tag{16}$$

find your next preferred basis B, and our new, equivalent program is

$$\max\{z(\vec{x}) = \vec{y}^\intercal \vec{b} + \vec{z_a} + (\vec{c}^\intercal - \vec{y}^\intercal A)\vec{x}\} \tag{17}$$

subject to

$$A_B^{-1} A\vec{x} = A_B^{-1}\vec{b} \tag{18}$$

$$\vec{x} \geq \vec{0}, \tag{19}$$

where

$$\vec{y} = A_B^{-\intercal} \vec{c_B}. \tag{20}$$

Or, I prefer to write it

$$\max\{z'(\vec{x}) = \vec{c'}^\intercal \vec{x} + \vec{z'_a}\} \tag{21}$$

subject to

$$A'\vec{x} = \vec{b'} \tag{22}$$

$$\vec{x} \geq \vec{0} \tag{23}$$

where

$$\vec{y} = A_B^{-\intercal} \vec{c_B} \tag{24}$$

$$\vec{c'}^\intercal = \vec{c}^\intercal - \vec{y}^\intercal A \tag{25}$$

$$\vec{z'_a} = \vec{y}^\intercal \vec{b} + \vec{z_a} \tag{26}$$

$$A' = A_B^{-1} A \tag{27}$$

$$\vec{b'} = A_B^{-1}\vec{b}. \tag{28}$$

This, then, is the basic process of canonization (has nothing to do with the Catholic saints):

1. rewrite the conditions using the procedure above, to satisfy the first condition;
2. rewrite the objective function using $\vec{y}$ from Eq. 24.

3.4 The Basic Solution

(This is material not covered in class.)

Every basis is associated with a unique *basic solution*. The elements of the basic solution $\vec{x_a}$ related to the basis B, which we will call the vector $\vec{x_B}$, come from the equation

$$\vec{x_B} = A_B^{-1}\vec{b}. \tag{29}$$

The elements not related to B are part of the partition N, of course, and all of those are 0, $\vec{x_N} = \vec{0}$. Put $\vec{x_B}$ and $\vec{x_N}$ back together in the proper order to get $\vec{x_a}$.

4 The Simplex Algorithm

4.1 The Convex Polytope

A *polytope* is an n-dimensional polygon (2-D) or polyhedron (3-D). It is *convex* if a line drawn between any two points on the surface of the polytope passes only through its interior.

A polytope can be created by cutting the space with a set of planes. In a linear program, each constraint defines a plane, and the total set defines the polytope. For our purposes here, the polytope doesn't have to be bounded in all dimensions and directions. Before any constraints, we start with the unbounded polytope that is the positive quadrant of our space. If all of the constraints are necessary, adding the constraints one by one will chop off some of the space and create a new face for the polytope, keeping it convex and adding some number of vertices and edges in the process.

Ultimately, the possible solution space of a linear program with a feasible solution is a polytope. One (or more) of the vertices of the polytope will represent the optimal solution. Thus, we can find that vertex by starting at any vertex and sliding along the edges until we find the optimal one. This sliding is our basic *simplex operation*. (Mathematicians will tell you it's not a precise name, that "simplex" really means something else, but it's the common name, so we'll go with it.)

Question: Does the basic feasible solution we begin with have *to be a vertex? Why won't any point in the interior of the polytope do? In three dimensions, running in any direction will first encounter a face, then a second run along the face will find an edge, and the third run along the edge will find a vertex. Will these three steps suffice in 3 dimensions? Perhaps the basis rotation in the core algorithm isn't good enough to make the algorithm work right, unless we first find a vector normal to the face we encounter on the first run? What about n dimensions?*

4.2 Anzen Dai-ichi

The idea of the simplex algorithm is pretty simple: repeat a simplex operation until no more simplex operations improve the result, and you're done. But there are a couple of things we need to worry about:

1. The second simplex operation might partly undo the work of the first!
2. At each vertex, we have several possible choices for which way to go next. If we choose poorly:
   (a) we might be inefficient in finishing; or
   (b) worse, we might wind up looping forever!
3. We need to be careful to recognize when we're on an edge that extends forever (that is, the solution is unbounded).
4. Finally, of course, we need some way to recognize when we are done, and have found the (or an) optimal solution.

4.3 The Basic Algorithm

See the separate file uploaded to SFS for pseudocode for the basic algorithm, slightly reformatted from p. 70 of textbook.

5 Homework

See the separate file uploaded to SFS.