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

May 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

There were a lot of blank looks last time as we discussed the geometric nature of the constraints and the objective function, and we didn't even get to discuss *standard equality form* (SEF) and the core idea of the *simplex operation*. So let's begin here with some geometry, then go back and pick up what we were supposed to cover in Sec. 4 and 5 of last time, before we start into the grungy parts of simplex.

1.1 The Geometric Nature of Constraints and Objective Functions

Or, *A Simple(!) Example of Prepping a Linear Program on a 2-D Simplex (i.e., a Triangle)*

By now, you've heard me say that a *simplex* is an n-dimensional triangle, but that the *simplex algorithm* applies to *polytopes* (n-dimensional polygons).

Let's look at a problem formulation for a 2-D simplex, that is, a triangle. Examples in this section are in Octave, rather than in R.

Consider the following LP, not yet in SEF (defined in the next section):

$$\max z(\vec{x}) = (\frac{19}{14}, -\frac{1}{28})(x_1, x_2)^\intercal \tag{1}$$

subject to

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 3 \end{pmatrix} \vec{x} \begin{matrix} \geq \\ \geq \\ \leq \end{matrix} \begin{pmatrix} 2 \\ 4 \\ 28 \end{pmatrix} \tag{2}$$

$$x_1, x_2 \geq 0. \tag{3}$$

It should be obvious that the first constraint is $x_1 \geq 2$, and the second constraint is $x_2 \geq 4$. The third constraint, $2x_1 + 3x_2 \leq 28$, is a diagonal line. These constraints, one at a time, then the whole set, are illustrated in Fig. 1. Our objective function is shown in Fig. 2, with the code for all of these figures in the next subsection.

Our first important task is to put the problem into standard equality form (SEF). Since our constraints are inequalities, we have to add slack variables, x_3, x_4 and x_5. (Slack variables and free variables are covered more rigorously in Sec. 1.3.)

$$\max z(\vec{x}) = (\frac{19}{14}, -\frac{1}{28})(x_1, x_2)^\intercal \tag{4}$$

subject to

$$\begin{pmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 2 & 3 & 0 & 0 & 1 \end{pmatrix} \vec{x} = \begin{pmatrix} 2 \\ 4 \\ 28 \end{pmatrix} \tag{5}$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0. \tag{6}$$

Now our program is in SEF, nice and pretty and waiting to be solved!

1.2 Interlude in Octave

We can draw each of those constraints using the following Octave code:

```
>> c1 = [0 0; 0 10; 2 10; 2 0];
>> c1v = [0 0; 0 10; 2 10; 2 0];
>> c2v = [0 0; 0 4; 10 4; 10 0];
>> fs = [ 1 2 3 4];
>> c3v = [0 28/3; 0 10; 10 10; 10 8/3];
>> fs3 = [1 2 3 4];
>> figure
>> xlim([0 10])
>> ylim([0 10])
>> axis('equal')
>> patch("Faces", fs, "Vertices", c1v, 'FaceColor', 'blue');
>> figure
>> xlim([0 10])
>> ylim([0 10])
>> axis('equal')
>> patch("Faces", fs, "Vertices", c2v, 'FaceColor', 'yellow');
>> figure
>> xlim([0 10])
>> ylim([0 10])
>> axis('equal')
>> patch("Faces", fs3, "Vertices", c3v, 'FaceColor', 'red');
>> figure
>> xlim([0 10]); ylim([0 10]); axis('equal');
>> patch("Faces", fs, "Vertices", c1v, 'FaceColor', 'blue');
>> patch("Faces", fs, "Vertices", c2v, 'FaceColor', 'yellow');
>> patch("Faces", fs3, "Vertices", c3v, 'FaceColor', 'red');
```

And plot the constrained objective function using:

```
>> v = [2 4; 2 8; 8 4]
v =

   2   4
   2   8
   8   4

>> f = [1 2 3]
f =

   1   2   3
>> col = [2.57; 1; 5]
col =

   2.5700
   1.0000
   5.0000

>> figure
>> xlim([0 10])
>> ylim([0 10])
>> axis('square')
>> axis('equal')
>> patch('Faces',f,'Vertices',v,'FaceVertexCData',col,'FaceColor','interp');
>> colorbar
```

1.3 Standard Equality Form (SEF)

Adapted from Sec. 4 of last time's text.

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

$$\max\{z(\vec{x}) = \vec{c}^{\intercal}\vec{x} : A\vec{x} = \vec{b}, \vec{x} \geq \vec{0}\}. \tag{7}$$

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.

Consider the LP

$$\max(1,-2,4)(x_1,x_2,x_3)^\intercal \tag{8}$$

subject to

$$\begin{pmatrix} 1 & 5 & 3 \\ 2 & -1 & 2 \\ 1 & 2 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \begin{matrix} \geq \\ \leq \\ = \end{matrix} \begin{pmatrix} 5 \\ 4 \\ 2 \end{pmatrix} \tag{9}$$

$$x_1, x_2 \geq 0. \tag{10}$$

x_3 is called a *free variable* since it doesn't have a non-negativity constraint. But a lot of our techniques and especially proofs depend on that. So, we introduce two new variables, x_3^+ and x_3^-, set $x_3 = x_3^+ - x_3^-$, and add constraints $x_3^+, x_3^- \geq 0$. After some algebra, you get

$$\max(1,-2,4,-4)(x_1,x_2,x_3^+,x_3^-)^\intercal \tag{11}$$

subject to

$$\begin{pmatrix} 1 & 5 & 3 & -3 \\ 2 & -1 & 2 & -2 \\ 1 & 2 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3^+ \\ x_3^- \end{pmatrix} \begin{matrix} \geq \\ \leq \\ = \end{matrix} \begin{pmatrix} 5 \\ 4 \\ 2 \end{pmatrix} \tag{12}$$

$$x_1, x_2, x_3^+, x_3^- \geq 0. \tag{13}$$

Getting closer, but we're still not quite there; we don't have equalities everywhere yet. So we introduce two *slack variables* x_4 and x_5. Now we get

$$\max(1,-2,4,-4,0,0)(x_1,x_2,x_3^+,x_3^-,x_4,x_5)^\intercal \tag{14}$$

subject to

$$\begin{pmatrix} 1 & 5 & 3 & -3 & 0 & -1 \\ 2 & -1 & 2 & -2 & 1 & 0 \\ 1 & 2 & -1 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3^+ \\ x_3^- \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \\ 2 \end{pmatrix} \tag{15}$$

$$x_1, x_2, x_3^+, x_3^-, x_4, x_5 \geq 0 \tag{16}$$

and we're finally in SEF! Note that when we replaced a $\leq$ constraint (second line of Eq. 12), we added a 1 to the array, and when we replaced a $\geq$ constraint (first line of Eq. 12), we added -1 to the array. This helps us keep our non-negativity constraint for all variables.

1.4 A Simplex Iteration

Adapted from Sec. 5 of last time's text.

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:

$$\max z(\vec{x}) = (2,3,0,0,0)(x_1,x_2,x_3,x_4,x_5)^\intercal \tag{17}$$

subject to

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 & 1 \end{pmatrix} \vec{x} = \begin{pmatrix} 6 \\ 10 \\ 4 \end{pmatrix} \tag{18}$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0. \tag{19}$$

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

$$\begin{pmatrix} x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 6 \\ 10 \\ 4 \end{pmatrix} - t \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} \tag{20}$$

from which we get

$$t \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix} \leq \begin{pmatrix} 6 \\ 10 \\ 4 \end{pmatrix} \tag{21}$$

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

$$A = \begin{pmatrix} 2 & 1 & 2 & -1 & 0 & 0 \\ 1 & 0 & -1 & 2 & 1 & 0 \\ 3 & 0 & 3 & 1 & 0 & 1 \end{pmatrix}. \tag{22}$$

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

$$A_B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{23}$$

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:

$$A_B = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ 3 & 0 & 1 \end{pmatrix}. \tag{24}$$

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 $A\vec{x} = \vec{b}$. $\vec{x_a}$ is a basic solution iff:

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

where x_N is the subset of the vector $\vec{x_a}$ of the columns *not* in the basis B.

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:

$$\max\{\vec{c}^\intercal \vec{x} + z_a : A\vec{x} = \vec{b}, \vec{x} \geq \vec{0}\} \tag{25}$$

where 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. 17-19 above:

$$\max z(\vec{x}) = (2,3,0,0,0)(x_1,x_2,x_3,x_4,x_5)^\intercal \tag{26}$$

subject to

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 & 1 \end{pmatrix} \vec{x} = \begin{pmatrix} 6 \\ 10 \\ 4 \end{pmatrix} \tag{27}$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0. \tag{28}$$

Note that in this case, $z_a = 0$.

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:

$$\max z(\vec{x}) = 17 + (0,0,-5/2,0,-1/2)(x_1,x_2,x_3,x_4,x_5)^\intercal \tag{29}$$

subject to

$$\begin{pmatrix} 1 & 0 & 1/2 & 0 & -1/2 \\ 0 & 1 & 1/2 & 0 & 1/2 \\ 0 & 0 & -3/2 & 1 & 1/2 \end{pmatrix} \vec{x} = \begin{pmatrix} 1 \\ 5 \\ 3 \end{pmatrix} \tag{30}$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0. \tag{31}$$

(Notice that now columns 1, 2, 4 form an identity matrix.) So how did we get here? First, we are going to need the *inverse* of a matrix or two, so let's take a quick break and see how to calculate them numerically using Octave or R, then we'll come back to the problem.

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 `*` 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. 29 to 31? There's about 3.5 pages of linear algebra in the textbook...

Fundamentally, take the program (P):

$$\max\{z(\vec{x}) = \vec{c}^{\intercal}\vec{x} + z_a\} \tag{32}$$

subject to

$$A\vec{x} = \vec{b} \tag{33}$$

$$\vec{x} \geq \vec{0} \tag{34}$$

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

$$\max\{z(\vec{x}) = \vec{y}^{\intercal}\vec{b} + z_a + (\vec{c}^{\intercal} - \vec{y}^{\intercal}A)\vec{x}\} \tag{35}$$

subject to

$$A_B^{-1}A\vec{x} = A_B^{-1}\vec{b} \tag{36}$$

$$\vec{x} \geq \vec{0}, \tag{37}$$

where

$$\vec{y} = A_B^{-\intercal}\vec{c_B}. \tag{38}$$

(Note that $A_B^{-\intercal}$ is the transpose of the inverse of the matrix A, and the inverse operation and transpose operation commute, so $A_B^{-\intercal} = (A_B^{-1})^{\intercal} = (A_B^{\intercal})^{-1}$.)

Or, I prefer to write it

$$\max\{z'(\vec{x}) = \vec{c'}^{\intercal}\vec{x} + z'_a\} \tag{39}$$

subject to

$$A'\vec{x} = \vec{b'} \tag{40}$$

$$\vec{x} \geq \vec{0} \tag{41}$$

where

$$\vec{y} = A_B^{-\intercal}\vec{c_B} \tag{42}$$

$$\vec{c'}^{\intercal} = \vec{c}^{\intercal} - \vec{y}^{\intercal}A \tag{43}$$

$$z'_a = \vec{y}^{\intercal}\vec{b} + z_a \tag{44}$$

$$A' = A_B^{-1}A \tag{45}$$

$$\vec{b'} = A_B^{-1}\vec{b}. \tag{46}$$

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. 42.

Apply all of this to Eqs. 26-28 and you will get Eqs. 29-31.

3.4 Additional Example

Due to popular request, in class we went through a second example of creating the canonical form for a different basis. Still the same problem,

$$\max z(\vec{x}) = (2,3,0,0,0)(x_1,x_2,x_3,x_4,x_5)^\intercal \tag{47}$$

subject to

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 & 1 \end{pmatrix} \vec{x} = \begin{pmatrix} 6 \\ 10 \\ 4 \end{pmatrix} \tag{48}$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0. \tag{49}$$

Note that in this case, $z_a = 0$.

As written, this is already in canonical form for $B = \{3,4,5\}$. The requested basis was $B = \{2,4,5\}$, which gives

$$A_B = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \tag{50}$$

and

$$A_N = \begin{pmatrix} 1 & 1 \\ 2 & 0 \\ -1 & 0 \end{pmatrix} \tag{51}$$

We are going to need both A_B^{-1} and $A_B^{-\intercal}$, so let's calculate them:

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

```
AB =

   1   0   0
   1   1   0
   1   0   1

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

   1   0   0
  -1   1   0
  -1  -0   1

octave:4> ABinvT = ABinv'
ABinvT =

   1  -1  -1
   0   1  -0
   0   0   1
```

(In Octave, the single quote mark, or apostrophe, gives you the transpose of a matrix or vector. This is not to be confused with the "prime" mark $'$ that I put on variables as I update them to new values.)

In Eq. 45, we saw that the rewritten constraint equation uses $A' = A_B^{-1}A$ and $\vec{b'} = A_B^{-1}\vec{b}$, so let's calculate those:

```
octave:5> Aprime = ABinv*A
ans =

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

octave:6> b = [6;10;4]
b =

    6
   10
    4

octave:7> bprime = ABinv*b
ans =

   6
   4
  -2
```

That gives us our constraints. Now, to rewrite the objective function is a little more complicated. Recall $\vec{c_B}$ is the subset of basis elements from the objective function coefficients $\vec{c}$ corresponding to our desired basis B, so in this case, elements 2, 4 and 5 of $(2, 3, 0, 0, 0)$, then we calculate $\vec{y} = A_B^{-\intercal}$,

```
octave:8> CB = [3; 0; 0]
CB =

   3
   0
   0


octave:9> y = ABinvT*CB
y =

   3
   0
   0
```

Now we can calculate the new $c' = c^\intercal - y^\intercal A$,

```
octave:12> c = [2; 3; 0; 0; 0]
c =

   2
   3
   0
   0
   0


octave:23> cprime = c' - y'*A
cprime =

  -1   0  -3   0   0
```

Finally, calculate $z'_a = \vec{y}^\intercal \vec{b} + z_a$,

```
octave:26> y'*b
ans =  18
```

giving us (hopefully) the revised linear program

$$\max\{z'(\vec{x}) = \vec{c'}^{\mathsf{T}}\vec{x} + z'_a\} \tag{52}$$

subject to

$$A'\vec{x} = \vec{b'} \tag{53}$$

$$\vec{x} \geq \vec{0} \tag{54}$$

where

$$A' = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 & 0 \\ -2 & 0 & -1 & 0 & 1 \end{pmatrix} \tag{55}$$

$$\vec{b'} = \begin{pmatrix} 6 \\ 4 \\ -2 \end{pmatrix} \tag{56}$$

$$\vec{c'} = \begin{pmatrix} -1 \\ 0 \\ -3 \\ 0 \\ 0 \end{pmatrix} \tag{57}$$

$$z'_a = 18 \tag{58}$$

If all has gone well, that has changed our problem statement from one that was canonical in the 3, 4, 5 basis to one that is canonical in the basis 2, 4, 5. Confirming our conditions: columns 2, 4, and 5 form an identity matrix ($A'_B = I$) (check!); the corresponding entries in our objective function are zero ($\vec{c'_B} = \vec{0}$) (check!).

Whew! It worked.

3.5 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

$$\vec{x_B} = A_B^{-1}\vec{b}. \tag{59}$$

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 the n-dimensional triangle, 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.

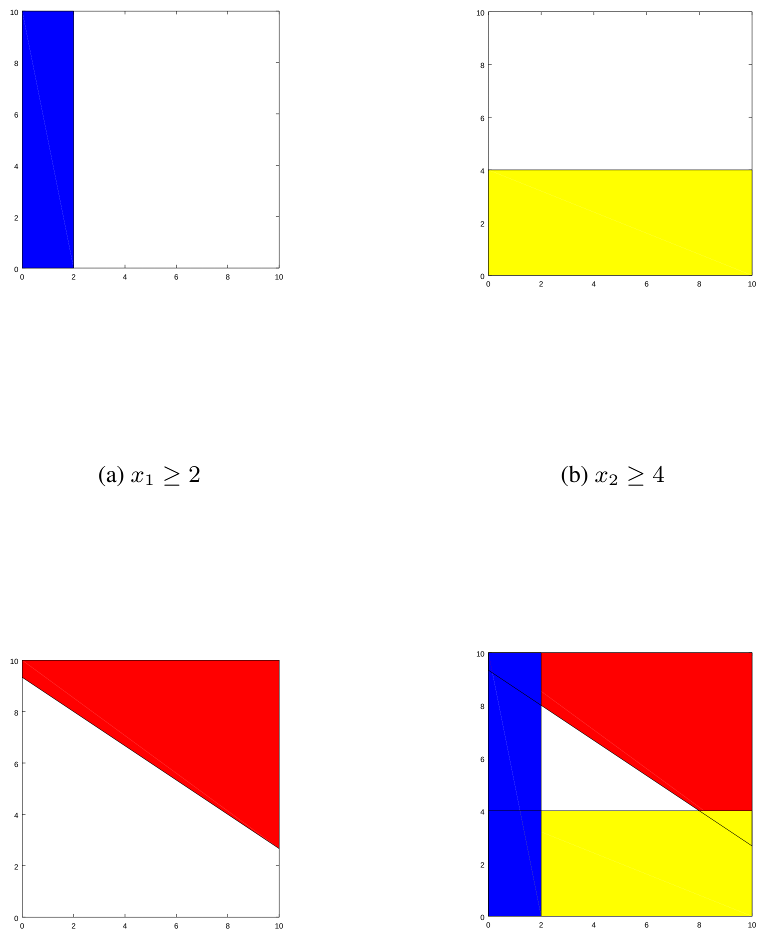
6 References for this Lecture

1. `https://www.gnu.org/software/octave/doc/interpreter/Three_002dDimensional-Pl` has examples of the Octave function `patch`.

2. `https://www.gnu.org/software/octave/doc/interpreter/Graphics-Object-Properti`

3. `https://jp.mathworks.com/help/matlab/ref/patch.html` is help in Japanese on using the `patch` function in Octave.

4. I recommend that you avoid learning more about finding the basic solution for right now; it's really messy. But if you want to, I found the video `https://www.youtube.com/watch?v=eY` helpful.



(a) $x_1 \geq 2$ (b) $x_2 \geq 4$

(c) $2x_1 + 3x_2 \leq 28$ (d) All three constraints.

Figure 1: The constraints in our simplex example. The three constraints collectively define a triangle of acceptable solutions.

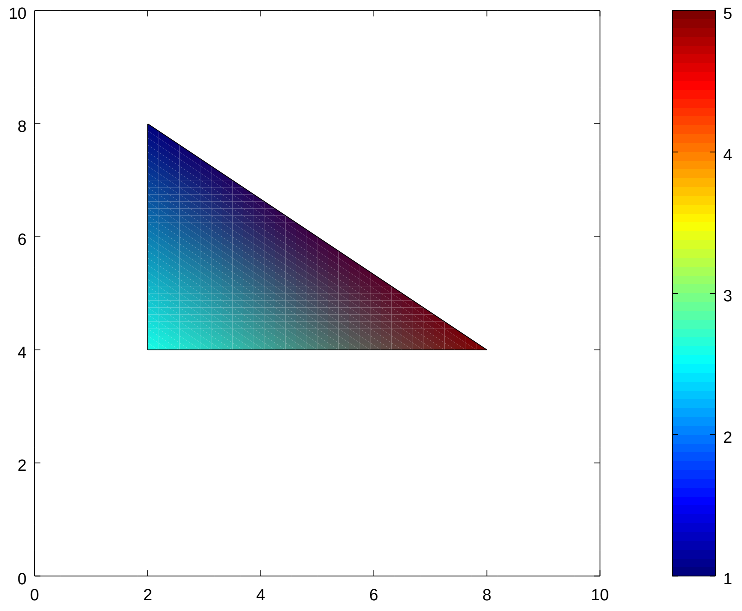


Figure 2: The objective function $z(\vec{x}) = \frac{19}{14}x_1 - \frac{1}{28}x_2$ in our 2-D simplex, bounded by the constraints above. For linear programs (problems), the optimal point always lies at a vertex (corner).