Linear Programming
The Simplex Algorithm: Part II
Chapter 5

University of Chicago
Booth School of Business
Kipp Martin

October 17, 2017
Outline

List of Files

Key Concepts

Revised Simplex

Revised Simplex with LU Decomposition

Other Issues
  Finding a Starting Basis
  Unbounded Problems
  Convergence
  The Future
List of Files

- revisedsimplex.m
- revisedsimplexlu.m
- lpptest1.m
- lpptest2.m
- lpptest3.m
- artificial.m
- unbounded.m
Key Concepts

▶ nonzero elements are the enemy

▶ density is our enemy

▶ zeros are our friend

▶ sparsity is our friend

▶ there is a big difference between toy textbook problems and problems that need to get solved in practice

▶ deja vu – nonzero elements are the enemy – we will encounter a number of other enemies throughout the quarter
Key Concepts

- revised Simplex
- revised Simplex with LU decomposition
- finding an initial basis
- unbounded linear programs
- degeneracy
Revised Simplex

In basicsimplex.m we partitioned the $A$ matrix into basic and nonbasic columns

$$A_B x_B + A_N x_N = b$$

And then multiplied by the basis inverse

$$A_B^{-1} A_B x_B + A_B^{-1} A_N x_N = A_B^{-1} b$$
$$x_B + A_B^{-1} A_N x_N = A_B^{-1} b$$

Here is the problem – even if $A_B$ and $A_N$ are sparse, $A_B^{-1} A_N$ is often dense. Multiplying $A_N$ by $A_B^{-1}$ often results in a very dense matrix.
Revised Simplex

The non-basic columns of a covering problem before pivoting.
Revised Simplex

The non-basic columns after pivoting – note the nonzero fill in.
**Bottom Line:** No one ever, ever works with matrix $A_B^{-1}A_N$ in practice. It is not practical. So we really have two problems or dragons to slay.

- we need the updated columns of $A_N$ (that is $A_B^{-1}A_N$) in order to do the minimum ratio test,

- we need $A_B^{-1}$ to find the updated columns, the updated right-hand side, and reduced costs.
Revised Simplex

In terms of basicsimplex.m we do the following:

\[ T = \text{inv}(AB) \ast A; \]
\[ b\overline{B} = \text{inv}(AB) \ast b; \]

which is necessary for the minimum ratio test

\[ \text{if } b\overline{B}(k) / T(k, N(\text{negRedCostIdx})) < \text{minRatioVal} \]

We also calculate

\[ w = c' - (cB' \ast \text{inv}(AB)) \ast A; \]

which is necessary to determine the pivot column

\[ \text{if } wN(k) < -0.0001 \]

so in general, \( T \) is \textbf{VERY} dense.
Revised Simplex

So let’s slay these two dragons one at a time. First, we eliminate the need to explicitly keep $T$ around. We store two things:

- The original $A$ matrix
- The basis inverse (for now)
Revised Simplex

**Storing $A$:** most real problems are actually quite sparse – that is the ratio of nonzero elements in $A$ to the number of elements in $A$ is quite small.

The vast majority of elements are zero. Consider for example a transportation problem.

$$A = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}$$

Zeros are very good friends. We never need to use them in arithmetic operations such as plus or times so we don’t bother to keep them around.
Revised Simplex

**Sparse Matrix Storage:** It is necessary to understand how a matrix is stored in practice.

Remember that real problems are **sparse**! Never store or operate on the **zero elements**.

Store the A matrix (constraint matrix) with three arrays (vectors):

- **values** – an array that holds the nonzero elements in the A matrix, the size is equal to the number of nonzero elements in the matrix
- **indexes** – an array that holds the row (column) indexes of the nonzero elements in the A matrix, the size is equal to the number of nonzero elements in the matrix
- **starts** – a pointer array that points the start of each column (row), the size is equal to the number of columns (rows) plus one in the matrix;
Revised Simplex

Column Major Storage:

Consider the following matrix.

\[
A = \begin{bmatrix}
0 & 1.5 & 0 & 7.7 & 0 \\
2.0 & 6.7 & 1.1 & 7.7 & 0 \\
7.5 & 0 & 1.1 & 10 & 1 \\
0 & 0 & 1.1 & 12 & 0 \\
3.1 & 0 & 0 & 1.9 & 0
\end{bmatrix}
\]

Then

\[
\text{values} = 2.0, 7.5, 3.1|1.5, 6.7|1.1, 1.1, 1.1|7.7, 7.7, 10, 12, 1.9|1
\]

\[
\text{indexes} = 1, 2, 4|0, 1|1, 2, 3|0, 1, 2, 3, 4|2
\]

\[
\text{starts} = 0, 3, 5, 8, 13, 14
\]

Note: we use 0-based counting for the array indexes
Revised Simplex

**Key Idea:** In revised simplex, we only store $A$ and only update the column we pivot on – none of the other columns! See `revisedsimplex.m`

```
aBAR = inv( AB)*AN(:,negRedCostIdx);
minRatioVal = inf;
for k = 1:numRows
    if aBAR( k) > 0
        if bBAR(k) / aBAR(k) < minRatioVal
            minRatioVal = bBAR(k) / aBAR( k);
            %record the index
            minRatioIdx = k;
        end
    end
end
```

The Revised Simplex Algorithm

Here is a formal mathematical statement of the Revised Simplex Algorithm.

**Step 1: (Initialization)** Initialize with a basis $B$ corresponding to a basic feasible solution and for the corresponding basis matrix calculate initial values of the right-hand-side vector

$$\bar{x}_{Bi} = A_B^{-1} b$$

**Step 2: (Pivot Column Selection)** Calculate the reduced costs

$$\bar{w}_N = c_N^\top - c_B^\top A_B^{-1} A_N$$

If $\bar{w}_N \geq 0$, stop, the current solution is optimal; else, select $q \in N$ such that $\bar{w}_q < 0$. 
The Revised Simplex Algorithm

How do we do the multiplication $c_B^\top A_B^{-1} A_N$ in practice?

Matrix multiplication is associative.

$$(c_B^\top A_B^{-1}) A_N = c_B^\top (A_B^{-1} A_N)$$

Which do we prefer?

Why?
The Revised Simplex Algorithm

Step 3: (Minimum Ratio Test and Pivot Row Selection)
Perform the minimum ratio test on column $q$ that was selected in Step 2. First find the updated column $q$.

$$
\bar{a}^q = A^{-1}_B a^q
$$

$$
i^* \leftarrow \text{argmin} \{ \bar{x}_{Bi} / \bar{a}_{iq} | \bar{a}_{iq} > 0, \ i = 1, \ldots, m \}
$$

Step 4: (Update the right-hand-side)

$$
\bar{x}_{Bi} \leftarrow \bar{x}_{Bi} - (\bar{x}_{Bi^*} / \bar{a}_{i^*q}) \bar{a}^q
$$

$$
\bar{x}_{Bi^*} \leftarrow \bar{x}_{Bi^*} / \bar{a}_{i^*q}
$$
Step 5: (Update the Basic and Non-Basic Index Sets)

\[ B \leftarrow (B \setminus \{B_i^*\}) \cup \{q\} \quad N \leftarrow (N \setminus \{q\}) \cup \{B_i^*\} \]

Go to Step 2.
The Revised Simplex Algorithm

In Step 2 we calculate the reduced costs by

\[ \bar{w}_N = c_N^\top - (c_B^\top A_B^{-1})A_N \]

not

\[ \bar{w}_N = c_N^\top - c_B^\top (A_B^{-1} A_N) \]
Revised Simplex

The MATLAB program, revisedsimplex.m, implements the revised Simplex algorithm as just described. (We assume a min problem in standard form.)

Inputs:

- $A$ – the constraint matrix
- $b$ – the right-hand-side vector
- $c$ – the objective function vector
- $B$ – an index set of the basic variables (feasible)

Outputs:

- $X$ – the optimal solution to the linear program
- $objVal$ – the optimal objective function value
- $B$ – the optimal basis
Revised Simplex

A problem in standard form:

\[
A = \begin{bmatrix}
.7 & 1 & 1 & 0 & 0 & 0 \\
.5 & 5/6 & 0 & 1 & 0 & 0 \\
1 & 2/3 & 0 & 0 & 1 & 0 \\
1/10 & 1/4 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\[
c = [-10; -9; 0; 0; 0; 0 ]
\]

\[
b = [630; 600; 708; 135]
\]

\[
B = [3 4 5 6]
\]

Make the call

\[
[X, objVal, B] = 	ext{revisedsimplex}( A, b, c, B)
\]
Revised Simplex with LU Decomposition

Okay, so with revised simplex we avoid working with the updated tableau,

\[ x_B + A_B^{-1}A_Nx_N = A_B^{-1}b \]

Time to slay the second dragon – we don’t want to explicitly have to calculate \( A_B^{-1} \) because it is too dense.

There are two places in the iterative process of revised simplex that we need \( A_B^{-1} \).

- In **Step 2** we need to calculate \( c_B^\top A_B^{-1} \)
- In **Step 3** we need to calculate \( \bar{a}^q = A_B^{-1}a^q \)
Revised Simplex with LU Decomposition

We avoid calculating $A_B^{-1}$ by using $LU$ decomposition.

**Reduced Cost Calculation:** We need to calculate

\[
\begin{align*}
\top u &= c_B A_B^{-1} \\
\top u A_B &= c_B \\
(\top u A_B)^\top &= (c_B^\top)^\top \\
A_B^\top u &= c_B
\end{align*}
\]

In other words solve the system

\[A_B^\top u = c_B\]
Revised Simplex with LU Decomposition

Solve the system $A_B^T u = c_B$ with $LU$ decomposition.

We know there exists a lower triangular matrix $L$, with diagonal elements of 1, and upper triangular matrix $U$, and permutation matrix $P$ such that

$$PA_B^T = LU$$

We solve

$$LUu = Pc_B$$

Let $y = Uu$. Do a forward solve on $Ly = Pc_B$ to obtain $y$ and then do a backward solve on $Uu = y$. 
Revised Simplex with LU Decomposition

Assume that the original $A$ matrix is

$$
A = \begin{bmatrix}
0.7 & 1 & 1 & 0 & 0 & 0 \\
0.5 & 5/6 & 0 & 1 & 0 & 0 \\
1 & 2/3 & 0 & 0 & 1 & 0 \\
1/10 & 1/4 & 0 & 0 & 0 & 1
\end{bmatrix}
$$

A basis corresponding to a basic feasible solution is

$$
B = [3, 4, 5, 2]
$$

\[c^T = \begin{bmatrix}
-10 & -9 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}\]

Set up a system of equations to find the reduced costs.
Revised Simplex with LU Decomposition

```
>> [L, U, P] = mylu( AB')

L =

1.0000  0   0   0
0   1.0000  0   0
0   0   1.0000  0
1.0000  0.8333 0.6667  1.0000

U =

1.0000  0   0   0
0   1.0000  0   0
0   0   1.0000  0
0   0   0   0.2500

P = I
```
Revised Simplex with LU Decomposition

We are solving

\[ LUu = P_{CB} \]

We set \( y = Uu \) and a forward solve on \( Ly = P_{CB} \)

\[ y_1 = 0 \]
\[ y_2 = 0 \]
\[ y_3 = 0 \]

\[ y_1 + \frac{5}{6}y_2 + \frac{2}{3}y_3 + y_4 = -9 \]
Revised Simplex with LU Decomposition

Do a backward solve on $Uu = y$

\[
\begin{align*}
  u_1 &= y_1 \\
  u_2 &= y_2 \\
  u_3 &= y_3 \\
  \frac{1}{4}u_1 &= y_4
\end{align*}
\]
Revised Simplex with LU Decomposition

This is implemented in revisedsimplexlu.m. In this code we do

\[
\begin{align*}
\% \text{ reduced cost calculations} \\
u &= \text{lusolve}(AB', \ cB); \\
wN &= cN' - u'AN;
\end{align*}
\]

Pretty easy!
Revised Simplex with LU Decomposition

We also need, $\bar{a}^q = A_B^{-1}a^q$. That is, we need to solve

\[
PA_B\bar{a}^q = Pa^q
\]

\[
LU\bar{a}^q = Pa^q
\]

Let $y = U\bar{a}^q$.

Do a forward solve on $Ly = Pa^q$ to obtain $y$ and then do a backward solve on $U\bar{a}^q = y$.

In the revisedsimplexlu.m code this is implemented as

\[
aBAR = lusolve(AB, AN(:,negRedCostIdx) );
\]
Revised Simplex with LU Decomposition

Calculate the updated column for variable 1 for use in the minimum ratio test.

>> [L, U, P] = mylu( AB)
L =

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

U =

1.0000 0 0 1.0000
0 1.0000 0 0.8333
0 0 1.0000 0.6667
0 0 0 0.2500
Revised Simplex with LU Decomposition

We solve \( LU\bar{a}^q = Pa^q \).

Do a forward solve on \( Ly = a^1 \).

\[
\begin{align*}
y_1 &= .7 \\
y_2 &= .5 \\
y_3 &= 1 \\
y_4 &= .1
\end{align*}
\]
Revised Simplex with LU Decomposition

Do a backward solve on \( Uu = y \)

\[
\begin{align*}
    u_1 + u_4 &= y_1 \\
    u_2 + \left(\frac{5}{6}\right)u_4 &= y_2 \\
    u_3 + \left(\frac{2}{3}\right)u_4 &= y_3 \\
    u_4 &= y_4
\end{align*}
\]
Revised Simplex

The MATLAB program, *revisedsimplexlu.m*, implements the revised Simplex algorithm as just described with $LU$ decomposition. (We assume a min problem in standard form.)

**Inputs:**

- $A$ – the constraint matrix
- $b$ – the right-hand-side vector
- $c$ – the objective function vector
- $B$ – an index set of the basic variables (feasible)

**Outputs:**

- $X$ – the optimal solution to the linear program
- $objVal$ – the optimal objective function value
- $B$ – the optimal basis
Revised Simplex

A problem in standard form:

\[
A = \begin{bmatrix}
.7 & 1 & 1 & 0 & 0 & 0 \\
.5 & 5/6 & 0 & 1 & 0 & 0 \\
1 & 2/3 & 0 & 0 & 1 & 0 \\
1/10 & 1/4 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]

\[
c = [-10; -9; 0; 0; 0; 0 ]
\]

\[
b = [630; 600; 708; 135]
\]

\[
B = [3 4 5 6]
\]

Make the call

\[
[X, objVal, B] = revisedsimplexlu( A, b, c, B)
\]
Revised Simplex with LU Decomposition

I actually have hidden a lot of detail from you.

In the MATLAB code revisedsimplexlu.m, at every iteration, we do a **brand new** $LU$ decomposition of the basis matrix.

However, from iteration-to-iteration the basis matrix changes by only one column. Hence an $LU$ decomposition from scratch is a huge waste!!!!

An incredible amount of research has gone into doing sparse $LU$ updates. We will not get into this.

If you understand what is in these slides you have a pretty good feeling for what is happening with real-world codes.
Finding a Starting Basis

The MATLAB simplex routines require as input, \( B \), which indexes a basic feasible solution. For some problems it is easy to find a starting basic feasible solution. The linear program

\[
\begin{align*}
\text{max} & \quad 10x_1 + 9x_2 \\
& \quad .7x_1 + x_2 \leq 630 \\
& \quad .5x_1 + (5/6)x_2 \leq 600 \\
& \quad x_1 + (2/3)x_2 \leq 708 \\
& \quad .1x_1 + .25x_2 \leq 135 \\
& \quad x_1, x_2 \geq 0
\end{align*}
\]

in standard form is

\[
\begin{align*}
\text{min} & \quad -10x_1 - 9x_2 \\
& \quad .7x_1 + x_2 + x_3 = 630 \\
& \quad .5x_1 + (5/6)x_2 + x_4 = 600 \\
& \quad x_1 + (2/3)x_2 + x_5 = 708 \\
& \quad .1x_1 + .25x_2 + x_6 = 135 \\
& \quad x_1, x_2, x_3, x_4, x_5, x_6 \geq 0
\end{align*}
\]
Finding a Starting Basis

Given the standard form

\[
\begin{align*}
\text{min} & \quad -10x_1 - 9x_2 \\
.7x_1 & + x_2 + x_3 = 630 \\
.5x_1 & + (5/6)x_2 + x_4 = 600 \\
x_1 & + (2/3)x_2 + x_5 = 708 \\
.1x_1 & + .25x_2 + x_6 = 135 \\
x_1, x_2, x_3, x_4, x_5, x_6 & \geq 0
\end{align*}
\]

a trivial basic feasible solution is indexed by

\[B = [3 \ 4 \ 5 \ 6]\]

Indeed, if the original linear program has all constraints of the form \(Ax \leq b\) and \(b \geq 0\), then we convert to standard form by adding a nonnegative slack variable to each row and putting the slack variable in the basis. The nonnegativity of \(b\) guarantees that this is a basic feasible solution.
Finding a Starting Basis

For problems with more general structure, finding a basic feasible solution by inspection is not so easy. Fortunately, there is a systematic procedure for finding a starting basic feasible solution, but it requires solving a linear program. (See Section 6.5 of the text).

First convert the original linear program to standard form.

$$\begin{align*}
\text{min } & c^T x \\
\text{s.t. } & Ax = b \\
& x \geq 0
\end{align*}$$

Then convert the standard form to a Phase I problem.

$$\begin{align*}
\text{min } & 0x + e^T a \\
\text{s.t. } & Ax + la = b \\
& x, \ a \geq 0
\end{align*}$$
Finding a Starting Basis

Converting to Phase I:

- Append to the constraint matrix $A$ an $m \times m$ identity matrix where $m$ is the number of constraints. Thus, the new constraint matrix is $[A \ I]$.
- Add $m$ new variables to the problem. These are called artificial variables. The coefficient of the new variables correspond to the identity matrix added.
- Give the original variables a cost of zero in the objective function.
- Give each artificial variable a cost of 1.0 in the objective function.

Result: If the simplex algorithm is applied to the Phase I problem for a feasible and bounded linear program, then it will find a basic feasible solution with no artificial variables and have optimal value equal to zero.
Finding a Starting Basis

**Phase I Example:** Consider the following linear program in standard form:

\[
\begin{align*}
\text{MIN} &\quad 2x_1 + 11x_2 + 7x_3 + 7x_4 + 20x_5 + \\
&\quad 2x_6 + 5x_7 + 5x_8 \\
\text{SUBJECT TO} &\quad x_1 + x_2 = 1 \\
&\quad x_3 + x_4 = 1 \\
&\quad x_5 + x_6 = 1 \\
&\quad x_7 + x_8 = 1 \\
&\quad 3x_1 + 6x_3 + 5x_5 + 7x_7 + x_9 = 13 \\
&\quad 2x_2 + 4x_4 + 10x_6 + 4x_8 + x_{10} = 10
\end{align*}
\]

Convert to a Phase I linear program.
Finding a Starting Basis

**Phase I Example:** The Phase I version of this problem is:

\[
\begin{align*}
\text{MIN} & \quad A_1 + A_2 + A_3 + A_4 + A_5 + A_6 \\
\text{SUBJECT TO} & \\
X_1 + X_2 + A_1 &= 1 \\
X_3 + X_4 + A_2 &= 1 \\
X_5 + X_6 + A_3 &= 1 \\
X_7 + X_8 + A_4 &= 1 \\
3 \ X_1 + 6 \ X_3 + 5 \ X_5 + 7 \ X_7 + X_9 + A_5 &= 13 \\
2 \ X_2 + 4 \ X_4 + 10 \ X_6 + 4 \ X_8 + X_{10} + A_6 &= 10
\end{align*}
\]
Finding a Starting Basis

**Phase I Example:** The problem in MATLAB, artificial.m is

\[
A = \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
3 & 0 & 6 & 0 & 5 & 0 & 7 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 2 & 0 & 4 & 0 & 10 & 0 & 4 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
b = \begin{bmatrix} 1 ; 1 ; 1 ; 1 ; 13 ; 10 \end{bmatrix};
\]

\[
c = \begin{bmatrix} 0 ; 0 ; 0 ; 0 ; 0 ; 0 ; 0 ; 0 ; 0 ; 1 ; 1 ; 1 ; 1 ; 1 \end{bmatrix};
\]

\[
B = \begin{bmatrix} 11 & 12 & 13 & 14 & 15 & 16 \end{bmatrix};
\]

\[
[X, \text{objVal}, B] = \text{revisedsimplexlu}(A, b, c, B)
\]
Finding a Starting Basis

**Phase 1 Example:** The MATLAB solution is

```matlab
objVal = 0
B = 1 3 6 5 7 4
```
Unbounded Problems

Recall that a linear program in standard form can be rewritten as

\[ z_0 = c_B^\top A_B^{-1} b + \bar{w}_N^\top x_N \]
\[ x_B = A_B^{-1} b - A_B^{-1} A_N x_N \]
\[ x_B, x_N \geq 0 \]

What if there is a variable \( k \in N \) such that?

- \( \bar{w}_k < 0 \)
- \( \bar{a}^k = A_B^{-1} a^k \leq 0 \)
Unbounded Problems

We are unbounded!!! We currently are at the point \( x^1 = [x_B \ x_N] \) with objective function value \( c^T A^{-1}_B b \). Move to the new point:

\[
    x^2 = x^1 + \lambda r
\]

for positive \( \lambda \) where component \( k \) of the \( r \) vector is 1, and the other nonzero components are given by \( \bar{a}^k \).

This is tough. Let’s illustrate.
Consider the following linear program in standard form.

\[
\begin{array}{llllll}
\text{min} & -2x_1 & -3x_2 & x_2 & -x_3 & = 1 \\
& x_1 & +x_2 & -x_4 & = 2 \\
& -0.5x_1 & +x_2 & +x_5 & = 8 \\
& -x_1 & +x_2 & +x_6 & = 6 \\
& x_1, & x_2, & x_3, & x_4, & x_5, & x_6 \geq 0
\end{array}
\]

The associated MATLAB file is unbounded.m.
Unbounded Problems

The MATLAB file is unbounded.m. defines the following:

% the constraint matrix
A = [0 1 -1 0 0 0
     1 1 0 -1 0 0
     -.5 1 0 0 1 0
     -1 1 0 0 0 1]

% the objective function
c = [-2; -3; 0; 0; 0; 0 ]

% the right-hand-side
b = [1; 2; 8; 6]

% Initial Basis
B = [1 2 5 6]
Unbounded Problems

The associated MATLAB file is unbounded.m.

```matlab
[tableau, B] = simplexpivot( A, b, c, B)
```

<table>
<thead>
<tr>
<th>-0.00</th>
<th>-0.00</th>
<th>1.00</th>
<th>2.00</th>
<th>-0.00</th>
<th>-0.00</th>
<th>-5.00</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>0.00</td>
<td>1.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1.50</td>
<td>-0.50</td>
<td>1.00</td>
<td>0.00</td>
<td>7.50</td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>2.00</td>
<td>-1.00</td>
<td>0.00</td>
<td>1.00</td>
<td>6.00</td>
</tr>
</tbody>
</table>
```

Variable \( x_4 \) has a reduced cost of \(-2\). What is the effect of bringing this variable into the basis?
Unbounded Problems

If variable $x_4$ pivots in at any positive value, no other variable pivots out. The current solution is

$$x^0 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 7.5 \\ 6.0 \end{bmatrix}$$

If we increase $x_4$ by $\lambda$ the new solution is

$$x = x^0 + \lambda r = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 7.5 \\ 6.0 \end{bmatrix} + \lambda \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ .5 \\ 1.0 \end{bmatrix}$$
Unbounded Problems

The vector

\[
\begin{bmatrix}
1 \\
0 \\
0 \\
1 \\
0.5 \\
1.0
\end{bmatrix}
\]

is called an extreme ray or extreme direction of the linear program. I can take any positive multiple of this, add it to a feasible solution, and still be feasible.

Since \( c^\top r = -2 \),

\[
c^\top (x^0 + \lambda r) = c^\top x^0 + \lambda c^\top r = -5 - 2\lambda
\]
Unbounded Problems

Experiment a bit with `unbounded.m` to get a feeling for what is going on.

\[ x_0 = [1; 1; 0; 0; 7.5; 6.0] \]
\[ r = [1; 0; 0; 1; .5; 1] \]

\[ \lambda = 100 \]
\% new solution
\[ x = x_0 + \lambda * r \]

\% new objective function value
\[ \text{objVal} = c' * x \]

\% should be feasible, i.e.
\% this should evaluate to 0
\[ A * x - b \]
Here is the geometry again.
Unbounded Problems

Every point (working in \((x_1, x_2)\) space) in the feasible region can be written as:

\[
x = \lambda_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \lambda_2 \begin{bmatrix} 0 \\ 2 \end{bmatrix} + \lambda_3 \begin{bmatrix} 0 \\ 6 \end{bmatrix} + \lambda_4 \begin{bmatrix} 4 \\ 10 \end{bmatrix} + \lambda_5 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \lambda_6 \begin{bmatrix} 2 \\ 1 \end{bmatrix}
\]

\[
\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1
\]

\[
\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6 \geq 0
\]

Observations:

- The feasible region is unbounded.
- How did I find the extreme ray corresponding to variable \(\lambda_6\)?
In practice, what does an unbounded problem tell you?
Why does the Simplex algorithm terminate?

- Every basic feasible solution corresponds to an extreme point.
- There are a finite number of extreme points.
- Assuming no degenerate pivots, the objective function value strictly decreases at every iteration. Hence, we cannot repeat extreme points. Therefore, we terminate with a finite number of iterations.

Remember – if there is an optimal solution to the problem, there is an optimal extreme point solution. A linear program can fail to have an optimal solution if and only if it is either infeasible or unbounded.
Convergence

Observations:

1. The change in the objective function value for a pivot on column \( k \) in row \( i \) is

\[
\bar{w}_k \left( \frac{\bar{b}_i}{\bar{a}_{ik}} \right)
\]

2. Since \( \bar{w}_k < 0, \bar{a}_{ik} > 0, \) and \( \bar{b}_i \geq 0, \) this change can only be zero if \( \bar{b}_i = 0. \)

A pivot on row \( i \) when \( \bar{b}_i = 0 \) is called a degenerate pivot. It is theoretically possible for the Simplex algorithm to cycle in the presence of degeneracy. Cycling is not a problem in practice.

However, degeneracy is a real problem in that it can really slow things down. Commercial codes have pivot selection rules that will prevent cycling and reduce the number of degenerate pivots.
The Future – Pricing

Here is where we are headed. Right now we explicitly price every column when we make the calculation

**Step 2: (Pivot Column Selection)** Calculate the reduced costs

\[
\overline{w}_N = c_N^\top - (c_B^\top A_B^{-1})A_N
\]

For many important applications this is not practical:

- there are so many columns in \( A_N \) it would require too much memory and time to price
- it is not even practical to enumerate all of the columns in \( A_N \)

We will do the pricing operation by solving another optimization problem or by implementing an algorithm.
The Future – Interior Point

The Simplex algorithm moves from extreme point to extreme point on the boundary of the feasible region.

At the end of the quarter we will study interior point algorithms that move through the interior of the feasible region.

We will apply interior point algorithms to both linear and non-linear programs.

Even for linear programs, interior point algorithms are based on ideas from nonlinear programming so we need to cover nonlinear programming in order to understand how interior point algorithms work.