ORF 307: Lecture 4
Linear Programming: Chapter 3
Degeneracy

Robert Vanderbei
February 12, 2015

Slides last edited on February 24, 2015
maximize \[ 2x_1 + 3x_2 \]
subject to \[ x_1 + 2x_2 \leq 2 \]
\[ x_1 - x_2 \leq 1 \]
\[ -x_1 + x_2 \leq 1 \]
\[ x_1, x_2 \geq 0. \]
Solution

Note: The horizontal axis, which one might call the $x_1$-axis, is where $x_2 = 0$ and is labeled as such.

In $(x_1, x_2)$ coordinates, the pivots visit the following vertices:

$(0, 0) \implies (0, 1) \implies (0, 1) \implies (4/3, 1/3)$.

Note that the second pivot went nowhere.
Degeneracy

Definitions.

A dictionary is degenerate if one or more “rhs”-value vanishes.

Example:

\[
\begin{align*}
\zeta &= 6 + w_3 + 5x_2 + 4w_1 \\
x_3 &= 1 - 2w_3 - 2x_2 + 3w_1 \\
w_2 &= 4 + w_3 + x_2 - 3w_1 \\
x_1 &= 3 - 2w_3 \\
w_4 &= 2 + w_3 - w_1 \\
w_5 &= 0 - x_2 + w_1
\end{align*}
\]

A pivot is degenerate if the objective function value does not change.

Examples (based on above dictionary):

1. If \( x_2 \) enters, then \( w_5 \) must leave, pivot is degenerate.
2. If \( w_1 \) enters, then \( w_2 \) must leave, pivot is \textit{not} degenerate.
Cycling

A *cycle* is a sequence of pivots that returns to the dictionary from which the cycle began.

Note: Every pivot in a cycle must be degenerate. Why?

Pivot Rules

A *pivot rule* is an explicit statement for how one chooses entering and leaving variables (when a choice exists).

Some Examples:

*Largest-Coefficient Rule.* (most common pivot rule for entering variable)

Choose the variable with the largest coefficient in the objective function.

*Random Positive-Coefficient Rule.*

Among all nonbasic variables having a positive coefficient, choose one at random.

*First Encountered Rule.*

In scanning the nonbasic variables, stop with the first one whose coefficient is positive.
Some pivot rule, such as the largest coefficient rule, will be proven never to cycle.
Some pivot rule, such as the largest coefficient rule, will be proven never to cycle.

An example that cycles using the following pivot rules:

- entering variable: largest-coefficient rule.
- leaving variable: smallest-index rule.

\[
\begin{align*}
\zeta &= x_1 - 2x_2 - 2x_4 \\
 w_1 &= -0.5x_1 + 3.5x_2 + 2x_3 - 4x_4 \\
w_2 &= -0.5x_1 + x_2 + 0.5x_3 - 0.5x_4 \\
w_3 &= 1 - x_1.
\end{align*}
\]

Here’s a demo of cycling (ignoring the last constraint)...
### Current Dictionary

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>obj</strong> =</td>
<td>0</td>
<td>+</td>
<td><strong>1</strong></td>
<td>x1</td>
<td>+</td>
<td><strong>-2</strong></td>
<td>x2</td>
<td>+</td>
</tr>
<tr>
<td><strong>w1</strong> =</td>
<td>0</td>
<td>-</td>
<td><strong>1/2</strong></td>
<td>x1</td>
<td>-</td>
<td><strong>-7/2</strong></td>
<td>x2</td>
<td>-</td>
</tr>
<tr>
<td><strong>w2</strong> =</td>
<td>0</td>
<td>-</td>
<td><strong>1/2</strong></td>
<td>x1</td>
<td>-</td>
<td><strong>-1</strong></td>
<td>x2</td>
<td>-</td>
</tr>
</tbody>
</table>

### Current Dictionary

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>obj</strong> =</td>
<td>0</td>
<td>+</td>
<td><strong>-2</strong></td>
<td>w1</td>
<td>+</td>
<td><strong>5</strong></td>
<td>x2</td>
<td>+</td>
</tr>
<tr>
<td><strong>x1</strong> =</td>
<td>0</td>
<td>-</td>
<td><strong>2</strong></td>
<td>w1</td>
<td>-</td>
<td><strong>-7</strong></td>
<td>x2</td>
<td>-</td>
</tr>
<tr>
<td><strong>w2</strong> =</td>
<td>0</td>
<td>-</td>
<td><strong>-1</strong></td>
<td>w1</td>
<td>-</td>
<td><strong>5/2</strong></td>
<td>x2</td>
<td>-</td>
</tr>
</tbody>
</table>

### Current Dictionary

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>obj</strong> =</td>
<td>0</td>
<td>+</td>
<td><strong>0</strong></td>
<td>w1</td>
<td>+</td>
<td><strong>-2</strong></td>
<td>w2</td>
<td>+</td>
</tr>
<tr>
<td><strong>x1</strong> =</td>
<td>0</td>
<td>-</td>
<td><strong>-4/5</strong></td>
<td>w1</td>
<td>-</td>
<td><strong>14/5</strong></td>
<td>w2</td>
<td>-</td>
</tr>
<tr>
<td><strong>x2</strong> =</td>
<td>0</td>
<td>-</td>
<td><strong>-2/5</strong></td>
<td>w1</td>
<td>-</td>
<td><strong>2/5</strong></td>
<td>w2</td>
<td>-</td>
</tr>
</tbody>
</table>

### Current Dictionary

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>obj</strong> =</td>
<td>0</td>
<td>+</td>
<td><strong>4</strong></td>
<td>w1</td>
<td>+</td>
<td><strong>-16</strong></td>
<td>w2</td>
<td>+</td>
</tr>
<tr>
<td><strong>x3</strong> =</td>
<td>0</td>
<td>-</td>
<td><strong>-4</strong></td>
<td>w1</td>
<td>-</td>
<td><strong>14</strong></td>
<td>w2</td>
<td>-</td>
</tr>
<tr>
<td><strong>x2</strong> =</td>
<td>0</td>
<td>-</td>
<td><strong>2</strong></td>
<td>w1</td>
<td>-</td>
<td><strong>-8</strong></td>
<td>w2</td>
<td>-</td>
</tr>
</tbody>
</table>
Cycling is rare for small problems! A program that generates random $2 \times 4$ fully degenerate problems was run more than one billion times and did not find one example!

However, for larger problems with lots of zeros, cycling is common and can be a real problem.
Algebra of a Pivot

\[
\begin{array}{cc}
  b & a \\
  \hline
  d & c \\
\end{array}
\quad \rightarrow 
\begin{array}{cc}
  \frac{b}{a} & \frac{1}{a} \\
  \hline
  d - \frac{bc}{a} & \frac{c}{a} \\
\end{array}
\]
m = 2;
n = 4;

numprobs = 1000000000;
stop = 0;
for k = 1:numprobs
    c = randn(1,n);
    A = randn(m,n);
    nonbasics = 1:n;
    basics = (n+1:n+m)';

    iter = 1;
    while max(c) > 0,
        [cj, col] = max(c);
        j = nonbasics(col);
        [i, row] = min(basics + (n+m)*(A(:,col) >= -1e-12));
        if i > n+m, break; end % UNBOUNDED POLYTOPE
        Arow = A(row,:);
        Acol = A(:,col);
        a = A(row,col);
        A = A - Acol*Arow/a;
        A(row,:) = -Arow/a;
        A(:,col) = Acol/a;
        A(row,col) = 1/a;
        ccol = c(col);
        c = c - ccol*Arow/a;
        c(col) = ccol/a;

        basics(row) = j;
        nonbasics(col) = i;

        if iter > 15,
            stop = 1; % CYCLING EXAMPLE FOUND
            'breaking'
            end
        iter = iter+1;
    end
if stop == 1, break; end
end
param m := 2;
param n := 4;

param c {1..n};    param A {1..m, 1..n};
param nonbasics {1..n};    param basics {1..m};
param row;    param col;
param ii;    param jj;
param Arow {1..n};    param Acol {1..m};
param cj;    param bi;
param a;    param ccol;
param iter;

for {k in 1..1000000000} {
    let {i in 1..m, j in 1..n} A[i,j] := Normal01();
    let {j in 1..n} c[j] := Normal01();
    let {j in 1..n} nonbasics[j] := j;
    let {i in 1..m} basics[i] := n+i;
    display k;
    let iter := 1;
    repeat while (max {j in 1..n} c[j] > 0) {
        let cj := 0;
        for {j in 1..n} {
            if (c[j] > cj) then {
                let col := j;
                let cj := c[j];
            }
        }
        let jj := nonbasics[col];
        let bi := m+n+1;
        for {i in 1..m: A[i,jj] < -1e-8} {
            if (basics[i] < bi) then {
                let bi := basics[i];
                let row := i;
            }
        }
        if bi > m+n then {break;} # unbounded polytope
        let ii := basics[row];
    }
}

let {j in 1..n} Arow[j] := A[row,j];
let {i in 1..m} Acol[i] := A[i,col];
let a := A[row,col];
let {i in 1..m, j in 1..n} A[i,j] := A[i,j] - Acol[i]*Arow[j]/a;
let {j in 1..n} A[row,j] := -Arow[j]/a;
let {i in 1..m} A[i,col] := Acol[i]/a;
let A[row,col] := 1/a;
let ccol := c[col];
let {j in 1..n} c[j] := c[j] - ccol*Arow[j]/a;
let {j in 1..n} c[j] := c[j] - ccol*Arow[j]/a;
let c[col] := ccol/a;

let basics[row] := jj;
let nonbasics[col] := ii;

if iter > 15 then {
    display "found a cycling example";
    break;
}

let iter := iter+1;
Perturbation Method

Whenever a vanishing “rhs” appears perturb it.
If there are lots of them, say $k$, perturb them all.
Make the perturbations at different scales:

\[
\text{other data} \gg \epsilon_1 \gg \epsilon_2 \gg \cdots \gg \epsilon_k > 0.\]

An Example.

\[
\begin{align*}
\text{Entering variable: } x_2 \\
\text{Leaving variable: } w_2
\end{align*}
\]
Recall current dictionary:

<table>
<thead>
<tr>
<th></th>
<th>obj</th>
<th>v1</th>
<th>x2</th>
<th>v3</th>
</tr>
</thead>
<tbody>
<tr>
<td>entering</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>leaving</td>
<td>4.0</td>
<td>-1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>rearranged</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>v2</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>x1</td>
<td>-2.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>x2</td>
<td>-3.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>obj</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>v1</td>
<td>2.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>x2</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>v3</td>
<td>1.0</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
</tr>
<tr>
<td>w1</td>
<td>-4.0</td>
<td>-1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Entering variable: $x_1$
Leaving variable: $w_3$
**Perturbation Method Applied to Cycling Example**

| obj | 0 | + | 0 | e1 | + | 0 | e2 | + | x1 | + | -2 | x2 | + | 0 | x3 | + | -2 | x4 |
|-----|---|---|---|----|---|---|----|---|----|---|----|----|---|----|---|----|---|
| w1  | 0 | + | 1 | e1 | + | 0 | e2 | - | 1/2 | x1 | - | -1 | x2 | - | -1/2 | x3 | - | 1/2 | x4 |
| w2  | 0 | + | 0 | e1 | + | 1 | e2 | - | 1/2 | x1 | - | -7/2 | x2 | - | -2 | x3 | - | 4 | x4 |

\[ \Downarrow \text{x}_1 \text{ enters, } w_2 \text{ leaves} \]

| obj | 0 | + | 0 | e1 | + | 2 | e2 | + | -2 | w2 | + | 5 | x2 | + | 4 | x3 | + | -10 | x4 |
|-----|---|---|---|----|---|---|----|---|----|----|---|----|----|---|----|---|---|---|
| w1  | 0 | + | 1 | e1 | + | -1 | e2 | - | -1 | w2 | - | 5/2 | x2 | - | 3/2 | x3 | - | -7/2 | x4 |
| x1  | 0 | + | 0 | e1 | + | 2 | e2 | - | 2 | w2 | - | -7 | x2 | - | -4 | x3 | - | 8 | x4 |

\[ \Downarrow \text{x}_2 \text{ enters, } w_1 \text{ leaves} \]

| obj | 0 | + | 2 | e1 | + | 0 | e2 | + | 0 | w2 | + | -2 | w1 | + | 1 | x3 | + | -3 | x4 |
|-----|---|---|---|----|---|---|----|---|----|----|---|----|----|---|----|---|---|---|
| x2  | 0 | + | 2/5 | e1 | + | -2/5 | e2 | - | -2/5 | w2 | - | 2/5 | w1 | - | 3/5 | x3 | - | -7/5 | x4 |
| x1  | 0 | + | 14/5 | e1 | + | -4/5 | e2 | - | -4/5 | w2 | - | 14/5 | w1 | - | 1/5 | x3 | - | -9/5 | x4 |

\[ \Downarrow \text{x}_3 \text{ enters, } x_2 \text{ leaves} \]

| obj | 0 | + | 8/3 | e1 | + | -2/3 | e2 | + | 2/3 | w2 | + | -8/3 | w1 | + | -5/3 | x2 | + | -2/3 | x4 |
|-----|---|---|---|----|---|----|----|---|----|----|---|----|----|---|----|---|---|---|
| x3  | 0 | + | 2/3 | e1 | + | -2/3 | e2 | - | -2/3 | w2 | - | 2/3 | w1 | - | 5/3 | x2 | - | -7/3 | x4 |
| x1  | 0 | + | 8/3 | e1 | + | -2/3 | e2 | - | -2/3 | w2 | - | 8/3 | w1 | - | -1/3 | x2 | - | -4/3 | x4 |

\[ \Downarrow w_2 \text{ enters, problem unbounded!} \]

Note: objective function increases with every pivot: \[ 0 < 2\epsilon_2 < 2\epsilon_1 < \frac{8}{3}\epsilon_1 - \frac{2}{3}\epsilon_2 \]
Other Pivot Rules

**Smallest Index Rule.**

Choose the variable with the smallest index (the \(x\) variables are assumed to be “before” the \(w\) variables).

Note: Also known as *Bland’s rule*.

**Random Selection Rule.**

Select at random from the set of possibilities.

**Greatest Increase Rule.**

Pick the entering/leaving pair so as to maximize the increase of the objective function over all other possibilities.

Note: Too much computation.
Theoretical Results

**Cycling Theorem.** If the simplex method fails to terminate, then it must cycle.

Why?

**Fundamental Theorem of Linear Programming.** For an arbitrary linear program in standard form, the following statements are true:

1. If there is no optimal solution, then the problem is either infeasible or unbounded.
2. If a feasible solution exists, then a basic feasible solution exists.
3. If an optimal solution exists, then a basic optimal solution exists.
maximize \( x_1 + 2x_2 + 3x_3 \)
subject to
\[
\begin{align*}
    x_1 + 2x_3 &\leq 3 \\
    x_2 + 2x_3 &\leq 2 \\
    x_1, x_2, x_3 &\geq 0.
\end{align*}
\]