Linear Programming

Homework 4

Nick Morris

05/02/2016

Question 1

An Example

The example used to visualize the structure of the primal formulation can be found in the file "Q1.txt". This example allowed the dual to be formed via primal-dual conversion rules.

Dual Formulation

The final formulation for the dual is the following:

```
set N;
set M;

param v;
param c[N];
param r[N];
param b[N];
param q[M];
param u[N,M];

var w[N] >= 0;
var f[M] <= 0;
var g[N,M] >= 0;
var h[N,M] <= 0;

maximize obj: sum{i in N}(b[i] * w[i]) + sum{j in M}(q[j] * f[j])
              + sum{i in N, j in M}(v * g[i,j]) + sum{i in N, j in M}(u[i,j] * h[i,j]);
s.t. xN{i in N}: r[i] * w[i] = c[i];
s.t. yNM{i in N, j in M}: w[i] + f[j] + g[i,j] + h[i,j] = 1;
```

Question 2

Part A

TRUE

If the primal has alternative optimal solutions then there must be at least one non-basic variable j in the primal with a reduced cost of zero. The dual basic solution is degenerate if and only if some non-basic variable j in the primal has a reduced cost of zero. The dual basic solution is given by: $c_B B^{-1}$ and if a reduced cost coefficient $\overline{c}_j = 0$ then this implies $\overline{c}_j - c_B B^{-1} A_j = 0$ which indicates that the dual constraint is active, therefore making the dual solution degenerate.

Part B

The dual solution remains the same because a new constraint has not entered the model. The primal constraint remains the same after non-zero scalar multiplication. The basis is unaffected by scalar multiplication of a constraint.

Part C

The dual solution would change because a new constraint has entered the model. The primal constraint has been has been replaced by a different constraint, despite this new constraint being a linear combination of two primal constraints. The basis is affected by vector addition of constraints.

Question 3

Code

The code is in the dsimplx_HW 4.R file provided. I suggest opening this file in notepad ++, go to Language → R → R and this will highlight key words and comments to make reading the code easier. Open up R and then go back to notepad++, highlight the entire script, and then copy and paste it into the "R Console" window in R. This will create the function "dsimplx()" in R.

Strings, Vectors, & Matrices

The function dsimplx() requires you to enter a string, 4 vectors, and a matrix as inputs. The way to write a string in R is with quotes (ie. "min", "max", etc). There are two R commands to know, to create the vectors and matrix. The command to create a vector is "`c(e1, e2, …, en)`" Below is a vector with 7 elements. All row vectors and column vectors should be entered this way, and the code will adjust them into row and column vectors.

```
c(1, 2, 3, 4, 7, 44, 88)
```

The commands to create a matrix are "`rbind(r1, r2, … , rn)`" and "`c(e1, e2, …, en)`". Below is a [3 x 7] matrix. The command rbind() binds vectors by rows, so the first vector below: `c(1, 2, 3, 4, 7, 44, 88)` is the first row vector in the matrix and the third vector below: `c(1, 2, 3, 4, 5, 6, 7)` is the third and last row vector in the matrix.

```
rbind(c(1, 2, 3, 4, 7, 44, 88), c(4, 6, 2, 8, 4, 9, 2), c(1, 2, 3, 4, 5, 6, 7))
```

An Example

Figure 1 below is a maximization problem in canonical form. The variables s1 and s2 are slack variables corresponding to inequality constraints L1 and L2. All inequality constraints must be of the form "<=" before adding slack variables. If necessary, make adjustments on ">=" constraints by multiplying the entire constraint by -1 and flipping the sign to "<=". The variable s3* is an artificial variable corresponding to equality constraint L3. A summary of the adjustments that must be performed is as followed:

1. All inequality constraints must be of the form "<="
2. Each inequality constraint must receive a slack variable to convert it to an equality constraint
3. Each equality constraint must receive an artificial variable

Maximize the Objective Function (P)
$P = 15 x_1 + 10 x_2 + 15 x_3$ subject to
L1: $1.0 x_1 + 1.0 x_2 + 1.0 x_3 + s_1 = 85$
L2: $1.25 x_1 + 0.5 x_2 + 1.0 x_3 + s_2 = 90$
L3: $-0.6 x_1 - 1.0 x_2 - 0.5 x_3 + s_3^* = -51.5$
$x_1 >= 0; x_2 >= 0; x_3 >= 0$
$s_1 >= 0; s_2 >= 0; s_3^* >= 0$

Figure 1: Linear Program in Canonical Form

The inputs for dsimplx() would be built the following way:

```
# vector of objective function coefficients
C = c(15, 10, 15, 0, 0, 0)

# matrix of constraint coefficients
A = rbind(c(1, 1, 1, 1, 0, 0),c(1.25, 0.5, 1, 0, 1, 0),c(-.6, -1, -.5, 0, 0, 1))

# vector of right hand side scalars
b = c(85, 90, -51.5)

# vector of 1's and 0's indicating a variable is or isn't a slack variable
slack = c(0, 0, 0, 1, 1, 0)

# vector of 1's and 0's indicating a variable is or isn't an artificial variable
artificial = c(0, 0, 0, 0, 0, 1)
```

This problem would be solved with dsimplx() in the following way:

```
dsimplx(type = "max", C = C, A = A, b = b, slack = slack, artificial = artificial)
$iterations
[1] 3

$solution
    value
obj  1225
x1     40
x2     10
x3     35
x4      0
x5      0
x6      0

$tableaus
$tableaus$`0`
      [,1]   [,2]  [,3]  [,4] [,5] [,6] [,7]
[1,]   0.0 -15.00 -10.0 -15.0    0    0    0
[2,]  85.0   1.00   1.0   1.0    1    0    0
[3,]  90.0   1.25   0.5   1.0    0    1    0
[4,] -51.5  -0.60  -1.0  -0.5    0    0    1

$tableaus$`1`
             [,1] [,2]       [,3]        [,4] [,5] [,6]       [,7]
[1,] 1287.5000000    0 15.0000000 -2.50000000    0    0 -25.000000
[2,]   -0.8333333    0 -0.6666667  0.16666667    1    0   1.666667
[3,]  -17.2916667    0 -1.5833333 -0.04166667    0    1   2.083333
[4,]   85.8333333    1  1.6666667  0.83333333    0    0  -1.666667
```

```
$tableaus$`2`
       [,1] [,2]  [,3] [,4]  [,5] [,6]          [,7]
[1,] 1275.0    0  5.00    0 15.00    0  7.105427e-15
[2,]   -5.0    0 -4.00    1  6.00    0  1.000000e+01
[3,]  -17.5    0 -1.75    0  0.25    1  2.500000e+00
[4,]   90.0    1  5.00    0 -5.00    0 -1.000000e+01

$tableaus$`3`
     [,1] [,2] [,3] [,4]       [,5]       [,6]      [,7]
[1,] 1225    0    0    0 15.7142857  2.8571429  7.142857
[2,]   35    0    0    1  5.4285714 -2.2857143  4.285714
[3,]   10    0    1    0 -0.1428571 -0.5714286 -1.428571
[4,]   40    1    0    0 -4.2857143  2.8571429 -2.857143
```

The output of the solution above shows that the problem required three dual simplex iterations to be solved and the maximum value of the objective function while respecting the constraints is 1225. The solution is given where {x1 x2 x3} correspond to {x1 x2 x3} as seen in Figure 1, whereas {x4 x5 x6} in the solution correspond to {s1 s2 s3*} as seen in Figure 1. The tableaus section shows a dual simplex tableau for each iteration. The first column of each tableau corresponds to the value of the objective function (ie. the first entry) and the value of each basic variable (ie. every entry except the first entry) at that iteration.

Bonus

Code

The code is in the rsimplx_HW 4.R file provided. I suggest opening this file in notepad ++, go to Language → R → R and this will highlight key words and comments to make reading the code easier. Open up R and then go back to notepad++, highlight the entire script, and then copy and paste it into the "R Console" window in R. This will create the function "rsimplx()" in R.

Refer back to HW 3 to see a step-by-step example of how to use the function "simplx()", because the function "rsimplx()" works the same way.

The Revision

The following code in "rsimplx()" is the key difference between the function "simplx()" and "rsimplx()". Initially the matrix B^{-1} is the zero matrix so when the if-else statement is evaluated for the first iteration, the matrix B^{-1} is computed using the standard inversion technique in R. The next iteration when the if-else statement is evaluated, the matrix B^{-1} is no longer the zero matrix, therefore the new matrix B_{NEW}^{-1} can be computed by constructing a matrix E and performing $E * B^{-1}$.

```
if(all(Binv) == 0)
{
    Binv = solve(B)
} else
{
    d = -db
    e = -d/d[l]
    e[l] = 1/d[l]
    E = diag(1,NROW(Binv),NCOL(Binv))
    E[,l] = e
    Binv = E%*%Binv
}
```