MIE 379 Programming Assignment

You are urged to begin work on this straight away and to try and complete it early. Students who are confident programmers should aim to complete Part Ic and you may find it easiest to consider this immediately without completing Parts Ia and Ib (although credit will be given for these parts).

Part Ia [8 marks]

This part of the assignment can be completed by those students who are *not* confident programmers but students with computing skills should proceed directly to Part Ib (and credit will be given for Part Ia).

You are required to write a set of Matlab functions that implement the Revised Simplex Method starting from a given initial basic feasible solution. The forms that these functions should take are as follows.

```
function [result,z,x,pi,basicvars] = rsm(m,n,c,A,b,basicvars)
% Solves a linear program using the Revised Simplex Method
% Assumes standard computational form
% Starts from the given initial basic feasible solution
% Input:
%   m,n       = number of constraints and variables
%   c         = nx1 cost vector
%   A         = mxn constraint matrix
%   b         = mx1 rhs vector
%   basicvars = 1xm vector of indices of basic variables
% Output:
%   result = 1 if problem optimal, -1 if unbounded
%   z      = objective function value
%   x      = nx1 solution vector
%   pi     = mx1 dual vector
```

The above function will call three other functions

```
function [s,minrc] = findEV(n,c,A,pi,basicvars)
function [r,minratio] = findLV(m,xB,BinvAs)
function [basicvars,cB,B,xB] = update(m,c,A,b,s,r,basicvars,cB,B,xB)
```

The full specifications of these functions are given in Appendix A. You should also remember that you will need functions to calculate $\boldsymbol{\pi}$ and $B^{-1}\mathbf{a}_s$. In this part of the question, your RSM code is expected to do nothing special computationally. You may compute a reduced cost vector for all variables (including basic variables that we know have a reduced cost of zero) if you wish. You can also use the function `inv(B)`, or the \ or `mldivide` operator, at any point in this code as many times as you like. The important goal is to get something working and giving the correct answers. I suggest that you run it on the examples given in class, as well as the test problems given which you can solve using Microsoft Excel Solver, to ensure the procedure is correct.

Part Ib [14 marks]
This part of the assignment can be conducted independently of Part Ic.

You are required to improve your Matlab code from Part Ia and implement a more computationally efficient version of the RSM that updates the inverse of the basis matrix, and the values of the basic variables, using the Gauss-Jordan pivoting procedure described in class. The basis representation should make use of the status vectors `varstatus` and `basicvars`. The forms that these functions should take are as follows.

```
function [result,z,x,pi,basicvars,Binv] = GJrsm(m,n,c,A,b,basicvars,Binv)
% Solves a linear program using RSM with Gauss-Jordon updates
% Assumes standard computational form
% Starts from the given initial basic feasible solution
% Input:
%   m,n       = number of constraints and variables
%   c         = nx1 cost vector
%   A         = mxn constraint matrix
%   b         = mx1 rhs vector
%   basicvars = 1xm vector of indices of basic variables
%   Binv      = mxm basis inverse matrix
% Output:
%   result    = 1 if problem optimal, -1 if unbounded
%   z         = objective function value
%   x         = nx1 solution vector
%   pi        = mx1 dual vector
%   basicvars = 1xm vector of indices of basic variables
%   Binv      = mxm basis inverse matrix
```

The above function will call three other functions

```
function [s,minrc] = GJfindEV(n,c,A,varstatus,pi)
function [r,minratio] = GJfindLV(m,xB,BinvAs)
function [varstatus,basicvars,cB,Binv,xB] =
GJupdate(m,c,s,r,BinvAs,varstatus,basicvars,cB,Binv,xB)
```

The full specifications of these functions are given in Appendix B. You should also remember that you will need functions to calculate $\boldsymbol{\pi}$ and $B^{-1}\mathbf{a}_s$. In this part of the question, your RSM code should not compute the reduced costs of basic variables, nor should it make use of functions such as `inv(B)` or the \ or `mldivide` operator, nor should the update step make use of an elementary matrix.

Part Ic [20 marks]

This part of the computer assignment can be conducted independently of Part Ia and Part Ib if you decide to skip those parts.

You are required to improve your Matlab code from Part Ia or Part Ib or complete this part independently, and implement a version of the RSM that includes a Phase I procedure to find an initial basic feasible solution to the problem, using artificial variables as described in class. The forms that these functions should take are as follows.

```
function [result,z,x,pi] = fullrsm(m,n,c,A,b)
% Solves a linear program using Gauss-Jordon updates
% Assumes standard computational form
% Performs a Phase I procedure starting from an artificial basis
% Input:
%   m,n        = number of constraints and variables
%   c          = nx1 cost vector
%   A          = mxn constraint matrix
%   b          = mx1 rhs vector
% Output:
%   result     = 1 if problem optimal, 0 if infeasible, -1 if unbounded
%   z          = objective function value
%   x          = nx1 solution vector
%   pi         = mx1 dual vector
```

The above function will call three other functions

```
function [s,minrc] = fullfindEV(m,n,c,A,phase1,varstatus,pi)
function [r,minratio] = fullfindLV(m,n,xB,BinvAs,phase1,basicvars)
function [varstatus,basicvars,cB,Binv,xB] =
fullupdate(m,c,s,r,BinvAs,phase1,varstatus,basicvars,cB,Binv,xB)
```

The full specifications of these functions are given in the Appendix C. You should also remember that you will need functions to calculate $\boldsymbol{\pi}$ and $B^{-1}\mathbf{a}_s$. If you decided to skip that part, then the specifications will look more like those from Part 1a. In this part of the question, your RSM code should not explicitly add artificial variables. Instead, `basicvars` should be initialized to be `n+1,…,n+m`, and when you encounter an element of `basicvars` that is greater than `n`, then your code should implicitly simulate the presence of the artificial variable. An example of this is the Extended Leaving Variable Criterion that you have seen in class. In Part Ic you should also avoid calculating the reduced costs as a vector.

Appendix A: Function specifications for Part Ia.

```
function [s,minrc] = findEV(m,n,c,A,pi,basicvars)
% Returns the index of the entering variable and it's reduced cost,
% or returns 0 if no entering variable exists
% Input:
%   m,n       = number of constraints and variables
%   c         = nx1 cost vector
%   A         = mxn constraint matrix
%   pi        = mx1 dual vector
% Output:
%   s     = index of the entering variable
%   minrc = reduced cost of the entering variable

function [r,minratio] = findLV(m,xB,BinvAs)
% Returns the position in the basis of the leaving variable,
% or returns 0 if no leaving variable exists
% Input:
%   m      = number of constraints and variables
%   xB     = mx1 basic variable vector
%   BinvAs = mx1 vector of Binv*As
% Output:
%   r        = position in the basis of the leaving variable
%   minratio = minimum ratio from ratio test

function [basicvars,cB,B,xB] = update(m,c,A,b,s,r,basicvars,cB,B,xB)
% Updates the basis representation.
% Input:
%   m,n       = number of constraints and variables
%   c         = nx1 cost vector
%   A         = mxn constraint matrix
%   b         = mx1 rhs matrix
%   s         = index of entering variable
%   r         = position in the basis of the leaving variable
%   basicvars = 1xm vector of indices of basic variables
%   cB        = mx1 basic cost vector
%   B         = mxm basis matrix
%   xB        = mx1 basic variable vector
% Output:
%   basicvars = 1xm updated basicvars vector
%   cB        = mx1 updated basic cost vector
%   B         = mxm updated basis matrix
%   xB        = mx1 updated basic variable vector
```

Appendix B: Function specifications for Part Ib.

```
function [s,minrc] = GJfindEV(n,c,A,varstatus,pi)
% Returns the index of the entering variable and it's reduced cost,
% or returns 0 if no entering variable exists
% Input:
%   n         = number of variables
%   c         = nx1 cost vector
%   A         = mxn constraint matrix
%   varstatus = 1xn vector, varstatus(i) = position in basis of variable i,
%               or 0 if variable i is nonbasic
% Output:
%   s     = index of the entering variable
%   minrc = reduced cost of the entering variable

function [r,minratio] = GJfindLV(m,xB,BinvAs)
% Returns the position in the basis of the leaving variable,
% or returns 0 if no leaving variable exists
% Input:
%   m      = number of constraints
%   xB     = mx1 basic variable vector
%   BinvAs = mx1 vector of Binv*As
% Output:
%   r        = position in the basis of the leaving variable
%   minratio = minimum ratio from ratio test

function [varstatus,basicvars,cB,Binv,xB] =
GJupdate(m,c,s,r,BinvAs,varstatus,basicvars,cB,Binv,xB)
% Updates the basis representation.
% Input:
%   m         = number of constraints
%   c         = nx1 cost vector
%   s         = index of entering variable
%   r         = position in the basis of the leaving variable
%   BinvAs    = mx1 Binv*As vector
%   varstatus = 1xn vector, varstatus(i) = position in basis of variable i,
%               or 0 if variable i is nonbasic
%   basicvars = 1xm vector of indices of basic variables
%   cB        = mx1 basic cost vector
%   Binv      = mxm basis inverse matrix
%   xB        = mx1 basic variable vector
% Output:
%   varstatus = 1xn updated varstatus vector
%   basicvars = 1xm updated basicvars vector
%   cB        = mx1 updated basic cost vector
%   Binv       = mxm updated basis inverse matrix
%   xB        = mx1 updated basic variable vector
```

Appendix C: Function specifications for Part Ic. Note that these specifications assume that you have completed Part 1b. If you decided to skip that part, then the specifications will look more like those from Part 1a.

```
function [s,minrc] = fullfindEV(n,c,A,varstatus,pi,phase1)
% Returns the index of the entering variable and it's reduced cost,
% or returns 0 if no entering variable exists
% Input:
%   n          = number of variables
%   c          = nx1 cost vector
%   A          = mxn constraint matrix
%   varstatus = 1xn vector, varstatus(i) = position in basis of variable i,
%                or 0 if variable i is nonbasic
%   phase1     = boolean, phase1 = true if Phase 1, or false otherwise
% Output:
%   s     = index of the entering variable
%   minrc = reduced cost of the entering variable

function [r,minratio] = fullfindLV(m,xB,BinvAs,phase1,basicvars)
% Returns the position in the basis of the leaving variable,
% or returns 0 if no leaving variable exists
% Input:
%   m          = number of constraints
%   xB         = mx1 basic variable vector
%   BinvAs     = mx1 vector of Binv*As
%   phase1     = boolean, phase1 = true if Phase 1, or false otherwise
%   basicvars = 1xm vector of indices of basic variables
% Output:
%   r         = position in the basis of the leaving variable
%   minratio = minimum ratio from ratio test

function [varstatus,basicvars,cB,Binv,xB] =
fullupdate(m,c,s,r,BinvAs,phase1,varstatus,basicvars,cB,Binv,xB)
% Updates the basis representation.
% Input:
%   m          = number of constraints
%   c          = nx1 cost vector
%   s          = index of entering variable
%   r          = position in the basis of the leaving variable
%   BinvAs     = mx1 Binv*As vector
%   phase1     = boolean, phase1 = true if Phase 1, or false otherwise
%   varstatus = 1xn vector, varstatus(i) = position in basis of variable i,
%                or 0 if variable i is nonbasic
%   basicvars = 1xm vector of indices of basic variables
%   cB         = mx1 basic cost vector
%   Binv       = mxm basis inverse matrix
%   xB         = mx1 basic variable vector
% Output:
%   varstatus = 1xn updated varstatus vector
%   basicvars = 1xm updated basicvars vector
%   cB         = mx1 updated basic cost vector
%   Binv        = mxm updated basis inverse matrix
%   xB         = mx1 updated basic variable vector
```