Math 415 - Intro to OR - Homework 5

The following homework is to be turned in on Canvas by 5 pm on Monday, November 7. For this assignment, if you submit your code early I will grade it and give you feedback so that you can resubmit for a higher grade. You may resubmit as many times as you wish until the final deadline on November 7.

In this assignment you will implement the revised simplex method with upper bounds and use it to solve a number of test problems.

I have supplied a set of 15 test problems ranging from tiny problems that will be helpful in initially debugging your code to bigger problems that will test your code's ability to solve real world LPs. These test scripts have the names test1.m, test2.m, ..., test15.m. Sample output for these test scripts is in test1.out, test2.out, ..., test15.out. I will test your code using these tests as well as other tests. You should definitely test your code with all of these tests before submitting it.

To simplify the assignment, I've written a high level routine, solvelp.m, that takes care of scaling the LP and removing redundant constraints. This routine calls the actual simplex.m function to do the two-phase simplex method. There is also a script, solvescript.m, which can be used to load in one of the NETLIB test problems and solve it with solvelp.

I have also supplied a collection of useful utility routines including routines for solving $Bx = b$ and $y^T B = c_B^T$, performing the ratio test to find the leaving basic variable, scaling problems, and updating the basis.

Your implementation of the simplex method must include the following features

1. The code should use appropriate tolerances ϵ_1 for the reduced costs, ϵ_2 for nonnegativity, and ϵ_3 for nonzero pivots.For now, use the values of 1.0×10^{-5} given below.

2. Your code should not explicitly compute the inverse of B, but should instead use the LU factorization to solve systems of equations.

3. Your code should use sparse matrices where possible.

4. Your code must handle the situation where an auxiliary variable appears in the optimal basis in phase I. These variables must be removed from the basis before phase II.

Your simplex code will have the following calling sequence. I've supplied simplex.m with this already filled in.

```
%
% [x,optobj,optbasis,nonbasis0,nonbasisu,totaliters,ray,y,w,z]=simplex(A,b,c,u,const,max
%
% Solves an LP of the form
%
% min c'*x+const
%     Ax=b
%      x >= 0
%      x <= u          (upper bounds may be +Inf)
%
% by the two-phase simplex method.
%
% Inputs:
%   A,b,c,u,const     Problem data.
%   maxiters          Maximum number of iterations, defaults to
%                     200*max(m,n/10)
%   printlevel        if 0, output nothing at all.
%                     if >0, output information every printlevel
%                     iterations and at the start and end of each phase.
%                     e.g. "400: obj=27.2" if obj is 27.2 at the
%                     completion of the 400th iteration of phase I.
%                     Defaults to 0.
%
% Outputs:
%  1. If the LP has an optimal BFS,
%     x              optimal solution.
%     optobj         optimal objective value.
%     optbasis       row vector listing basic variables.
%     nonbasis0      row vector of variables nonbasic at 0.
%     nonbasisu      row vector of variables nonbasic at u.
%     totaliter      total phase I and phase II iterations.
%     ray            zeros(n,1)
%     y              optimal dual solution as a column vector
%     w              optimal dual solution as a column vector
%     z              optimal dual solution as a column vector
%
%  2. If the LP is infeasible,
%     x              NaN*ones(n,1)
%     optobj         NaN
%     optbasis       []
%     nonbasis0      []
%     nonbasisu      []
%     totaliters     Phase I iterations.
%     ray            NaN*ones(n,1)
%     y              NaN*ones(m,1)
%     w              NaN*ones(n,1)
%     z              NaN*ones(n,1)
%
%  3. If the LP is unbounded,
%     x              Last BFS.
%     optobj         -Inf
%     optbasis       Last basis
%     nonbasis0      Last nonbasis0
%     nonbasisu      Last nonbasisu
%     totaliters     total phase I and phase II iterations.
%     ray            optimal ray.  c*(x+t*ray) -> -infinity as
%                    t->infinity, A*(x+t*ray)=b, 0<=x+t*ray<=u.
%     y              NaN*ones(m,1)
%     w              NaN*ones(n,1)
%     z              NaN*ones(n,1)
%
%  4. If max iterations exceeded in either phase, exit with
%     error('max iterations exceeded')
%
function [x,optobj,optbasis,nonbasis0,nonbasisu,totaliters,ray,y,w,z]=simplex(A,b,c,u,co
%
% Zero tolerances.
%
epsilon1=1.0e-5;
epsilon2=1.0e-5;
epsilon3=1.0e-5;
%
% Get basic problem size data.
%
[m,n]=size(A);
%
% Set default parameters if needed.
%
if ((nargin < 6) | isempty(maxiters))
  maxiters=200*max(m,n/10);
end
if ((nargin < 7) | isempty(printlevel))
  printlevel=0;
end
%
% Set default parameters if needed.
%
if ((nargin < 6) | isempty(maxiters))
  maxiters=10*m;
```

```
end
if ((nargin < 7) | isempty(printlevel))
  printlevel=0;
end
```