Homework Set #8, Math 451/551

1. Solve the following system in three ways: First, just use matlab (this way you'll know the solution which will help you troubleshoot the other two ways. Second, use the basic Gaussian elimination. Lastly, do Gaussian elimination with scaled (partial) pivoting, first by hand, and then implement it as a code.

$$\begin{bmatrix} 0.2641 & 0.1735 & 0.8642 \\ 0.9411 & 0.0175 & 0.1463 \\ -0.8641 & -0.4243 & 0.0711 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -0.7521 \\ 0.6310 \\ 0.2501 \end{bmatrix}$$

2. Count how many short operations (additions and subtractions) and long operations (multiplications and divisions) are involved in running the code that appears below. Replace $n = 4$ with n large. You should get that the code has an operation count of $\mathcal{O}(n^3)$.

3. Code up SOR and Jacobi (Gauss Seidel is SOR with 1 for the relaxation parameter). The point is to compare the number of iterations required of each of the 3 methods to reach a specified error tolerance. Use

```
U = randn(N);
A = U*U'+150*eye(N);
X = randn(N,1);
B=A*X
```

   Hence, you know the solution X to this problem. For small N the condition number of this problem should be order 1, but it will get large if N is large.

   (a) Compute the condition number, the time and iteration count for each method, to reach an absolute error (infinity norm) of 10^{-12}.

   (b) Plot the error as a function of the iteration count in a semi log plot.

   (c) Try small $N = 10, 50, 100, 200$, and report what happens.

   (d) Try to find the optimal SOR relaxation parameter (the eigenvalues of the matrix will help here. Have matlab estimate these).

The following is a MATLAB implementation of the basic Gaussian elimination algorithm for solving the linear system of equations $Ax = b$.

```
% Gauss_np.m
% Gaussian Elimination without pivoting.

  clear;
  format short;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 0: Assign the matrix A and the vector b
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  n = 4;
  A = [ 6,  -2, 2,   4;
       12,  -8, 6,  10;
        3, -13, 9,   3;
       -6,   4, 1, -18 ];
  b = [ 0.23; 0.32; 0.33; 0.31 ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 1: Basic Gaussian Elimination.
% The matrix A is overwritten to store its LU factorization.
%   L is unit lower triangular:
%      L(i,j) = 0 for j>i; L(i,i) = 1;  L(i,j) = A(i,j) for i>j.
%   U is upper triangular:
%      U(i,j) = A(i,j) for j>=i; U(i,j) = 0 for i>j.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  for k = 1:(n-1)
    for i = (k+1):n
        A(i,k) = A(i,k) / A(k,k);
      for j = (k+1):n
        A(i,j) = A(i,j) - A(i,k) * A(k,j);
      end
    end
  end

% output the LU factorization of the original matrix A.
  A

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 2: Forward substitution to solve  L * y = b where
% L(i,j) = 0 for j>i; L(i,i) = 1;  L(i,j) = A(i,j) for i>j.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
  y = zeros(n,1);                % Initialize y to be a column vector
                                 % of length n with zeros entries.
  y(1) = b(1);
  for i = 2:n
      y(i) = b(i);
    for j = 1:(i-1)
      y(i) = y(i) - A(i,j)*y(j);
    end
  end

% output y (the intermediate solution):
  y

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 3: Back substitution to solve  U * x = y
% U(i,j) = A(i,j) for j>=i; U(i,j) = 0 for i>j.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  x = zeros(n,1);                % Initialize x to be a column vector
                                 % of length n with zeros entries.
  x(n) = y(n) / A(n,n);
  for i = (n-1):-1:1
      x(i) = y(i);
    for j = (i+1):n
      x(i) = x(i) - A(i,j)*x(j);
    end
      x(i) = x(i) / A(i,i);
  end

% output x (the final solution):
  x
```