MATH 437 *Homework #4*

1.(a) Show that the basic iteration process given by

$$Qx^{(k+1)} = (Q - A)x^{(k)} + b$$

is equivalent to the following:

$$x^{(k+1)} = x^{(k)} + z^{(k)},$$

where $z^{(k)}$ satisfies the equation $Qz^{(k)} = r^{(k)}$ with $r^{(k)} = b - Ax^{(k)}$.

(b). Using the notation in (a), show that

$$r^{(k+1)} = (I - AQ^{-1})r^{(k)}, \quad z^{(k+1)} = (I - Q^{-1}A)z^{(k)}.$$

2. Programming: please print out your results together with your code.

Consider solving the linear system $Ax = b$ where A is a sparse matrix. Dealing with sparse matrices efficiently involves avoiding computations involving the zero entries. To do this, the matrix must be stored in a scheme which only involves the nonzero entries. We shall use a modified Compressed Sparse Row (CSR) structure. We refer to http://www.netlib.org/utk/people/JackDongarra/etemplates/node373.html for a discussion. This structure is designed so that it is easy to access the entries in a row. Our modification is made so that it is also easy to access the diagonal entry in any row.

The CSR structure involves three arrays: val, col_ind and row_ptr. val is an array of real numbers and stores the actual (nonzero) entries of A. col_ind is an integer array which contains the column indices for nonzero entries in A. The length of val and col_ind are equal to the number of nonzero entries in A. Finally, row_ptr is an integer array of dimension $n+1$ and contains the row offsets (into the arrays val and col_ind). By convention, row_ptr(n+1) is set to the total number of nonzeroes plus one. For example, consider

$$A = \begin{pmatrix} 2 & -1 & 0 & 0 \\ -1/3 & 3 & -2/3 & 0 \\ 0 & -1/4 & 4 & -3/4 \\ 0 & 0 & -1/5 & 5 \end{pmatrix}.$$

The modified CSR structure is as follows:

val	2	-1	3	$-1/3$	$-2/3$	4	$-1/4$	$-3/4$	5	$-1/5$
col_ind	1	2	2	1	3	3	2	4	4	3
pow_ptr	1	3	6	9	11					

Note that the i's entry of row_ptr points to the start of the nonzero values (in val) for the i's row. It also points to the start of the column indices for that row. The modification is that we always put the diagonal entry at that location, i.e. val(row_ptr(i)) $= A_{ii}$. The general CSR structure does not do this. Indeed, the general CSR storage does not have a diagonal entry whenever the diagonal entry is zero.

(a). Given a positive integer N, consider the following N by N matrix:

$$A = N^2 \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}.$$

Create a m-file CreateSystemMatrix.m to generate the above matrix in the modified CRS format. The code should be like

```
function A = CreateSystemMatrix(N)
        % create the struct of A
        A = struct('val',[],'col_ind',[],'row_ptr',[]);
        % assign number of nonzero entries
        num_non_zeros = ???;
        % initialize arrays
        A.val = zeros(num_non_zeros,1);
        A.col_ind = zeros(num_non_zeros,1);
        A.row_ptr = zeros(N+1,1);
        % put nonzero entries from the first row into A
        ns = N^2;
        A.val(1) = 2*ns; A.val(2) = -ns;
        A.col_ind(1) = 1; A.col_ind(2) = 2;
        A.row_ptr(1) = 1;
        % put nonzero entries from the second row to (N-1)th
        % row into A
        for i = 2:N-1
                ???
        end
        % put nonzero entries from the last row into A
        ???
        A.row_ptr(N+1) = num_non_zeros + 1;
end
```

Now we are in a position to solve the linear system using iterative methods with the CRS format. An example to use the CRS format is the matrix-vector multiplication. Let src and dst be the input and output arrays, respectively. We need to compute

$$\mathsf{dst}(i) = \sum_{j=1}^{N} A_{ij} * \mathsf{src}(j), \qquad \text{for } i = 1, 2, \ldots, N.$$

Here is the Matlab code:

```
function dst = vmult(A, src)
        num_rows = length(src);
        dst = zeros(num_rows,1);
        for i=1:num_rows
          dst(i) = 0;
          for j = A.row_ptr(i) : (A.row_ptr(i+1)-1)
```

```
                dst(i) = dst(i) + A.val(j) * src(A.col_ind(j));
            end
        end
end
```

Here we note the the loop for j from A.row_ptr(i) to A.row_ptr(i+1)-1 gives the access to the nonzero entries in the i row.

(b). Create an m-file JacobiIteration.m which does one step of the Jacobi iteration given the CRS format of A and the right hand side vector b. Let src and dst be the input and output arrays, respectively. Recall the iteration

$$D * \mathsf{dst} = D * \mathsf{src} + (b - A * \mathsf{src}) = b - (A - D) * \mathsf{src},$$

where D is the diagonal part of A (i.e. $A_{ii} = A.\mathsf{val}(\mathsf{row\_ptr}(\mathsf{i}))$ with the row index i). So we need to update dst by

$$\mathsf{dst}(i) = (b(i) - \sum_{j \neq i} A_{ij} * \mathsf{src}(j))/A_{ii}, \qquad \text{for } i = 1, 2, \ldots, N.$$

The code should look like:

```
function dst = JacobiIteration(A,b,src)
        num_rows = length(src);
        for i=1:num_rows
          ??? %update dst with Jacobi algorithm
        end
end
```

(c). Create an m-file GaussSeidelIteration.m which does one step of the Gauss-Seidel iteration given the CRS format of A and the right hand side vector b. Let src and dst be the input and output arrays, respectively. Recall the iteration

$$(D + L) * \mathsf{dst} = -U * \mathsf{src} + b,$$

where L and U is the lower and upper triangular part of A. We solve dst using forward substitution, i.e. initialize dst with src and compute

$$\mathsf{dst}(i) = (b(i) - \sum_{j \neq i} A_{ij} * \mathsf{dst}(j))/A_{ii}, \qquad \text{for } i = 1, 2, \ldots, N$$

(Why?). The code should look like:

```
function dst = GaussSeidelIteration(A,b,src)
        num_rows = length(src);
        for i=1:num_rows
          ??? %update dst with Gauss-Seidel algorithm
        end
end
```

(d). Write a driver routine to solve the system $Ax = b$ with Jacobi and Gauss-Seidel methods, where A is given in part (a) with $N = 4, 8, 16, 32, 64$ and $b = (1, 1, \ldots, 1)$. Set the initial vector x_0 to be the zero vector and stop the iteration when $\|Ax - b\|_2 < 10^{-12}$ (use $\mathsf{norm}(\mathsf{vmult}(A, x) - b)$ to compute the l^2 norm in MATLAB). Report the number of iterations as a function of N (i.e. a table with values of N in the first column and #iter in the second column).

(e) (Bonus) Use the function `vmult` to create a Conjugate Gradient (CG) subroutine and solve the above linear system with CG. Report the number of iterations as a function of N.