COL 726 Homework 3

Due date: Friday, 1 March, 2019

Questions 1–5 are worth 3 marks each. Question 6 is worth 5 marks.

1. Suppose $\mathbf{A}$ is an $m \times m$ "banded" matrix, i.e. a matrix whose entries a_{ij} are nonzero only if $-l \leq j - i \leq u$ for some constants l and u. For example, a banded matrix with $l = 2, u = 1$ is shown on the right.

$$\begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ \times & \times & \times & \ddots & \\ & \ddots & \ddots & \ddots & \times \\ & & \times & \times & \times \end{bmatrix}$$

   (a) Give an algorithm for computing the LU decomposition of $\mathbf{A}$ without pivoting in $O(lum)$ flops. Count the number of flops it takes, to leading order in m.

   (b) The banded structure may not be maintained if pivoting is performed. Find a 5×5 matrix $\mathbf{A}$ with $l = u = 1$ such that, after LU decomposition with partial pivoting, the factor $\mathbf{L}$ has a nonzero value in its bottom left entry. Give all the matrices $\mathbf{P}, \mathbf{L}, \mathbf{U}$ in the factorization.

2. Suppose I have already have the Cholesky factorization $\mathbf{A} = \mathbf{L}\mathbf{L}^T$ of an $m \times m$ SPD matrix $\mathbf{A}$. Now I enlarge $\mathbf{A}$ to an $(m+1) \times (m+1)$ matrix $\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{b}^T & c \end{bmatrix}$, where $\mathbf{b} \in \mathbb{R}^m$ and $c \in \mathbb{R}$. How can the Cholesky factorization of $\mathbf{M}$ be computed in $O(m^2)$ time? Give a mathematical justification as well as all the steps of the final algorithm.

3. Let $\mathbf{A}\mathbf{x} = \mathbf{b}$ be an $m \times m$ system of equations. Consider the following algorithm:

   choose a guess for the values $x_1, x_2, \ldots, x_m$
   repeat
       for $k \leftarrow 1, \ldots, m$ **do**
           solve equation k to update variable k, i.e. $x_k \leftarrow \frac{1}{a_{kk}}(b_k - \sum_{j \neq k} a_{kj} x_j)$
       end for
   until convergence

   (a) Express one complete execution of the **for** loop as a formula for the new guess $\mathbf{x}^{(n+1)}$ in terms of the old guess $\mathbf{x}^{(n)}$. What iterative method does this perform?

   (b) Suppose I choose two permutations of the set $\{1, \ldots, m\}$, namely $(p_1, \ldots, p_m)$ and $(q_1, \ldots, q_m)$. At the kth step of the **for** loop, I solve equation p_k to update variable q_k. How can this method be expressed in similar terms as (a)?

4. Consider an $m \times m$ matrix $\mathbf{A}$ with all eigenvalues real, distinct, and nonzero. Suppose $\mathbf{b}$ lies in the span of only n eigenvectors of $\mathbf{A}$, where $n < m$. Show that the Arnoldi iteration "breaks down" in at most n steps, i.e. $\mathbf{A}\mathbf{q}_k$ lies in the previous Krylov subspace $\mathcal{K}_k = \langle \mathbf{q}_1, \ldots, \mathbf{q}_k \rangle$ for some $k \leq n$. Then, show that GMRES can find the solution $\mathbf{x}_*$ to the equation $\mathbf{A}\mathbf{x} = \mathbf{b}$ even in this case.

5. Let $\mathbf{A}$ be a symmetric positive definite matrix with $\|\mathbf{A} - \mathbf{I}\|_2 = 0.6$.

   (a) Prove that all eigenvalues of $\mathbf{A}$ lie in the interval $[0.4, 1.6]$. Consequently, give an upper bound on the relative error norm $\|\mathbf{e}_n\|_\mathbf{A} / \|\mathbf{e}_0\|_\mathbf{A}$ after n iterations of conjugate gradients on a linear system $\mathbf{Ax} = \mathbf{b}$.

(b) Suppose $\mathbf{A}$ has an eigenvalue $\lambda_1 = 1$ with an associated unit eigenvector $\mathbf{v}_1$, and the remaining eigenvalues are $\lambda_2, \ldots, \lambda_m$. Let $\mathbf{B} = \mathbf{A} + \mathbf{ww}^T$ where $\mathbf{w} = 7\mathbf{v}_1$. Verify that $\mathbf{B}$ has the same eigenvectors as $\mathbf{A}$, and find all its eigenvalues. (Note: $\lambda_1, \ldots, \lambda_m$ are not in any sorted order.)

(c) Consider the conjugate gradient method applied to a linear system $\mathbf{Bx} = \mathbf{y}$. Give an upper bound on $\|\mathbf{e}_n\|_\mathbf{B}/\|\mathbf{e}_0\|_\mathbf{B}$ after n iterations. Your answer should depend only on n, and when evaluated at $n = 2$ should result in a number less than 0.8.

6. Sparse matrices often arise in the analysis of networks. Here, we will consider electrical networks of nodes connected by resistors.

Suppose you are given a network of m nodes and $O(m)$ resistors as a list of tuples of the form (i, j, R_{ij}), indicating that nodes i and j are connected by a resistance R_{ij}. Also assume that the first and last nodes are connected via unit resistors to a voltage source $V = 1$ and ground $V = 0$ respectively. Some example networks can be constructed using the function `makeNetwork` provided on the course webpage; make sure to read its comments for more details.

(a) The net outgoing current from any node i is given by

$$I_i = \left(\sum_{j \text{ connected to } i} \frac{V_i - V_j}{R_{ij}} \right) + I_i^{\text{out}},$$

where I_i^{out} is $-(1 - V_i)$ for the first node, V_i for the last node, and 0 otherwise. The network is solved by finding the unknown node voltages $\mathbf{v} = [\ldots, V_i, \ldots]^T$ such that all net currents $\mathbf{i} = [\ldots, I_i, \ldots]^T$ are zero. Find a way to express $\mathbf{i}$ in the form $\mathbf{i} = \mathbf{Av} + \mathbf{b}$, where the matrix $\mathbf{A}$ and vector $\mathbf{b}$ depend only on the network and not on $\mathbf{v}$. In your report, define the entries a_{ij} of $\mathbf{A}$ and show that the matrix is symmetric. In your program, write a function `applyA(network, v)` that maps $\mathbf{v}$ to $\mathbf{Av}$ in $O(m)$ time, and a function `getB(network)` that returns $\mathbf{b}$.

(b) Implement a function `cg(Afun, b, tolerance)` that performs conjugate gradient iterations to solve a linear system $\mathbf{Ax} = \mathbf{b}$. The first argument of `cg` should be a function such that `Afun`$(\mathbf{x}) = \mathbf{Ax}$. Terminate the iterations when $\|\mathbf{r}_n\|_2/\|\mathbf{b}\|_2 \leq$ tolerance, and return the final iterate $\mathbf{x}_n$ and the history of residual norms $[\|\mathbf{r}_0\|_2/\|\mathbf{b}\|_2, \ldots, \|\mathbf{r}_n\|_2/\|\mathbf{b}\|_2]$. Then, you should be able to solve $\mathbf{i} = \mathbf{Av} + \mathbf{b} = \mathbf{0}$ for a network by calling `cg(lambda v: applyA(network, v), -getB(network), tolerance)`. Test it out on `makeNetwork('wheatstone')`.

(c) Implement a function `getDiag(network)` that returns the diagonal of $\mathbf{A}$ as a vector $\mathbf{d}$. Then implement a function `pcg(Afun, b, d, tolerance)` that performs conjugate gradients with symmetric preconditioning using the preconditioner $\mathbf{M} = \text{diag}(\mathbf{d})$. Your function `pcg` should not perform any iterations itself, just call `cg` with a modified `Afun` and a modified $\mathbf{b}$.

Try running `cg` on `makeNetwork('random1', 1000)` with tolerance 10^{-6}. Visualize the convergence of the method by plotting $\|\mathbf{r}_n\|_2/\|\mathbf{b}\|_2$ on a log scale as a function of n, and include the plot in your report. Then run `cg` and `pcg` on `makeNetwork('random2', 1000)` with tolerance 10^{-6}, plot the convergence of both methods on the same plot, and include it as well.