COL 726 Homework 3

14–28 February, 2020

1. Consider a matrix partitioned as $\mathbf{M} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$, where the diagonal blocks $\mathbf{A}$ and $\mathbf{D}$ are square and $\mathbf{A}$ is invertible.
   (a) Find a "block LU" factorization $\mathbf{M} = \begin{bmatrix} \mathbf{L}_{11} & \\ \mathbf{L}_{21} & \mathbf{L}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ & \mathbf{U}_{22} \end{bmatrix}$.
   (b) Consequently, prove that $\mathbf{M}$ is invertible if and only if $\mathbf{S} = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}$ is invertible, and give its inverse $\mathbf{M}^{-1}$ in that case.

2. Suppose $\mathbf{A} \in \mathbb{C}^{m \times m}$ is a Hermitian positive definite matrix whose Cholesky factorization $\mathbf{A} = \mathbf{L}\mathbf{L}^*$ is known. You are now given an arbitrary vector $\mathbf{v} \in \mathbb{C}^m$, and you wish to compute the Cholesky factorization of $\mathbf{A} + \mathbf{v}\mathbf{v}^*$. Find an $O(m^2)$ algorithm to do so, and prove that it always works.

   Hint: Try finding the first column of the new Cholesky factor. Is it now possible to proceed recursively on an $(m-1) \times (m-1)$ problem?

3. (a) Suppose we want to solve the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A} = \begin{bmatrix} 2 & -1 & \\ -1 & 2 & -1 \\ & -1 & 2 \end{bmatrix}$ and $\mathbf{b}$ is an arbitrary vector. Give an upper bound on $\|\mathbf{e}_k\|/\|\mathbf{e}_0\|$ after k iterations of the Jacobi method. Similarly, bound the error after k iterations of the Gauss-Seidel method.
   (b) One method for over-relaxation is $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega(\mathbf{x}^{\mathrm{GS}} - \mathbf{x}^{(k)})$, where $\mathbf{x}^{\mathrm{GS}}$ is the result of a Gauss-Seidel step. However, a more effective strategy is to apply over-relaxation immediately when updating each component of the vector, $x_i^{(k+1)} = x_i^{(k)} + \omega(x_i^{\mathrm{GS}} - x_i^{(k)})$. For both approaches, find the iteration function $\mathbf{x}^{(k+1)} = \mathbf{G}\mathbf{x}^{(k)} + \mathbf{c}$, and the splitting $\mathbf{A} = \mathbf{M} - \mathbf{N}$ that it corresponds to.

4. When solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ using GMRES, starting the Arnoldi iteration from $\mathbf{b}$ is essential. Justify this statement by explaining the cases in which GMRES can fail to find the solution if started from a different vector, and prove that it will always find the solution when started from $\mathbf{b}$.

5. Consider the problem $\mathbf{A}\mathbf{x} = \mathbf{b}$ with a real symmetric positive definite matrix $\mathbf{A}$. Recall that the Arnoldi iteration finds orthonormal bases for successive Krylov subspaces, $\mathcal{K}_n = \langle \mathbf{q}_1, \ldots, \mathbf{q}_n \rangle$. Suppose instead I apply A-weighted Gram-Schmidt to find bases $\mathcal{K}_n = \langle \mathbf{z}_1, \ldots, \mathbf{z}_n \rangle$ such that $\mathbf{z}_i^T \mathbf{A} \mathbf{z}_j = 0$ for all $i \neq j$. What is the relationship between these basis vectors and the conjugate gradient algorithm? Prove your claim.

6. Let G be an undirected graph with vertices $1, 2, \ldots, m$. The graph Laplacian of G is a sparse, symmetric positive semidefinite matrix $\mathbf{L}$ with diagonal entries $l_{ii} = \deg(i)$ and off-diagonal entries $l_{ij} = -1$ iff vertices i and j are connected by an edge. The graph Laplacian is a key concept in many areas of both theoretical and applied graph theory.

   The provided code (Matlab / Python) returns the Laplacian of an m_1-by-m_2 grid graph as an $m_1 m_2 \times m_1 m_2$ sparse matrix (in Python, a `scipy.sparse.dok_matrix`). The Laplacian is singular, so let's work with $\mathbf{A} = \mathbf{L} + \frac{20}{m_1^2 + m_2^2}\mathbf{I}$ instead (use `speye`/`scipy.sparse.eye` for $\mathbf{I}$).

   (a) Write a function `R = incompleteCholesky(A)` that computes an incomplete Cholesky factorization of $\mathbf{A}$ and returns the upper triangular factor $\tilde{\mathbf{R}}$ as a sparse matrix. This is the same as the Cholesky factorization except that, at each step of the factorization process, entries that were zero in $\mathbf{A}$ are left untouched in $\tilde{\mathbf{R}}$. Your function should run in strictly less than $O(m^3)$ time, by using e.g. `find(A(i,:))` / `A.getrow(i).nonzero()` to iterate over only the nonzero entries of $\mathbf{A}$.

   (b) Although $\tilde{\mathbf{R}}^T\tilde{\mathbf{R}} \neq \mathbf{A}$, we can still use $\tilde{\mathbf{R}}^T\tilde{\mathbf{R}}$ as $\mathbf{M}$ in a stationary iterative method to solve $\mathbf{Ax} = \mathbf{b}$. Derive such a method and implement a function `icIteration(A, b, R, x)` that performs one step $\mathbf{x}^{(n)} \mapsto \mathbf{x}^{(n+1)}$. Use built-in methods for matrix-vector multiplication and backsubstitution (`R\b` / `scipy.sparse.linalg.spsolve_triangular(R, b)`).

   (c) Implement a conjugate gradient solver, `conjugateGradient(A, b, tolerance)`, for solving $\mathbf{Ax} = \mathbf{b}$. Keep iterating until $\|\mathbf{r}_n\|_2 / \|\mathbf{b}\|_2 \leq$ tolerance, and return the final iterate $\mathbf{x}_n$ and the history of relative residual norms $(\|\mathbf{r}_1\|_2/\|\mathbf{b}\|_2, \ldots, \|\mathbf{r}_n\|_2/\|\mathbf{b}\|_2)$.

   (d) Implement `preconditionedConjugateGradient(A, b, R, tolerance)`, which does the same thing but using $\mathbf{M} = \tilde{\mathbf{R}}^T\tilde{\mathbf{R}}$ as a symmetric preconditioner. You may use the standard CG method applied to the transformation given in Trefethen and Bau, or the "untransformed PCG" method described in the article by Shewchuk.

   Test your solvers by choosing a random vector $\mathbf{b} \in \mathbb{R}^{m_1 m_2}$, running the method, and plotting the relative residual norm $\|\mathbf{r}_n\|/\|\mathbf{b}\|$ on a log scale versus iteration number n. In your report, include a single graph showing convergence plots of all three methods on the same $\mathbf{b}$. All your functions are expected to scale to moderately large matrices.

Submission: Submit a PDF of your answers for all questions to Gradescope. Submit the code for Question 6 in a zip file to Moodle. Both submissions must be uploaded before the assignment deadline.

Python submissions should contain a single file, `q6.py`, which contains all the functions. Matlab submissions necessarily will require multiple `.m` files named after the functions they contain. No function is required to produce any side-effects like printing out values or drawing plots. Any results you are asked to show should go in the PDF.