Fall 2015: Numerical Methods I
Assignment 2 (due Oct. 8, 2015)

1. [Properties of LU decomposition, 1+2pt]
   
   (a) Give an example of an invertible $3 \times 3$ matrix that does not have zero entries, for which the LU decomposition without pivoting fails.

   (b) Show that the LU factorization of an invertible matrix $A \in \mathbb{R}^{n \times n}$ is unique. That is,

   $$A = LU = L_1 U_1$$

   with upper triangular matrices $U, U_1$ and lower triangular matrices $L, L_1$ that have 1’s in the diagonal necessarily implies that $L = L_1$ and $U = U_1$. **Hint:** Show first that the set $L$ of lower triangular matrices with 1’s on the diagonal is a subgroup of the group of invertible matrices, i.e., $L$ contains the identity matrix, the product of two elements in $L$ is again in $L$, and the inverse of an element in $L$ is also in $L$.

2. [Spectral radius, diagonally dominant matrices, 1+2+1pt] Let $A \in \mathbb{R}^{n \times n}$, $n \geq 1$.

   (a) Show that the spectral radius $\rho(A)$ is a lower bound to all induced matrix norms, i.e.,

   $$\rho(A) := \max_{1 \leq i \leq n} |\lambda_i(A)| \leq \|A\|,$$

   where $\|A\|$ is the matrix norm induced by an arbitrary vector norm $\| \cdot \|$.

   (b) Recall that a matrix is called strictly (row) diagonally dominant if

   $$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \quad \text{for } i = 1, \ldots, n.$$

   Show that the LU-factorization without pivoting of an invertible matrix $A$ can be performed if $A^T$ is strictly diagonally dominant.

   (c) Can you find an example of an invertible, strictly (row) diagonally dominant matrix, where the LU factorization without pivoting fails?

3. [LU decomposition for a tridiagonal matrix, 3pt] Given is a tridiagonal matrix

   $$A = \begin{bmatrix}
a_1 & c_1 & & b_1 \\
a_2 & c_2 & \ddots & \ddots & b_2 \\
& \ddots & \ddots & a_{n-1} & c_{n-1} \\
& & b_{n-2} & a_{n-1} & c_{n-1} \\
& & & b_{n-1} & a_n
\end{bmatrix}.$$
Assuming that $A$ has an LU decomposition, i.e., $A = LU$ with
\[
L = \begin{bmatrix}
1 & & \\
d_1 & 1 & \\
& \ddots & \ddots \\
d_{n-1} & & 1 \\
\end{bmatrix}, \quad U = \begin{bmatrix}
e_1 & f_1 & & \\
& \ddots & \ddots & \\
& & \ddots & \ddots \\
e_{n-1} & & & f_{n-1} \\
& & & e_n \\
\end{bmatrix},
\]
derive recursive expressions for $d_i, e_i$ and $f_i$.

4. **[Forward substitution, 5pt]** We implement forward substitution to solve the triangular system
\[
Lx = b,
\]
where $L \in \mathbb{R}^{n \times n}$ is an invertible, lower triangular matrix, and $x, b \in \mathbb{R}^n$. Write functions that take $L$ and $b$ as input arguments and return the solution $x$, and hand in code listings with your assignment.

- Implement a naive forward substitution $x = \text{forward1}(L, b)$ by looping over rows (outer loop) and over columns (inner loop) (compare with Program 1 in Section 3.2.1 of Quarteroni/Sacci/Saleri, where the inner operation is replaced by looping over columns).
- Implement forward substitution $x = \text{forward2}(L, b)$ by looping over the rows (compare with class notes and Program 1 in Section 3.2.1 of Quarteroni/Sacci/Saleri).
- Implement forward substitution $x = \text{forward3}(L, b)$ by looping over the columns (compare with class notes and Program 2 in Section 3.2.1 of Quarteroni/Sacci/Saleri).

Choose a lower triangular matrix with random entries to verify that your implementations are correct by comparing solutions with the build-in solver. Report timings for large systems ($n = 10,000$ and larger) for each of your implementation as well as the build-in implementation for solving linear systems ($\backslash$ in MATLAB).\footnote{Despite the fact that each method requires roughly the same number of floating point operations, timings should differ significantly. This is mainly due to memory access and how matrices are stored in memory. MATLAB stores matrices as one-dimensional arrays, column by column. You can see that by accessing matrices with only one index, i.e., using $A(k)$ for an $n \times n$ matrix returns the entry $A(m, l)$, where $k = m + (l-1)n$, $1 \leq l, 1 \leq m \leq n$. Numbers that are next to each other in memory can be read from memory much faster than numbers that are stored further away from each other.}

5. **[Implementing Choleski, 3+2pt]** Implement the Choleski decomposition for a symmetric and positive definite (spd) matrix $A$. The function $C = \text{mychol}(A, b)$ should terminate if it encounters a zero or negative diagonal element, and otherwise return the upper triangular Choleski factor $C$ such that $A = C'C$.

- Verify your implementation by comparing with the build-in Choleski factorization. Document that comparison and hand in a code listing of $\text{mychol}()$.
- Report timings for large (e.g., $n = 500, 1000, 5000, 10000$) spd matrices for your own as well as the build-in Choleski factorization.\footnote{You can generate a random spd matrix by multiplying a matrix with random entries with its transpose. The resulting matrix is symmetric and positive semidefinite, and almost always positive definite.} Do the timings grow as expected from floating point operation counts?
6. [Sherman-Morrison formula, 6pt] Sometimes, one has to solve linear systems that are rank-1 modifications of other linear systems, for which a factorization is already available. Let us derive a solution algorithm for the modified system. Let \( A \in \mathbb{R}^{n \times n} \) be invertible and \( u, v \in \mathbb{R}^n \) be column vectors.

(a) Show that the matrix \( A + uv^T \) is invertible if and only if \( v^T A u \neq -1 \), and that in this case the inverse is

\[
(A + uv^T)^{-1} = A^{-1} - \frac{1}{1 + v^T A^{-1} u} A^{-1} uv^T A^{-1}.
\]

(b) Let \( v^T A^{-1} u \neq -1 \) and assume given the LU decomposition of \( A \). Specify an efficient algorithms based on the Sherman-Morrison formula to solve the rank-1 modified system \((A + uv^T)x = b\) for a given right hand side \( b \in \mathbb{R}^n \).

(c) Use the above algorithm to solve the system

\[
\begin{bmatrix}
2 & 2 & 3 & 4 & 5 \\
2 & 6 & 6 & 8 & 10 \\
3 & 6 & 10 & 12 & 15 \\
4 & 8 & 12 & 18 & 20 \\
5 & 10 & 15 & 20 & 30 \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
\end{bmatrix}
= 
\begin{bmatrix}
16 \\
32 \\
46 \\
62 \\
80 \\
\end{bmatrix}
\]

by choosing a proper diagonal matrix for \( A \).

7. [Iterative solution of linear systems, 6pt] Let us study iterative solvers for linear systems \( Ax = b \), where \( A \in \mathbb{R}^{n \times n} \) and \( x, b \in \mathbb{R}^n \). For every \( n \), \( A \) and \( b \) have the same structure, for instance for \( n = 7 \):

\[
A = \begin{bmatrix}
0 & 0 & 3 & -1 & 0 & 0 & 0 \\
0 & 0 & 1 & 3 & -1 & 0 & 0 \\
0 & 0 & 0 & -1 & 3 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 3 & -1 \\
-1 & 0 & 0 & 0 & 0 & -1 & 3 \\
3 & -1 & 0 & 0 & 0 & 0 & -1 \\
-1 & 3 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}, \quad b = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\]

(a) Why is it not possible to apply the Jacobi or the Gauss-Seidel iteration for solving this linear system directly? Which rows/columns must be exchanged such that the resulting linear system can be solved with these methods? In the following, we always consider the system with properly exchanged rows/columns.

(b) Argue why the Jacobi and the Gauss-Seidel method converge.

(c) Specify the iteration for the Jacobi and the Gauss-Seidel iteration component-wise.

(d) Implement both methods using component-wise expressions—you should never have to assemble a matrix. Apply your implementation for the solution of the problem for \( n = 10^2, 10^3, 10^4 \), and report the number of iterations for each \( n \) for both methods. Use \( x_0 = 0 \) as initialization, and terminate the iteration when the norm of the residual
\[ r_k := b - A x_k \] has decreased by a factor of \(10^8\) compared to the initial residual \(r_0\), i.e., when
\[
\frac{\|r_k\|}{\|r_0\|} < 10^{-8}.
\]

8. **[Solving without explicit knowledge of system matrix, up to 4pt extra credit]** As mentioned in class, it can happen that one needs to solve a linear system of the form
\[ A x = b, \]
where the entries of \(A\) are not available explicitly, but we can apply \(A\) to vectors. Imagine we only have a function \(A\text{fun}\), which applies the matrix \(A\) to vectors, i.e., \(A\text{fun}(x) = Ax\).

(a) I provide you with a function \(A\text{fun}\) corresponding to a linear matrix \(A\). The function corresponds to the discretization of a differential operator, and can be applied to vectors of length \(N^2\), where \(N \in \mathbb{N}\)—that is, the underlying matrix is of size \(N^2 \times N^2\). We know that the diagonal entries of the underlying matrix \(A\) are all equal to \(4(N+1)^2\), and that the matrix is symmetric and positive definite. Experiment with the Jacobi method to solve the system for \(N = 50\), where the right hand side \(b\) is a vector of all ones.

(b) Try to find the optimal damping parameter for the Jacobi method. Compare the number of iterations required to achieve a tolerance without damping and with the optimal damping parameter.

(c) An important iterative solver for symmetric positive definite linear systems in the conjugate gradient (CG) method. Similarly to the Jacobi method, CG also only requires a function \(A\text{fun}\) rather than the explicit matrix. Moreover, CG does not even require knowledge of the diagonal of \(A\). Solve the linear system using CG and compare the number of iterations with the Jacobi method.

---

\(^3\)Please only attempt this problem if you are interested in it and not for the extra points; the problem is meant to give you more freedom and allow you to explore aspects of iterative solvers.

\(^4\)This can happen, for instance, if \(A\) involves a sequence of linear matrix applications and solves and it would be very inefficient to multiply out all these operations, or if \(A\) involves linear operations for which fast algorithms (e.g., the fast Fourier transform (FFT) or the fast multipole method (FMM)) are available.

\(^5\)Download the MATLAB function from [http://cims.nyu.edu/~stadler/num1/material/Afun.m](http://cims.nyu.edu/~stadler/num1/material/Afun.m).

\(^6\)This will require an estimate of smallest and largest eigenvalues. MATLAB provides the function \(\text{eigs}\) for estimating eigenvalues of large matrices that are not available explicitly. Use \(\text{help eigs}\) to learn more.

\(^7\)Learn about MATLAB’s CG implementation \(\text{pcg}\) by reading the corresponding help pages. We will not have much time to cover the CG method in class, but it is one of the most important iterative solvers, and an important member of the family of so-called Krylov subspace methods.

\(^8\)Each iteration of CG requires one call to \(A\text{fun}\), i.e., one application of \(A\) to a vector. The same is true for the Jacobi method, and usually the computation of \(Ax\) is the most expensive part of the algorithms. Thus, the cost per CG and Jacobi iteration is similar, and one can just compare the iterations numbers.