Fall 2022: Numerical Analysis
Assignment 2 (due Oct 5, 2022 at 11:59pm ET)

Handing in code listings When code listings are required, please include them when you upload your homework on Gradescope. We will try to look at them and point out where things could be improved. Being able to write useful code is important for both this class and in real life. In your programs, use meaningful variable names, try to write clean, concise and easy-to-read code, and use comments for explanation.

1. **[Newton's method in two dimensions, 1+1+2+2pt]** Let $f : \mathbb{R}^2 \mapsto \mathbb{R}^2$ defined by $f(x,y) = (f_1(x,y), f_2(x,y))^T$, where

$$f_1(x,y) = 2x^2 + 8y^2 - 8, \quad f_2(x,y) = y - \frac{\sqrt{3}}{2}x^2.$$

We want to find the roots of f, i.e., all pairs $(x,y) \in \mathbb{R}^2$ such that $f(x,y) = (0,0)^T$. These are the points where the two curves intersect.

   (a) Sketch or plot the sets $\mathcal{S}_i = \{(x,y) \in \mathbb{R}^2 : f_i(x,y) = 0\}$, $i = 1,2$, i.e., the set of all zeros of f_1 and f_2. What geometrical shapes do these sets have?

   (b) Calculate analytically the roots of f, i.e., the intersection of the sets $\mathcal{S}_1$ and $\mathcal{S}_2$.

   (c) Calculate the Jacobian matrix of f, defined by

$$J_f(x,y) = \begin{pmatrix} \partial_x f_1(x,y) & \partial_y f_1(x,y) \\ \partial_x f_2(x,y) & \partial_y f_2(x,y) \end{pmatrix} \in \mathbb{R}^{2\times 2}.$$

   Here, $\partial_x f_i(x,y)$ and $\partial_y f_i(x,y)$, $i = 1,2$ denote the partial derivatives of f_i with respect to x and y, respectively.

   (d) The Newton method in 2D is as follows: Starting from an initial value $(x_0, y_0)^T \in \mathbb{R}^2$, compute the iterates

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ y_k \end{pmatrix} - [J_f(x_k, y_k)]^{-1} f(x_k, y_k), \text{ for } k = 0, 1, \ldots,$$

   where $[J_f(x_k, y_k)]^{-1}$ is the inverse of the Jacobi matrix of f evaluated at (x_k, y_k). Implement the Newton method in 2D and use it to calculate the first 5 iterates for the starting values $(x_0, y_0) = (2, 3)$ and $(x_0, y_0) = (-1.5, 2)$. Plot these iterates in the xy-plane together with the curves $\mathcal{S}_1$ and $\mathcal{S}_2$. Please also hand in your code.

2. **[Properties of LU factorization, 1+2pt]** We study basic properties of the LU-factorization.

   (a) Give an example of an invertible 3×3 matrix that does not have any 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, if

$$A = LU = L_1 U_1$$

   with upper triangular matrices U, U_1 and unit lower triangular matrices L, L_1, then necessarily $L = L_1$ and $U = U_1$. Use the results we discussed in class about products of lower/upper triangular matrices, and their inverses.

3. **[LU for transposed matrix, 2+2pt]** Let $n \geq 2$. Consider a matrix $A \in \mathbb{R}^{n\times n}$ for which every leading principal submatrix of order less than n is non-singular.

   (a) Show that A can be factored in the form $A = LDU$, where $L \in \mathbb{R}^{n\times n}$ is unit lower triangular, $D \in \mathbb{R}^{n\times n}$ is diagonal and $U \in \mathbb{R}^{n\times n}$ is *unit* upper triangular.

   (b) If the factorization $A = LU$ is known, where L is unit lower triangular and U is upper triangular, show how to find the LU-factors of the transpose A^T. Note that our requirement for an LU-factorization is that L is *unit* lower triangular, and U is upper triangular.

4. **[LU factorization of tridiagonal matrix, 3+1pt]** Given is a tridiagonal matrix, i.e., a matrix with nonzero entries only in the diagonal, and the first upper and lower subdiagonals:

$$A = \begin{bmatrix} a_1 & c_1 & & & \\ b_1 & a_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-2} & a_{n-1} & c_{n-1} \\ & & & b_{n-1} & a_n \end{bmatrix}.$$

   (a) Assuming that A has an LU decomposition $A = LU$ with

$$L = \begin{bmatrix} 1 & & & \\ d_1 & 1 & & \\ & \ddots & \ddots & \\ & & d_{n-1} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} e_1 & f_1 & & \\ & \ddots & \ddots & \\ & & e_{n-1} & f_{n-1} \\ & & & e_n \end{bmatrix},$$

   derive recursive expressions for d_i, e_i and f_i. You can do this similar to what we did in class when deriving explicit expressions for the entries in the LU decomposion.

   (b) Use this above formula to give the LU-factorization for the matrix[1]

$$A = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \in \mathbb{R}^{N\times N}.$$

5. **[Cholesky factorization, 3+2pt]**

   (a) A matrix A is called symmetric if $A = A^T$. If we compute $A = LDU$ for a symmetric matrix A, and find that each diagonal entry of $D > 0$, argue that A can be factored as $A = RR^T$ and find R in terms of L, D, U. Note that the factorization $A = RR^T$ can only be done for some types of symmetric matrices (specifically symmetric-positive-definite (spd) matrices), and is referred to as a Cholesky factorization.

[1] One encounters this matrix often: It occurs when one attempts to find, for a given function $f : [0,1] \mapsto \mathbb{R}$, a function $u : [0,1] \mapsto \mathbb{R}$ such that

$$-u'' = f \text{ in } (0,1), \text{ and } u(0) = 0, u(1) = 0.$$

If one approximates $-u''$ at $(N+2)$ points $x_i = i/(N+1)$, $i = 0, \ldots, N+1$, using the so-called finite-difference gradient

$$-u''(x_i) \approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2},$$

then finding an approximation of the function u at x_i, $i = 1, \ldots, N$ requires solving a system with the matrix A.

(b) The algorithm to find the Cholesky factorization of an $n \times n$ matrix A is as follows: for $j = 1, \ldots, n$

i. $r_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} r_{jk}^2}$

ii. for $i > j$: $r_{ij} = \frac{1}{r_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} r_{ik} r_{jk} \right)$

Use this to compute the Cholesky factorization of

$$A = \begin{bmatrix} 2 & 1 & 1/2 & 1/4 \\ 1 & 4 & 1 & 1/2 \\ 1/2 & 1 & 4 & 1 \\ 1/4 & 1/2 & 1 & 2 \end{bmatrix}.$$

To avoid computing the square root of a negative number (which happens for matrices that are not spd), check at each step that the quantity under the square root in the computation of r_{jj} is positive. If it is not, display an error message[2] and stop the code. Please hand in your commented code.

6. **[Right hand sides with many zeros, 4pt]** For a given dimension n, fix some k with $1 \leq k \leq n$. Now let $L \in \mathbb{R}^{n \times n}$ be a non-singular lower triangular matrix and let the vector $\boldsymbol{b} \in \mathbb{R}^n$ be such that $b_i = 0$ for $i = 1, 2, \ldots, k$.

   (a) Let the vector $\boldsymbol{y} \in \mathbb{R}^n$ be the solution of $L\boldsymbol{y} = \boldsymbol{b}$. Show, by partitioning L into blocks, that $y_j = 0$ for $j = 1, 2, \ldots, k$.

   (b) Use this to give an alternative proof of Theorem 2.1 (iv), i.e., that the inverse of a non-singular lower triangular matrix is itself lower triangular.

7. **[Inverse matrix computation, 1+1+2pt]** Let us use the LU-decomposition to compute the inverse of a matrix[3].

   (a) Describe an algorithm to compute the inverse of a matrix $A \in \mathbb{R}^{n \times n}$ that uses the LU-decomposition of an by solving n systems of equations (one for each unit vector).

   (b) Give the floating point operation count of this algorithm.

   (c) Improve the algorithm by taking advantage of the structure (i.e., the zero entries—see previous problem) of the right-hand side. What is the new algorithm's floating point operation count?

[2] Python has a command `assert` that is useful for this.

[3] This also illustrates that computing a matrix inverse is significantly more expensive than solving a linear system. That is why to solve a linear system, you should *never* use the inverse matrix!