Numerical Analysis
Assignment 2 (due March 12, 2020)

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

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

   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.

2. **[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

$$A = LU = L_1U_1$$

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

3. **[4pt]** For a given dimension n, fix some k with $1 \le k \le n$. 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 $i = 1, 2, \dots, k$.

   (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 $j = 1, 2, \dots, k$.

   (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.

4. **[4pt]** Let $n \ge 2$. 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.

5. **[5pt]** Implement backward substitution to solve systems $U\boldsymbol{x} = \boldsymbol{b}$, i.e., write a function `x = backward(A,b)`, which expects as inputs an upper triangular matrix $U \in \mathbb{R}^{n\times n}$, and a right hand side vector $\boldsymbol{b} \in \mathbb{R}^n$, which returns the solution vector $\boldsymbol{x} \in \mathbb{R}^n$. The function should find the size n from the vector $\boldsymbol{b}$ and also check if the matrix and the vector sizes are compatible before it starts to solve the system. Please hand in your code. Apply your program for the computation of for $\boldsymbol{x} \in \mathbb{R}^4$, with

$$U = \begin{bmatrix} 1 & 2 & 6 & -1 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 4 & -1 \\ 0 & 0 & 0 & 2 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} -1 \\ -3 \\ -2 \\ 4 \end{bmatrix}.$$

6. **[3+2pt]** LU factorization without pivoting.

   (a) Implement the LU factorization using $(2.18), (2.19)$ from the textbook (hence assuming no permutations are required), and apply it to the matrix

$$A = \begin{bmatrix} 6 & 2 & 1 & -1 \\ 2 & 4 & 1 & 0 \\ 1 & 1 & 4 & -1 \\ -1 & 0 & -1 & 3 \end{bmatrix}.$$

   (b) Generalize your code to handle input matrices A of any size $n \geq 2$. To avoid division by very small numbers or zero, check at each step that the absolute value of u_{jj} in (2.18) is not smaller than 10^{-8}. If it is, display an error message[1] and stop the code. Please also hand in your code.

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

   (a) Describe an algorithm that uses the LU-decomposition of an $n \times n$ matrix A for computing A^{-1} by solving n systems of equations (one for each unit vector).

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

   (c) Improve the algorithm by taking advantage of the structure (i.e., the zero entries—see question 5a) of the right-hand side. What is the new algorithm's floating point operation count?

8. **[2+1+2pt+2pt (extra credit)]** Let us explore matrix norms and condition numbers.

[1]MATLAB has the command `error('message')` for doing that.

[2]This also illustrates that computing a matrix inverse is significantly more expensive than solving a linear system.

(a) For the following matrix given by

$$A = \begin{bmatrix} 1 & -2 \\ 3 & -1 \end{bmatrix},$$

calculate $\|A\|_1$, $\|A\|_2$, $\|A\|_\infty$ as well as the condition numbers for each norm by hand. Is A well or ill-conditioned?

(b) Recall the formulas from Theorems 2.7 and 2.8 in the text book. If you assume that taking the absolute value and determining the maximum does not contribute to the overall computational cost, how many *flops* (floating point operations) are needed to calculate $\|A\|_1$ and $\|A\|_\infty$ for $A \in \mathbb{R}^{n\times n}$? By what factor will the calculation time increase when you double the size of matrix size?

(c) Now implement a simple code that calculates $\|A\|_1$ and $\|A\|_\infty$ for a matrix of any size $n \geq 1$. Try to do this without using loops[3]! Using system sizes of $n_1 = 100$, $n_{k+1} = 2n_k, k = 1, \dots, 7$, determine how long your code takes[4] to calculate $\|A\|_1$ and $\|A\|_\infty$ for a matrix $A \in \mathbb{R}^{n_i \times n_i}$ with random entries and report the results. Can you confirm the estimate from (b)?

(d) **(extra credit)** MATLAB has the build-in function `norm` to calculate matrix norms.[5] Calculate for the system sizes in (c) $\|A\|_1$ and $\|A\|_\infty$ using both your implementation and MATLAB's `norm` function, determine for each n_i how long each code takes and plot the results in one graph. On average, by what factor is MATLAB's implementation faster than yours?

Please also hand in your code.

9. **[3+3pt]** Estimates for vector and matrix norms.

(a) Show that, for any $\boldsymbol{v} \in \mathbb{R}^n$, we have

$$\|\boldsymbol{v}\|_\infty \leq \|\boldsymbol{v}\|_2 \quad \text{and} \quad \|\boldsymbol{v}\|_2^2 \leq \|\boldsymbol{v}\|_1 \|\boldsymbol{v}\|_\infty .$$

In each case, give an example of a nonzero $\boldsymbol{v}$ for which equality is obtained.

(b) Let us generalize the definition of matrix norms to non-square matrices. We define the $\|\cdot\|_p$ matrix norms ($p \in \{1, 2, \infty\}$) for an $m \times n$ matrix A by

$$\|A\|_p = \sup_{\boldsymbol{v}\in\mathbb{R}^n\setminus\{\boldsymbol{0}\}} \frac{\|A\boldsymbol{v}\|_p}{\|\boldsymbol{v}\|_p}$$

where the norm in the numerator is defined on $\mathbb{R}^m$ and the norm in the denominator is defined on $\mathbb{R}^n$.

Using the problem above, show that

$$\|A\|_\infty \leq \sqrt{n}\,\|A\|_2 \quad \text{and} \quad \|A\|_2 \leq \sqrt{m}\,\|A\|_\infty$$

In each case, give an example of a nonzero matrix A for which equality is obtained.

[3]The commands needed in MATLAB are `abs` and `sum`. Most commands can not only applied to numbers, but also to vectors, where they apply to each component.

[4]In MATLAB use the *stop watch* commands `tic` and `toc`.

[5]Use `help norm` to find out how to obtain the matrix norm that is induced by either the 1,2 or ∞-vector norm.