Spring 2021: Numerical Analysis
Assignment 2 (due Monday March 8th 2pm)

1. **[Compound interest, 8pts]** For a yearly interest rate $0 < r < 1$ compounded over n intervals, an amount of money C grows to be

$$f(C, r, n) = C\left(1 + \frac{r}{n}\right)^n \tag{1}$$

after one year. Let $C = 1$ and $r = 0.025$. If n is large, there may be loss of digits when evaluating this using finite-precision arithmetic.

(a) [2pts] If n is *extremely* large, say $n = 10^{16}$, in IEEE double precision arithmetic (try it in Matlab),

$$f\left(1.0, 0.025, 10^{16}\right) = 1.0, \tag{2}$$

when in fact,

$$\begin{aligned} f(1.0, 0.025, 10^{16}) &\approx \lim_{n\to\infty}\left(1 + \frac{0.025}{n}\right)^n \\ &= e^{0.025} \\ &\approx 1.025315\ldots \end{aligned} \tag{3}$$

What happened?

(b) [2pts] To compute $f(C, r, n)$ without roundoff problems in Matlab, compute first $\ln f$ using the (magic!) built-in Matlab function $log1p$ which computes $\ln(1 + x)$ without loosing digits even for very small x, and then compute f from its logarithm. Write down the formulas used. Try this for $n = 10^{16}$. Then repeat the calculation for $n = 10^8$. From now on take $n = 10^8$.

(c) [2pts] Using the result from part (b), how many digits of accuracy do you get for f with direct evaluation of (1)?

(d) [1pts] For large n, we can just use the approximation $f(C, r, n) = Ce^r$. How many digits of accuracy do you get with this approximation?

(e) [1pts] An improved approach for large n is to compute a few terms in the Taylor series expansion (not a trivial calculation per se),

$$(1 + rx)^{1/x} = \mathrm{e}^r\left[1 - \frac{r^2}{2}x + O\left(x^2\right)\right],$$

and then use this approximation for small x. How many digits of accuracy do you get using this approach?
Don't just report answers, explain how you computed this.

2. **[Backward substitution implementation, 5pts]** [3pts] Write a code for backward substitution to solve systems of the form $U\boldsymbol{x} = \boldsymbol{b}$, 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. Apply your program for the computation of for $\boldsymbol{x} \in \mathbb{R}^4$, with

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

[2pts] How do you know that your code is working correctly?

3. **[LU factorization of tridiagonal matrix, 6pt]** 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 iterative expressions for d_i, e_i and f_i, i.e., how to compute the value for $i+1$ from the values at i, and how to start for $i = 1$ (the formula can involve any of $a/b/c/e/d/f$ but only values already computed in *previous* iterations).
Hint: You could check your answer by implementing the formulas in code and checking that $LU = A$ in Matlab for some specific example.

4. **[Inverse matrix computation, 8pts]** Let us use the LU-decomposition to compute the inverse of a matrix[1].

   (a) [2pts] 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).

[1] 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!

(b) [2pts] Calculate the floating point operation count of this algorithm. It is OK to use estimates from class/worksheets but write them down so the grader knows what you are doing.

(c) [4pts] Improve the algorithm by taking advantage of the structure (i.e., the many zero entries) of the right-hand side. What is the new algorithm's floating point operation count?
[*Hint:* Consider splitting the solution vector for the k-th equation from part (a) into two pieces, and solve for each piece separately, on paper or using forward/back substitution.]

5. **[Stability of the Gaussian elimination, 8pts]**

Consider the linear system

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{4}$$

where $\boldsymbol{A}$ is an $n \times n$ matrix that has ones on the diagonal, minus ones below the diagonal, and ones in the last column, with all other entries zero. For example, when $n = 5$, we have

$$\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \end{pmatrix}.$$

(a) [3pts] Prove that A is invertible for any n, by induction. [Hint: Perform a *column operation* on $\boldsymbol{A}$ to eliminate the reduce it to a smaller matrix of size $n-1$ and ask whether that smaller matrix is invertible under the induction hypothesis.]

(b) [3pts] Now consider the matrix $\boldsymbol{A}$ for some unspecified (arbitrary) n. Perform Gaussian elimination on $\boldsymbol{A}$ to obtain the upper triangular matrix $\boldsymbol{U}$ appearing in the LU factorization $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$. What is $\max_{i,j} |u_{i,j}|$ as a function of n?

(c) [2pts] For large n, e.g., $n = 2000$, what problems can you envision if you try to solve (4) using Gaussian elimination on a computer? Explain.
[*Note: This is one rare example matrix for even Matlab will fail to solve a linear system correctly even though the matrix is well-conditioned, see discussion in Section 7.5 of Practice textbook.*]

6. [**Matrix square root, 6pts**] Newton's method for finding roots can be extended to *matrix-valued functions* as well. Here you will devise a Newton method (i.e., generalize the Babylonian method) to compute the square root of a matrix. If it exists, the square root of a real symmetric $n \times n$ matrix $\boldsymbol{A}$ is another *real square symmetric* matrix $\boldsymbol{X}$ such that

$$\boldsymbol{X}^T\boldsymbol{X} = \boldsymbol{A} \tag{5}$$

Just like the square root of even a positive number is not unique, the matrix square is not unique (one can roughly think of having to choose n signs, as we will revisit in a future homework once we cover eigenvalue decompositions).

(a) [2pts] In class/worksheet we used derivatives to obtain Newton's method. Instead of computing derivatives of matrix-valued functions, however, it is useful to think of computing derivatives from a *linearization* of the function around a given value (this allows to generalize the notion of a derivative and makes it easier to compute in some cases). Set $\boldsymbol{X} = \boldsymbol{X} + \delta\boldsymbol{X}$ in (5) and keep only the terms that are linear in the 'perturbation" $\delta\boldsymbol{X}$. Use this to write down an equation for $\delta\boldsymbol{X}$. [*Note: Another way to say this is to ask you to write down a first-order Taylor series of* $\boldsymbol{f}(\boldsymbol{X}) = \boldsymbol{X}^T\boldsymbol{X} - \boldsymbol{A}$.]

(b) [2pts] The equation you obtained for $\delta\boldsymbol{X}$ in part (a) can be solved explicitly for any n. To see why, explain why the equations are linear and explain how many unknowns and how many equations there are (remember to use symmetry of matrices). [*Note: In Matlab the function sylvester solves this kind of equation.*] It is OK if you assume a unique solution exists. Take $n = 2$ and write down the solution explicitly. [*Hint: It is always a good idea to check by plugging in specific numbers.*]

(c) [2pts] It would be nice to write down an explicit formula for the solution of the equation you got from part (a) for any n. Do this by assuming that the matrices $\boldsymbol{X}$ and $\delta\boldsymbol{X}$ commute, i.e., that

$$\boldsymbol{X}(\delta\boldsymbol{X}) = (\delta\boldsymbol{X})\boldsymbol{X}. \tag{6}$$

[Hint: Recall that $\boldsymbol{X}$ is symmetric.].
Note: One can prove (6) holds at all iterations if the initial guess $\boldsymbol{X}_0$ commutes with $\boldsymbol{A}$; if interested, look at the paper "Newton's Method for the Matrix Square Root" by Nicholas Higham, freely available on the web.