Numerical analysis: Homework 2

Instructor: Anil Damle
Due: February 19, 2024

Policies

You may discuss the homework problems freely with other students, but please refrain from looking at their code or writeups (or sharing your own). Ultimately, you must implement your own code and write up your own solution to be turned in. Your solution, including plots and requested output from your code should be typeset and submitted via the Gradescope as a pdf file. This file must be self contained for grading. Additionally, please submit any code written for the assignment as zip file to the separate Gradescope assignment for code.

Question 1:

When asked "is it reasonable to compute e^{-x} via its Taylor expansion" GPT-4 responds:[1]

> Yes, it is reasonable to compute e^{-x} via its Taylor expansion. The Taylor series expansion of e^{-x} is:
>
> $$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \cdots$$
>
> This is an infinite series that converges for all values of x, so you can use it to compute e^{-x} for any x.
>
> However, be aware of the limitations:
>
> 1. **Precision**: The more terms you include in the series, the more accurate your result will be. However, if x is very large, you may need to sum a large number of terms to get an accurate result.
>
> 2. **Efficiency**: If you are computing e^{-x} for very large values of x, it may be more efficient to use other methods, such as computing e^x and then taking its reciprocal.
>
> For small to moderate values of x, computing e^{-x} via its Taylor expansion can be a reasonable and accurate method.

Let's ignore the efficiency part for a moment (as there are far better ways to compute e^{-x} from that standpoint). Assuming we take enough terms, do you agree that for "small to moderate values of x" using the Taylor expansion provides an "accurate" result for e^{-x} on a computer (using, e.g., IEEE double precision)? Let's say moderate here means on the order of 10 to 100—a reasonable interpretation. Justify your response.

[1] See `https://chat.openai.com/share/dc160f6d-d73a-4729-94e5-ee16d409573c`, and remember that the output is random—you may get a different response if you ask the same question.

Question 2:

When discussing LU decompositions in class, we considered matrices $G^{(k)}$ (so-called Gauss Transforms) that when applied to a matrix A introduced zeros in the k^{th} column below the diagonal. Assuming all of the matrices involved in this problem are $n \times n$, we may write these matrices compactly as

$$G^{(k)} = I - \ell^{(k)} e_k^T,$$

where $\ell^{(k)} = [0, \ldots, 0, \ell_{k+1,k}, \ldots \ell_{n,k}]^T$ is zero in the first k entries. We will now prove two properties of these matrices that we used in class.

(a) Prove that $\left(G^{(k)}\right)^{-1} = I + \ell^{(k)} e_k^T$

(b) Prove that $\left(G^{(1)}\right)^{-1} \left(G^{(2)}\right)^{-1} \cdots \left(G^{(n-1)}\right)^{-1} = I + \sum_{k=1}^{n-1} \ell^{(k)} e_k^T$

Question 3:

Assume that you are given an $n \times n$ non-singular matrix of the form

$$A = D + u e_{n-1}^T + e_{n-1} v^T$$

where u and v are length n vectors, and D is a diagonal matrix whose diagonal entries we will encode in the vector d, i.e., $D_{i,i} = d_i$.

Note: While considering the following questions you may stumble across the Sherman-Morrison-Woodbury formula and find it provides a potential solution. You are welcome to try it out and see what happens. However, it will yield a less accurate solution for the given instance of this problem in part (c) than the suggested approach based on an LU factorization.

(a) Under the assumption that we may compute the LU decomposition of A without requiring any pivoting, devise a means to solve a linear system of the form $Ax = b$ using $\mathcal{O}(n)$ time. For the purposes of this problem we will also consider the cost of memory allocation and therefore you cannot form A explicitly as that would take $\mathcal{O}(n^2)$ time.

(b) Implement your method and demonstrate that it achieves the desired scaling. You can do this with random instances of the problem (i.e., randomly generate d, u, and b). While these random instances may benefit from pivoting, they are unlikely to "require" pivoting in a strict sense (i.e., you might be solving the problem inaccurately, but will not end up dividing by zero).

(c) Download the file on the course website containing vectors d, u, v, b and the true solution x_{true}. Each vector is written as plaintext into its own file with entries delineated by newlines. (Technically it is saved as column-oriented csv file with commas delineating columns and newlines delineating rows—but here we just have column vectors.) This format should be easily readable in Python, Julia, or Matlab; just be sure to "read" all the provided digits of the input.

Solve the associated linear system $Ax = b$ for x. Using a logarithmic scale for the error, plot both the absolute error of the difference between your computed solution and x_{true} and the absolute value of the residual vector $r = b - Ax$ generated by your solution vs the index of the respective vector (i.e., the x-axis will range from 1 to 50,000). Also report the 2-norm of both r and the difference between your computed solution and x_{true}. What do you observe?

(d) By avoiding pivoting, we were able to get a fast, $\mathcal{O}(n)$ algorithm. However, what do you observe about the accuracy of the solution and how might you explain it?

(e) If we introduce partial pivoting into our algorithm can we guarantee that the algorithm will still have $\mathcal{O}(n)$ complexity? If not, what about $\mathcal{O}(n^2)$ complexity?

(f) **Ungraded, but interesting to try:** Sometimes, though not always, we may be able to pair a faster, albeit problematic, algorithm with an iterative method (a class of algorithms we will talk more about later in this course) to clean up the solution. One simple methodology is called iterative refinement. Specifically, given an approximate solution $\tilde{x}$ such that $A\tilde{x} \approx b$, we can refine the solution via the following procedure:

   (a) Compute $r = b - A\tilde{x}$

   (b) Solve the linear system $Az = r$

   (c) Let $x = \tilde{x} + z$

   If we solve $Az = r$ exactly, then x will be a solution. However, in general since that is also a problem involving A the solution for z is only approximate and it requires a more careful analysis to determine how much we may improve the solution $\tilde{x}$. In fact, it could even be repeated multiple times. If you are so inclined, try this out for this problem and see if it helps.

QUESTION 4:

Consider the following $n \times n$ matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ -1 & \cdots & -1 & 1 & 0 & 1 \\ -1 & \cdots & -1 & -1 & 1 & 1 \\ -1 & \cdots & -1 & -1 & -1 & 1 \end{bmatrix}$$

that has all ones one the diagonal and in the last column, all the entries below the diagonal are -1, and all other entries are zero.

(a) Consider computing an LU decomposition of A (with or without partial pivoting, the answer will be the same either way). Work out an expression for the largest entry of U in terms of n.

(b) If we computed an LU factorization in practice and encountered an entry of this size do you think it would be problematic? why or why not?

(c) Does the problem persist if we use complete pivoting?

Note that this matrix demonstrates the worst case behavior for the size of elements in U when computing an LU factorization with partial pivoting. Nevertheless, such cases are considered rare in practice and LU with partial pivoting is widely used.