CS 4220 / MATH 4260: Homework 2

Instructor: Anil Damle
Due: February 20, 2019

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 CMS as a pdf file. Additionally, please submit any code written for the assignment via the CMS as well. This can be done by either including it in your solution as an appendix, or uploading it as a zip file via the CMS.

Question 1:

Let A and B be $n \times n$ matrices with real entries (you may assume the entries are explicitly representable in floating point if you like—though it does not change the answer in a meaningful way). For any i and j with $1 \leq i, j \leq n$ bound the floating point error of computing $C_{i,j}$ where $C = AB$. Your bound should be in terms of n, μ (machine precision), and entries of A and B. Given your solution, when might you expect large error in a relative sense?

Question 2:

When discussing Gaussian elimination and then 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 outlined 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.* $D_{i,i} = d_i$. (In the course of thinking about this problem, you may stumble across the so-called 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 not yield an accurate solution for the given instance of this problem in part (c).)

(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

(c) Download the files on the course website containing vectors d, v, b and the true solution x_{true}. 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. What do you observe? (If you are using Julia rather than Matlab, there is a package you can use to read the input file available at: `https://github.com/JuliaIO/MAT.jl`. Similarly, both NumPy and SciPy provide means to read the input file.)

(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 the 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, and all the entries below the diagonal are -1.

(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?

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.