CS 6210: HOMEWORK 1

Instructor: Anil Damle
Due: September 7, 2023

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

QUESTION 1:

This question is intended to serve as a bit of a theoretical warm up for the course. Matrix factorizations and norms will play a key role in our analysis of algorithms.

(a) For any $A \in \mathbb{R}^{n\times n}$ prove that there exists an orthogonal matrix $U \in \mathbb{R}^{n\times n}$ and symmetric positive semi-definite matrix $H \in \mathbb{R}^{n\times n}$ such that

$$A = UH,$$

you may use the existence of the SVD in your proof.

(b) Prove that for any induced matrix norm $\|\cdot\|$ and $A \in \mathbb{R}^{n\times n}$

$$\rho(A) \leq \|A\|,$$

where $\rho(A)$ is the spectral radius of A (i.e., the magnitude of the largest eigenvalue).

(c) **(An ungraded, slightly more challenging question)** Since real matrices can have complex eigenvalues, if we want to discuss the Schur form (i.e., $A = UTU^*$ with unitary U and upper triangular T) of a real matrix that is not symmetric we may have to consider complex numbers. However, it is useful to consider what we can accomplish with only real numbers.

For the purposes of this problem you may only use the fact that for any matrix $A \in \mathbb{R}^{n\times n}$ there exists at least one scalar $\lambda \in \mathbb{C}$ and associated vector $v \in \mathbb{C}^n$ such that $Av = \lambda v$, and not assume the existence of any other matrix factorizations. Prove that for for any matrix $A \in \mathbb{R}^{n\times n}$ there exists an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ such that

$$Q^T A Q = \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1m} \\ 0 & R_{22} & \cdots & R_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R_{mm} \end{bmatrix}$$

where each R_{ij} is real and R_{ii} is either 1×1 or 2×2 with complex eigenvalues. Such a decomposition is known are the real Schur decomposition.

QUESTION 2:

This question is meant to be a bit of a warm up for something we will do repeatedly in this course: implement, validate, and test algorithms we discuss. Since we have not really discussed any algorithms yet, we will start by simply exploring the variety of performance we can observe even with something as simple as matrix-matrix multiplication. Implement the following four algorithms for computing $C = AB$:

(a) $C(i,j) = \sum_k A(i,k)B(k,j)$; for this part you can only use built in scalar multiplication

(b) $C(i,j) = A(i,:)B(:,j)$; you may now leverage your chosen languages calls to routines for computing inner products

(c) $C = \sum_k A(:,k)B(k,:)$; you may now leverage your chosen languages calls to compute outer products and add matrices

(d) $C(:,i) = AB(:,i)$; you may now leverage your chosen languages calls to compute matrix-vector products.

(e) As a point of comparison we will also use the "built in" routine for computing matrix-matrix multiplication (e.g., simply writing $C = A{*}B$ in Matlab), this is our way of accessing the routine for matrix-matrix multiplication from BLAS (`http://www.netlib.org/blas/`).

For all the above algorithms clearly illustrate that your implementation is $\mathcal{O}\left(n^3\right)$ (as part of this question, argue why your choice of plot clearly and unambiguously illustrates the correct scaling, think about the axes), compare and contrast their performance, and argue about why you believe you might be seeing such differences.

QUESTION 3:

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.

---

[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.

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.

QUESTION 4:

Is plausible/reasonable to try and construct a backwards stable algorithm for computing the outer product xy^T given vectors $x, y \in \mathbb{R}^n$? What about a stable algorithm? Justify your response.

QUESTION 5:

For any square non-singular matrix A, prove that $1/\kappa_2(A)$ is the relative distance to the nearest singular matrix to A in the two-norm. In other words, prove that

$$\frac{1}{\kappa_2(A)} = \min_{\mathrm{rank}(X)<n} \frac{\|A - X\|_2}{\|A\|_2}.$$

QUESTION 6:

Say we solve a problem f with a stable algorithm $\tilde{f}$ at input x. Prove that an upper bound on the relative accuracy behaves as $\mathcal{O}(\kappa(x)\mu)$, where μ is machine precision and $\kappa(x)$ is the relative condition number of f at x.