Homework 2
APPM 5650 Fall 2021
Randomized Algorithms

Due date: Friday, Sep 3 2021, in class **Instructor**: Prof. Becker
Theme: Computational and Linear Algebra Background

Instructions Collaboration with your fellow students is allowed and in fact recommended, although direct copying is not allowed. Please write down the names of the students that you worked with. The internet is allowed for basic tasks only, not for directly looking for solutions.

An arbitrary subset of these questions will be graded.

Problem 1: [MATH] Consider checking if two $n \times n$ matrices A and B are equal, but at least one matrix is given only implicitly (if both matrices are explicitly given, then just compute $\|A - B\|_F$ in $\mathcal{O}(n^2)$ time). For example, your friend claims that they can multiply two matrices with their code, so they compute $C \cdot D = A$, but you're suspicious. You have C and D, and you could compute $C \cdot D = B$ using a code that you trust, but this multiplication is expensive ($\mathcal{O}(n^{2.81})$ using Strassen or $\mathcal{O}(n^3)$ using classical multiplies).

Freivalds' algorithm from the 1970s solves this problem by picking a vector $x \in \mathbb{R}^n$ where each entry $x_i \in \{0, 1\}$ is iid Bernoulli with $p = 1/2$. You then compute Ax and Bx and check if $Ax = Bx$. If $A = B$, then it's always true that $Ax = Bx$. If $A \neq B$, then there's a chance that $Ax = Bx$ still, since, for example, you could have chosen $x = 0$.

Prove that if $A \neq B$, then $\mathbb{P}[Ax = Bx] \leq \frac{1}{2}$. From this result, you can create a practical algorithm where you re-run the test with a new copy of x until either $Ax \neq Bx$, which certifies that $A \neq B$, or stop at iteration k with the guarantee that $A = B$ with probability greater than $1 - 2^{-k}$.

Problem 2: [PROGRAMMING] Let $U \in \mathbb{R}^{n \times 2}$ be a matrix where the first column is $\ln(1), \ln(2), \ldots, \ln(n)$ and the second column is $\ln(n+1), \ln(n+2), \ldots, \ln(2n)$. [There is nothing special about this matrix, I just want everyone to use the same matrix so we can check our answers]

Then let S be a sparse $n \times n$ matrix that is all zeros except for the entries $k \in \Omega$ where Ω is the set of all prime numbers less than 10^8. The entry S_k denotes the i, j^{th} entry of S where $k = i + n(j-1)$, i.e., the usual linear indexing of a matrix in column-major order with 1-based indexing. Define $S_k = \ln(k)$ for $k \in \Omega$.

Task: For $n = 10^5$, compute $\|UU^T - S\|_F$ to 4 significant digits. You may want to do math tricks to make this efficient. You *are* allowed to use any linear algebra package you want.

Hints: in MATLAB, you may find `reshape` convenient to construct u, and `sparse` and `ind2sub` and `primes` helpful to construct S. In Python, `numpy.reshape`, `scipy.sparse.csc_matrix` and `numpy.unravel_index` are the analogous commands, and for prime numbers, you can use `sympy.sieve` with `sieve.extend` and `sieve._list` [quite slow] or try a faster pure numpy solution following suggestions on the internet (such as this blog post which I have pated below or this stack exchange answer). Other useful Python packages in general are `numpy.linalg` and `scipy.sparse.linalg`.

You might find it helpful to compare with the naive implementation for a small n to make sure your code is correct.

On my laptop a few years ago, after setting up U and S (which takes 1.4 seconds), the norm computation takes 0.002 seconds for $n = 10^5$ in Matlab, and similar in Python.

Below is the Python code from the blog post (their algorithm "6", modified for Python 3) which you are welcome to use:

```python
import math
def primes(upto=1000000):
    upto = int(upto)
    primes=np.arange(3,upto+1,2)
    isprime=np.ones((upto-1)//2,dtype=bool)
    for factor in primes[:int(math.sqrt(upto))]:
        if isprime[(factor-2)//2]: isprime[(factor*3-2)//2::factor]=0
    return np.insert(primes[isprime],0,2)
```

Problem 3: [PROGRAMMING] There is no "uniform" probability distribution on the real line since it is unbounded, but on the other hand, we can generalize the uniform distribution to interesting things other than bounded intervals. In this problem, consider the set of orthogonal $n \times n$ matrices. **What kind of eigenvalues can an orthogonal matrix have (are these matrices even diagonalizable?)? Is the set of all orthogonal matrices bounded?**

Now, think of some scheme to generate random orthogonal matrices in as "uniform" a way as possible; this is called the "Haar distrubution". You are not graded based on correctness, but on trying reasonable ideas and then trying to **provide evidence** (mathematical or numerical) that you are correct.

Problem 4: [MATH, NOT REQUIRED] If you're wanting to brush up on numerical linear algebra background, try some of the problems in the sections 2.2 – 2.4 in Golub and van Loan's "Matrix Computation" (either 3rd ed '96 or 4th ed '13); or try the questions at the end of chapter 1 in James Demmel's "Applied Numerical Linear Algebra" (a free PDF is available on SIAM's website since CU has a subscription).

If you haven't seen the complexity of matrix multiplication (e.g., Strassen's algorithm), read the first four paragraphs of this paper by Virginia Vassilevska Williams.