Math 566 **Homework 1** **Due Jan. 20, 2022**
Grady Wright

- Problems marked NLA (Numerical Linear Algebra) are from the book.

1. (**Matrix manipulations**, 10 pts) NLA: Question 1.1 (a)–(b)
2. (**Rank one matrices**, 10 pts) Show the following
   - If A has rank one, then $A = \mathbf{u}\mathbf{v}^T$ for some vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^m$.
   - If $A = \mathbf{u}\mathbf{v}^T$ for some vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^m$, then A has rank one.

   Note: "Show" here means that you need to show the result mathematically.
3. (**Inverse of rank one-perturbation**, 15 pts) Suppose $\mathbf{u}, \mathbf{v} \in \mathbb{R}^m$ and consider the matrix $A = I + \mathbf{u}\mathbf{v}^T$, where I is the m-by-m identity matrix. The matrix A is known as a rank-one perturbation of the identity matrix.
   (a) Show that if A is nonsingular, then its inverse can be written as $A^{-1} = I + \beta\mathbf{u}\mathbf{v}^T$ for some scalar β.
   (b) Determine an expression for β from part (a).
   (c) Determine what $\mathbf{u}$ and $\mathbf{v}$ make A singular.
   (d) If A is singular, give a vector that is in the null space of A.
4. (**More on rank-one perturbations of the identity matrix**, 10 pts) Consider again the rank one perturbation of the m-by-m identity matrix $A = I + \mathbf{u}\mathbf{v}^T$.
   (a) Determine how many floating-point operations or FLOPs (additions, subtractions, multiplications, and divisions) it takes to compute $A\mathbf{x}$, for some non-zero vector $\mathbf{x} \in \mathbb{R}^m$.
   (b) What are the entries of $\text{diag}(A)$?
5. (**Matrix norms**, 10 pts) Consider the following matrix:

$$A = \begin{bmatrix} 4 & -1 & 2 \\ 1 & 2 & 3 \\ -1 & 7 & -5 \end{bmatrix}$$

   (a) Compute the matrix $1-$, $2-$, $\infty-$, and Frobenius-norms of A.
   You may use any software (e.g., MATLAB, Python, or Julia) to compute these quantities. In MATLAB, the command is `norm`.
   (b) For any n-by-n matrix A the following bounds on hold true

$$\begin{aligned} \frac{1}{\sqrt{n}}\|A\|_2 \le \|A\|_1 &\le \sqrt{n}\|A\|_2 \\ \frac{1}{\sqrt{n}}\|A\|_2 \le \|A\|_\infty &\le \sqrt{n}\|A\|_2 \\ \frac{1}{n}\|A\|_\infty \le \|A\|_1 &\le n\|A\|_\infty \\ \|A\|_1 \le \|A\|_F &\le \sqrt{n}\|A\|_2 \\ \rho(A) \le \|A\|_p, &\text{ for } p = 1, 2, \infty \end{aligned}$$

   Verify numerically that each of these bounds hold for the matrix A above by computing the norms explicitly. (Note that the last inequality holds true for any p).

6. (**Matrix multiplication**, 15 pts) Straightforward block partitioned 2×2 matrix multiplication $C = AB$ can be written

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

where

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

In the scheme discovered by Strassen (1969), the computations are rearranged into

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_2 &= (A_{21} + A_{22})B_{11} \\ P_3 &= A_{11}(B_{12} - B_{22}) \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{12})B_{22} \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

$$\begin{aligned} C_{11} &= P_1 + P_4 - P_5 + P_7 \\ C_{12} &= P_3 + P_5 \\ C_{21} &= P_2 + P_4 \\ C_{22} &= P_1 - P_2 + P_3 + P_6 \end{aligned}$$

To make this into an efficient algorithm, each of the matrix-multiplications involved in computing $P_1, \ldots, P_7$ is again computed with Strassen's method, but now on matrices of half the size. This is applied recursively until it is cheaper to compute the matrix-matrix products directly, rather than with Strassen's method.

(a) Verify analytically (by hand or using Mathematica, MATLAB, or Maple) that the Strassen algorithm will give the same result as the traditional algorithm. Make sure to check that you never commute any matrices when doing the derivation.

(b) A MATLAB implementation of Strassen's algorithm is given below. Using this code, or translating it into another language, verify numerically that Strassen's method works for random input matrices of size $n = 2^m$, $m = 4, 5, 6$, and 7. Compare the answers this algorithm gives to the answer computed with regular matrix multiplication by computing the norm of the difference.

```
00001 function c = strass(a,b)
00002 nmin = 16;
00003 [n,n] = size(a);
00004 if n <= nmin;
00005    c = a*b;
00006 else
00007    m = n/2; u = 1:m; v = m+1:n;
00008    p1 = strass(a(u,u)+a(v,v),b(u,u)+b(v,v));
00009    p2 = strass(a(v,u)+a(v,v),b(u,u));
00010    p3 = strass(a(u,u),b(u,v)-b(v,v));
00011    p4 = strass(a(v,v),b(v,u)-b(u,u));
00012    p5 = strass(a(u,u)+a(u,v),b(v,v));
00013    p6 = strass(a(v,u)-a(u,u),b(u,u)+b(u,v));
00014    p7 = strass(a(u,v)-a(v,v),b(v,u)+b(v,v));
00015    c = [p1+p4-p5+p7,p3+p5; p2+p4, p1-p2+p3+p6];
00016 end
```

(c) One can show that the floating point operations (FLOPs) required for Strassen's algorithm is $7 \cdot 7^m - 6 \cdot 4^m$ (i.e. $O(n^{\log_2 7}) \approx O(n^{2.81})$). Determine the value of $n = 2^m$ at which Strassen's algorithm has a lower FLOP count than the standard method of multiplying matrices (i.e. $2n^3 - n^2$). How practical does Strassen's algorithm seem?

(d) (Extra credit, 5 pts) Prove that the total arithmetic operation count for Strassen's algorithm is $7 \cdot 7^m - 6 \cdot 4^m$. (Hint: use induction)

Several algorithmic advances have been made in matrix-matrix multiplication that further reduce the standard $O(n^3)$ FLOPs. The best known algorithm requires $O(n^{2.37188})$ operations [1]. However, these algorithms are even less practical than Strassen's method since n must be astronomical before they require an actual lower number of FLOPs. Interestingly, machine learning has recently also been used to find new techniques for matrix multiplication that require even fewer operations than the best known algorithms in some special cases [2].

References

[1] Ran Duan, Hongxun Wu, and Renfei Zhou. Faster matrix multiplication via asymmetric hashing. *arXiv preprint arXiv:2210.10173*, 2022.

[2] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.