Assignment 6.

Due Thurs., Nov. 16, at 11:59 pm.

Please do not share these problems or the solutions associated with them outside of our class, and do not post them publicly online. Doing so is a violation of academic conduct and can result in unfortunate consequences.

Reading: Lectures 20-21

1. p. 162, Exercise 21.6.

2. **Sher(man)-ly you're joking, Mr. Morrison!**

   The Sherman-Morrison Woodbury formula says that if A is an invertible $m \times m$ matrix, and B is a rank-k matrix that is factored into the product $B = XCY$, with $X \in \mathbb{C}^{m\times k}$, $Y \in \mathbb{C}^{k\times m}$, and $C \in \mathbb{C}^{k\times k}$, C invertible, then the inverse of the sum $A + XCY$ is given by

   $$(A + XCY)^{-1} = A^{-1} - A^{-1}X(C^{-1} + YA^{-1}X)^{-1}YA^{-1}.$$

   This is useful if we already have the inverse of A stored in some way or can otherwise easily compute it, and then need the inverse of $A+$ "low rank update".

   (a) Derive the Sherman-Morrison Woodbury formula by doing block Gaussian elimination to solve the system:

   $$\begin{bmatrix} A & X \\ Y & -C^{-1} \end{bmatrix} \begin{bmatrix} T \\ R \end{bmatrix} = \begin{bmatrix} I \\ 0 \end{bmatrix}.$$

   (b) Write pseudocode for a fast algorithm for solving a linear system $(C + ab^T)x = v$, where C is an $m \times m$ circulant matrix and a, b, v are m-vectors. Assume $C + ab^T$ and C are full rank. How many FFTs do you need and what is the computational complexity of your algorithm? (Note that a small annoyance with MATLAB is that the computation of $b^T F^{-1}$ involves a hidden scalar factor related to MATLAB's definition of the FFT. You can do it fast with the following code: `(fft(b)/m)'`. I don't know how Python or Julia encodes the fft, but you may run into a related problem and need to figure out appropriate scaling.)

   (c) Implement your fast algorithm in MATLAB or a program of your choice. Write a test routine that does the following:

      i. Generates a random true solution x_t with $m = 4000$, generates C using a random vector that represents the first column of C, as well as random vectors a, b, and generates $v = (C + ab^T)x_t$,

      ii. Tests and reports the accuracy of your method for the test problem,

iii. Compares the speed of your method to the naive approach of forming $M = C + ab^T$ and then solving $Mx = v$ using backslash.

Turn in your test routine and any functions it depends on.

3. **Ring around the matrix.** Consider the $m \times m$ matrix $M = A + B$, where B consists of *border data* associated with M:

$$M = A + \begin{bmatrix} b_{11} & 0 & \cdots & 0 & b_{1m} \\ b_{21} & 0 & \cdots & 0 & b_{2m} \\ \vdots & \vdots & & \vdots & \vdots \\ b_{m-1,1} & 0 & \cdots & 0 & b_{m-1,m} \\ b_{m1} & 0 & \cdots & 0 & b_{mm} \end{bmatrix}.$$

These kinds of matrices show up in numerical methods for solving time dependent differential equations that may have evolving boundary conditions. In such a scenario, A may represent a differential operator that remains constant, while B is updated at each time step. We will assume A is nonsingular for this problem.

(a) Write B as a rank r factorization $B = XCY$, where $X \in \mathbb{C}^{m \times r}$, $Y \in \mathbb{C}^{r \times m}$,and C is the $r \times r$ identity matrix. What is the maximum possible value of $r = \text{rank(B)}$?

(b) Consider a time dependent problem $M(t)x(t) = f(t)$, where A is fixed for all time, but $B(t)$ and $f(t)$ may change as t changes. At each timestep t_k, we need to compute a solution $x(t_k)$, where $(A + B(t_k))x(t_k) = f(t_k)$, with $f(t_k)$ and $B(t_k)$ given. Describe in pseudocode an algorithm for computing solutions $x(t_k)$ at $k = 1, 2, \ldots, p$ steps that makes judicious use of the decomposition $PA = LU$. In big O notation, what is the computational complexity of the solver for the first timestep? What is the complexity at any subsequent timestep?

(c) Implement solvers `x1=firstsolve(A, B1, f1)` and `xk=stepsolve(P, L,U Bk, fk)`, for solving the sytem $(A + B_k)x_k = f_k$, where $B_k = B(t_k)$, $f_k = f(t_k)$. You are welcome to use MATLAB's LU decomposition function, which has syntax `[L, U, P] = lu(A)`.

(d) Test the accuracy of your solver for a problem where $m = 2048$, A is randomly generated, and B_k, $x_{k_{true}}$ (the true solution at step k) are randomly generated at each timestep t_k, with $k = 1, 2, \ldots, 20$. Report the error $\max_{k \in \{1,\ldots,20\}} \|x_{k_{true}} - x_k\|_2$. Turn in your test, along with all functions it depends on, including the ones you created in part (c).