HW 6 Tianbo Zhang 1938501

21.6. Suppose $A \in \mathbb{C}^{m\times m}$ is *strictly column diagonally dominant*, which means that for each k,

$$|a_{kk}| > \sum_{j\neq k} |a_{jk}|. \tag{21.11}$$

Show that if Gaussian elimination with partial pivoting is applied to A, no row interchanges take place.

Let A be: $\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & & \\ \vdots & & \ddots & \\ a_{m1} & & & a_{mm} \end{bmatrix}$

Since A is strictly column diagonally dominant.
We know $|a_{kk}|$ is the biggest for each column k

Then note in the process of partial pivoting
For each pivot we are choosing the largest absolute value of each pivot column to be the pivot row.

We will prove this by induction:
Base Case:
For the first row there isn't any row interchange.
Since a_{11} is the biggest for the 1st column.

Induction suppose for first k columns of the elimination step there isn't any row interchange.

Then note the strictly column diagonally dominant submatrix for the first k column is preserved.

Then for the k+1 column, the k+1 to m row will have:
$a'_{k+1+i,k+1} = a_{k,k+1} - \frac{a_{k,k}}{a_{k+1+i,k}} a_{k+1+i,k+1}$ for $i = 0, \cdots, m-k$

Note that for the k operation we have:
$a'_{k,k} = a_{k+1,k} - \frac{a_{k+1,k-1}}{a_{k,k-1}} a_{k,k}$

Hence due to the colum dominance $a_{k+1,k+1}$ is still the biggest element
Thus there will be no row interchange.

□

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 $X, Y \in \mathbb{C}^{m \times k}$ 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 $C + ab^T$, where C is an $m \times m$ circulant matrix and a, b 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?

(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 $b = (C + ab^T)x_t$,

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 $Mx = b$ using backslash.

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

a) i) First we want to eliminate Y so we multiply the first row by YA^{-1}

Then subtract it from the second row we get:

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

Then note the second row gives:

$$(-C^{-1}-YA^{-1}X)R = -YA^{-1}$$

$$\Rightarrow R = (-C^{-1}-YA^{-1}X)^{-1}(-YA^{-1})$$

Then note the first row gives:

$$AT + XR = I$$

Then substitute R back into the expression we have:

$$AT - X(-C^{-1}-YA^{-1}X)^{-1}YA^{-1} = I$$

$$\Rightarrow AT + X(C^{-1}+YA^{-1}X)^{-1}YA^{-1} = I$$

$$\Rightarrow AT = I - X(C^{-1}+YA^{-1}X)^{-1}YA^{-1}$$

$$\Rightarrow T = A^{-1} - A^{-1}X(C^{-1}+YA^{-1}X)^{-1}YA^{-1}$$

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

b) First note since $a, b \in \mathbb{C}^{m \times 1}$

If we want to apply the Sherman-Morrison Woodbury

We have the formula be:

$$(C + ab^T)^{-1} = C^{-1} - C^{-1}a/(1 + b^TC^{-1}a)\,b^TC^{-1}$$

$$= C^{-1} - C^{-1}ab^TC^{-1}/(1+b^TC^{-1}a)$$

This is because it's a scalar

Then note we can find x by:

$$(C+ab^T)^{-1}v = C^{-1}v - C^{-1}a\,b^TC^{-1}v\,/\,(1+b^TC^{-1}a)$$

Then note all the red part can use fft.

Pseudo-code: $m \times m$ $m \times 1$

To solve $(C+ab^T)x = v$ $m \times 1$ $1 \times m$

We have: Find the FFT of the first column of C^{-1}

Find the FFT of a.

Find the FFT of v $m \times 1 \cdot 1 \times m$

Find ifft($Fc^{-1} \cdot Fv$)

ifft($Fc^{-1} \cdot Fa$)

Then plug them in.

We calculated 3 FFTs and the run time is $O(m \log m)$

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

$$M = A + \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1,m-1} & b_{1m} \\ b_{21} & 0 & \cdots & 0 & b_{2m} \\ \vdots & \vdots & & \vdots & \vdots \\ b_{m-1,1} & 0 & \cdots & 0 & b_{m-1,m} \\ b_{m1} & b_{m2} & \cdots & b_{m,m-1} & b_{mm} \end{bmatrix}.$$

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 $X, Y \in \mathbb{C}^{m \times r}$, 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).

a) First the only important vectors in matrix B
Are the vectors along the edges
Then note the maximum rank is determined by the linearly independent column and rows.
Thus we have the maximum possible rank for B is:
r = 2.
Then note let C be the 2x2 identity matrix
X and Y should be in the form of:

$$X = \begin{bmatrix} b_{11} & b_{1m} \\ \vdots & \vdots \\ b_{m1} & b_{mm} \end{bmatrix} \quad Y = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix}$$

Then we will have:
$B = XCY$

b) Psuedo code for first solve:
Find [L, U, P] = lu(A);
Find the inverse of A using $A^{-1} = U^{-1}L^{-1}P$
Determine X and Y by using formula in a)
Use the Sherman - Morrison Woodbury formula to find $M_1^{-1}(t)$
Get $X_1(t)$ by:
Calculate $A^{-1}X$ using Asolver(P, L, U, X)
Calculate $A^{-1}f_1(x)$ using Asolver(P, L, U, $f_1(x)$)
Plug them in the Sherman - Morrison Woodbury formula
Runtime: $O(m^3)$ → since lu() is the one require most run time

Psuedo code for step solve:
Find inverse of A by using $PA = LU \rightarrow A^{-1} = U^{-1}L^{-1}P$
Determine X and Y by using formula in a)
Use the Sherman - Morrison Woodbury formula to find $M_k^{-1}(t)$
Get $X_k(t)$ by:
Calculate $A^{-1}X$ using Asolver(P, L, U, X)
Calculate $A^{-1}f_k(x)$ using Asolver(P, L, U, $f_k(x)$)
Plug them in the Sherman - Morrison Woodbury formula
Runtime: $O(m^3)$ → since the Sherman - Morrison Woodbury formula requires most run time.