CS 770: Assignment 2

Ronghao Yang ID:20511820

October 13, 2017

1 Question 1

Algorithm 1 will explain the implementation of row changing in detail.

```
Data: $A \in R^n$
Result: L,U,P (LU = PA)
L=eye(n);
P=L;
U=A;
for $k$ = 1:n do
    ind ←index of the row that its first non-zero number has the largest magnitude;
    if ind != k then
        switch row k and row ind in U;
        switch row k and row ind in P;
        if $k \geq 2$ then
            switch row k and row ind in L;
        end
    end
    L ← usual way to update L;
    U ← usual way to update U;
end
```

Algorithm 1: LU decomposition with row pivoting

Let

$$A = \begin{bmatrix} 10 & -7 & 0 \\ -3 & 4 & 6 \\ 7 & -2 & 5 \end{bmatrix}$$

By running the algorithm,

$$L = \begin{bmatrix} 1.0000 & 0 & 0 \\ 0.7000 & 1.0000 & 0 \\ -0.3000 & 0.6552 & 1.0000 \end{bmatrix}$$

$$U = \begin{bmatrix} 10.0000 & -7.0000 & 0 \\ 0 & 2.9000 & 5.0000 \\ 0 & 0 & 2.7241 \end{bmatrix}$$

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$L * U = P * A = \begin{bmatrix} 10 & -7 & 0 \\ 7 & -2 & 5 \\ -3 & 4 & 6 \end{bmatrix}$$

2 Question 2

2.1 Question 2a

$$M1 = \begin{bmatrix} 1 & 0 & 0 \\ 1000 & 1 & 0 \\ 2000 & 0 & 1 \end{bmatrix}$$

$$M1 * A = A_2 = (1.0e + 03) * \begin{bmatrix} 0.000 & 0.002 & 0.003 \\ 0 & 2.004 & 3.005 \\ 0 & 4.001 & 6.006 \end{bmatrix}$$

$$M2 = \begin{bmatrix} 1.000 & 0 & 0 \\ 0 & 1.000 & 0 \\ 0 & -1.997 & 1.000 \end{bmatrix}$$

$$U = M2 * A_2 = A_3 = (1.0e + 03) * \begin{bmatrix} 0.000 & 0.002 & 0.003 \\ 0 & 2.004 & 3.005 \\ 0 & -0.001 & 0.005 \end{bmatrix}$$

$$L = (1.0e + 03) * \begin{bmatrix} 0.001 & 0 & 0 \\ -1.000 & 0.001 & 0 \\ -2.000 & 0.002 & 0.001 \end{bmatrix}$$

2.2 Question 2b

$$L = \begin{bmatrix} 1.0000 & 0 & 0 \\ 0.5000 & 1.0000 & 0 \\ -0.0005 & 0.6299 & 1.0000 \end{bmatrix}$$

$$U = \begin{bmatrix} -2.0000 & 1..0720 & 5.6430 \\ 0 & 3.1760 & 1.8015 \\ 0 & 0 & 1.8681 \end{bmatrix}$$

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

$$L * U = \begin{bmatrix} -2.0000 & 1.0720 & 5.6430 \\ -1.0000 & 3.7120 & 4.6230 \\ 0.0010 & 2.0000 & 3.0000 \end{bmatrix}$$

$$P * A = \begin{bmatrix} -2.0000 & 1.0720 & 5.6430 \\ -1.0000 & 3.7120 & 4.6230 \\ 0.0010 & 2.0000 & 3.0000 \end{bmatrix}$$

2.3 Question 2c

Without pivoting, the LU decomposition could generate numbers with big exponents due to the small denominators. Since we only have four digits mantissa, this could result into losing digits in small numbers.

3 Question 3

Partial Pivoting:
Partial pivoting requires $O(n^2)$ number of number comparisons to determine the largest pivot.

Full Pivoting:
Full pivoting requires $O(n^3)$ number of number comparisons to determine the largest pivot.

4 Question 4

A has an LU factorization $\leftarrow$ each upper left block $A_{1:k,1:k}$ is nonsingular:
If A has an LU factorization, the A can be written as

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = LU$$

Where

$$L = \begin{bmatrix} L_{11} & L_{12} \\ L_{21} & L_{22} \end{bmatrix} U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix}$$

As we can see here, A_{11} is a product of L_{11} and U_{11}, so $\det(A_{11}) = \det(L_{11})\det(U_{11})$, L_{11} and U_{11} are both triangular matrices, the determinant of a triangular matrix is the just the product of its diagonal entries. Since in both L_{11} and U_{11}, their diagonal entries are all non-zero, so $\det(L_{11})\det(U_{11})$ is non-zero, so $\det(A_{11})$ is also non-zero. Therefore, A_{11} is non-singular, so each upper left block $A_{1:k,1:k}$ is non-singular.

Each upper left block $A_{1:k,1:k}$ is nonsingular $\leftarrow$ A has an LU factorization :
This problem can be solved by induction:
Step 1:

When k = 1: $A_{11} = [1][a_{11}]$

Since A_{11} is non-singular, we know that a_{11} is not zero

Step 2:

Assume when k = s, $A_{1:s,1:s} = LU$

Step 3:

When k = s+1:

$$\mathrm{A}_{1:s+1,1:s+1} = \begin{bmatrix} A_{1:s,1:s} & a \\ b & c \end{bmatrix}$$

We can see that when $\mathrm{L}_{new} = \begin{bmatrix} L & 0 \\ bU^{-1} & 1 \end{bmatrix} U_{new} = \begin{bmatrix} L & L^{-1}a \\ 0 & c - bU^{-1}L^{-1}a \end{bmatrix}$

$$A_{1:s+1,1:s+1} = L_{new}U_{new}$$

If we want to show that $L_{new}U_{new}$ is the answer here, we need to show that $U_{22} \neq 0$

As we have seen early,$\det(A_{1:s+1,1:s+1}) = \det(L_{new})\det(U_{new})$

Since both L_{new} and U_{new} are triangular matrices

The determinant of a triangular matrix is just the product of its diagonal entries

Since $\det(A_{1:s+1,1:s+1}) \neq 0$

Then none of the diagonal entries of L_{new} and U_{new} is 0, so $U_{22} \neq 0$

Proof Done

5 Question 5

5.1 Question 5a

We can write A as

$$A = \begin{bmatrix} a_{11} & a_{12} & ..... & a_{1n} \\ a_{21} & a_{22} & ..... & a_{2n} \\ .... \\ a_{n1} & a_{n2} & ..... & a_{nn} \end{bmatrix}$$

Let $A^{-1} =$

$$A^{-1} = \begin{bmatrix} b_{11} & b_{12} & ..... & b_{1n} \\ b_{21} & b_{22} & ..... & b_{2n} \\ .... \\ b_{n1} & b_{n2} & ..... & b_{nn} \end{bmatrix}$$

Such that:

$$AA^{-1} = I$$

To simplify this problem, we can write A^{-1} as

$$A^{-1} = \begin{bmatrix} B_1 & B_2 & ..... & B_n \end{bmatrix}$$

Where

$$B_i = \begin{bmatrix} b_{1i} \\ b_{2i} \\ ..... \\ b_{ni} \end{bmatrix}$$

We can also write I as:

$$I = \begin{bmatrix} I_1 & I_2 & ..... & I_n \end{bmatrix}$$

Where

$$I_i = \begin{bmatrix} I_{1i} \\ I_{2i} \\ ..... \\ I_{ni} \end{bmatrix}$$

To solve A^{-1}, we can do so by solving all the B_is, for each B_i

$$AB_i = I_i$$

This algorithm solves A^{-1} by solving n systems of equations.

An alternative method for calculating the inverse of matrix A is do row operations.

$$[I|A^{-1}] \leftarrow [A|I]$$

5.2 Question 5b

To transform the matrix A to its echelon form, we need

$$\frac{n(n-1)}{2}$$

divisions,

$$\frac{2n^3 + 3n^2 - 5n}{6}$$

multiplications and

$$\frac{2n^3 + 3n^2 - 5n}{6}$$

subtractions. Overall, the cost is $O(n^3)$.

Therefore, if we can save the echelon form of A, then the overall cost is $O(n^3)$.

5.3 Question 5c

To take the advantage of matrix sparsity, we use LU decomposition for inverting a matrix. In this question, we keep using the notations we denoted in Question 5a.

$$A = LU$$

For each B_i and I_i

$$LUB_i = Ii$$

This method is fast, because matrix I is sparse, forward-substitution and backward-substitution can be calculated efficiently. When calculating $Ly = I_i$, before we see the first 1, whenever we see a 0, we can just replace the corresponding entry in y with 0 0 without any calculations. Similar for calculating the inverse of A using row reductions, we can also take the advantage of entries in I being 0, whenever we do a multiplication or addition, if we see a 0, we could simple set the new entry to the original corresponding entry or just 0, since $x + 0 = x$ and $x \times 0 = 0$.

6 Question 6

When A is a triangular matrix, the inverse of A will be dense. If A is a large matrix, A^{-1} would have most of its entries being non-zero. Storing such a matrix could be really expensive.

Besides, calculating the inverse of A requires $O(n^3)$ of time, while $The\ Thomas\ Algorithm$ requires only 8n-7 flops.

7 Question 7

By LU factorization with pivoting, $LU = PA$, therefore

$$L_1 L_2 .... L_{n-1} U = PA$$

$$U = L^{'}_{n-1} L^{'}_{n-2} .... L^{'}_1 PA$$

Since PA is just interchanging rows of A, $\|PA\|_{max} = \|A\|_{max}$

For each $L^{'}_s$

$$\begin{aligned} \|L^{'}_s A\|_{max} &= \max_{i,j} |(L^{'}_s A)_{i,j}| \\ &= \max_{i,j} |\textstyle\sum_{a=1}^{n} (\mathrm{L}^{'}_s)_{ia} a_{ja}\ | \\ &\leq \max_{i,j} |\textstyle\sum_{a=1}^{n} (\mathrm{L}^{'}_s)_{ia}| \max_{i,j} |a_{ij}\ | \\ &= \|L^{'}_s\|_{\infty} \max(\mathrm{A}) \\ &\leq 2\max(\mathrm{A}) \end{aligned} \tag{1}$$

Since there are $n-1$ L, the growth factor ρ has an upper bound of 2^{n-1}

8 Question 8

The randomized matrix is created by:

rand(1000,1)*rand(1,1000)

The run time of the LU decomposition by $Matlab\ lu$ function is around 0.022 seconds, while the run time of the LU decomposition by my own LU function is around 6.187 seconds. Matlab's function runs faster than my code.

Assumption:

The running time of lu algorithm depends on the hardware of the computer and the square of the size of the matrix.

When n = 100:

Time: 0.00025 seconds

When n = 200:

Time: 0.00058 seconds

When n = 400:

Time: 0.00207 seconds

As we can see here, from $n = 100$ to $n = 200$, the time change is around twice, it's not obvious to verify the assumption here. From $n = 200$ to $n = 400$, the size of the matrix increases twice, the runtime increases about 4 times, it maybe a verification for our assumption.