Due on 23 May 2021

M.Sc. Data Science

LAA - Homework 3

1. Let $A \in \mathbb{R}^{n\times n}$ be positive definite. Prove the following:

   (a) For any full rank X in $\mathbb{R}^{n\times n}$, the matrix X^TAX is also positive definite.

   Let us assume, X^TAX is not positive definite. Let, $\mathbf{B} = X^TAX$ So, we can find a nonzero $Z \in \mathbb{R}^n$ such that $Z^TBZ \leq 0$.
   Then, $Z^TBZ = Z^TX^TAXZ = (XZ)^TA(XZ) \leq 0$
   But we already know that A is positive definite. Then by definition $XZ = 0$. But we also know that, X has a full rank n. So, Z must be 0_v.(Contradiction)
   Hence our assumption on X^TAX is not true. X^TAX is positive definite (Proved).

   (b) All principal submatrices of A are also positive definite (a principal submatrix of A is obtained by removing some number of rows and columns of A with the same indices).
   Note that this statement is false for general submatrix of A (as was stated earlier). Positive definiteness (and semi-definiteness) are properties tied with the diagonal elements of A, so the statement is true for any submatrix that shares its diagonal with a subset of the diagonal of A, i.e. a principal submatrix.

   Let's take a k element index set $v = \{v_1, v_2, v_3, \ldots, v_k\}$ such that $1 \leq v_1 \leq v_2 \leq v_3 \cdots \leq v_k \leq n$. We also take a $n \times n$ Identity Matrix and we are applying indexing on the Identity Matrix. Let I_v means the columns of the identity matrix indexed by v.
   From the rules of matrix multiplication we can get the principle submatrices of A. Let any such submatrix is, ${I_v}^TAI_v$. From the previous theorem, for any full column rank $X \in \mathbb{R}^{n\times n}$, X^TAX is positive definite. Here I_v also has full column rank. So, ${I_v}^TAI_v$ is also positive definite. Hence, all the submatrices of A are positive definite.

   (c) A is positive definite if and only if its symmetric part $T = \left(\dfrac{A+A^T}{2}\right)$ is positive definite. Is this true for $A \in \mathbb{C}^{n\times n}$? - if true prove it, if false provide counter-example.

   Proving the forward direction :

   It is clear that, $x^TAx = x^TTx$ for any non zero $x \in \mathbb{R}^n$. Let one of the eigen values of T is λ. Then $Tx = \lambda x$.
   Replacing Tx with λx, we get, $x^TAx = x^T\lambda x = \lambda x^Tx$. As, $x^Tx > 0$ for Real Vectors and A is positive definite, then λ is also positive.

   Proving the backward direction :

   Let, T has positive eigen values. So, we can write it in terms of Schur decomposition as,

$T = PDP^T$. Where D is the diagonal matrix consisting of all the eigen values of T in it's diagonal. $D = \text{diag}(\lambda_i)$ for all diagonal elements.So, $P^TTP = D$. Let, $x \in \mathbb{R}^n$ and $y = P^Tx$ then,

$$\begin{aligned} & x^TAx = x^TTx && (1)\\ \implies & x^TTx = x^T(PDP^T)x && (2)\\ \implies & x^T(PDP^T)x = (P^Tx)^TD(P^Tx) && (3)\\ \implies & (P^Tx)^TD(P^Tx) = y^TDy && (4)\\ \implies & y^TDy = \sum_{i=1}^{n} \lambda_i {y_i}^2 > 0 && (5) \end{aligned}$$

So, each λ_i is positive. (Proved)

Providing counter example for Only-if direction :
Now if $A \in \mathbb{C}^{n\times n}$, then the result might not hold true. We will show it by a counter example.
Say, $A = \begin{pmatrix} 1 & i \\ -i & 1 \end{pmatrix}$, So, $\det(A) = 1 + i^2 = 0$. So, A is not positive definite as it is singular.
But $T = A + A^T = \frac{1}{2}\left[\begin{pmatrix} 1 & i \\ -i & 1 \end{pmatrix} + \begin{pmatrix} 1 & -i \\ i & 1 \end{pmatrix}\right] = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$
Here T is positive definite as, $X^TTX = X^TX > 0$

(d) A has a LU factorization with diagonal entries of U being positive.

From the proof 1.(b), All principal submatrices of A are also positive definite. And another theorem states that, If $A \in \mathbb{R}^{n\times n}$, and $\det(A(1:k, 1:k)) \neq 0$ for $k = 1 : n-1$, then there exists a unit lower triangular $L \in R^{n\times n}$ and an upper triangular $U \in R^{n\times n}$ such that $A = LU$. So, let $x = (L^{-1})^T = L^{-T}$.Then $B = X^TAX = L^{-1}(LU)L^{-T} = UL^{-T}$ is positive definite and has positive diagonal entries. As L^{-T} is unit upper triangular and this implies, the diagonal entries are positive.

---

2. Show that the singular values of a symmetric matrix are the absolute values of its eigenvalues.

Let A be a $n \times n$ symmetric matrix. We know that, if we apply SVD on A we get $A = U\Sigma V^T$. Where $\Sigma = \begin{pmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n \end{pmatrix}$ where $\sigma_1 > \sigma_2 > \cdots > \sigma_n$.

It is given that A is a symmetric matrix, then $A^T = A$. So, u

$$
\begin{aligned}
A &= U\Sigma V^T && (6)\\
\implies \quad A^T &= U\Sigma V^T && (7)\\
\implies AA^T &= U\Sigma V^T(U\Sigma V^T)^T && (8)\\
\implies \quad A^2 &= U\Sigma^2 U^T && (9)
\end{aligned}
$$

Hence, We can say that, Σ^2 is the diagonal matrix of the eigen values of A^2 in the diagonals. We aleady know that, The diagonal elemts of Σ are the square root of the eigen values of A^TA or AA^T. So, in the symmetric case, the singular values will be $\sqrt{\lambda_1^2}, \sqrt{\lambda_2^2}, \ldots, \sqrt{\lambda_n^2}$. Which is equivalent to $|\lambda_1|, |\lambda_2|, \ldots, |\lambda_n|$.

3. Banded matrices: for the following exercises, you may refer to Section 4.3 of G-L. You are encouraged to explore other topics in this section.

   Let A be a banded matrix with lower bandwidth p and upper bandwidth q. Prove the following:

   (a) LU factorization preserves the bandwidth of a banded matrix: if $A = LU$ is the LU factorization of A, then the lower bandwidth of L is p and the upper bandwidth of U is q.

   We will solve it by induction. Since,
   $A = \begin{pmatrix} \alpha & w^T \\ v & B \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{v}{\alpha} & I_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & B - \frac{vw^T}{\alpha} \end{pmatrix} \begin{pmatrix} \alpha & w^T \\ 0 & I_{n-1} \end{pmatrix}$
   It is clear that, $B - \frac{vw^T}{\alpha}$ has upper bandwidth q and lower bandwidth p because there the first q components of w and first p components of v are nonzero. Let L_1U_1 is the LU factorization of this matrix. Using induction and sparcity of w and v, it follows that,
   $L = \begin{pmatrix} 1 & 0 \\ \frac{v}{\alpha} & L_1 \end{pmatrix}$, $\quad U = \begin{pmatrix} \alpha & w^T \\ 0 & U_1 \end{pmatrix}$ have teh desired bandwidth properties and satisfy $A = LU$.

   (b) Compare the Gaussian elimination algorithm for general matrices (Algorithm 3.4.1) and for banded matrices (Algorithm 4.3.1). What are the differences between the two algorithms? Derive the operation count for Algorithm 4.3.1.

   In **Gaussian Elimination Algorithm** for general matrix we did partial pivoting and if the pivot element $A(k,k) \neq 0$, then we did general Gaussian Elimination operation. where

```
p = k+1 : n
    A(p,k) = A(p,k)/A(k,k)
    A(p,p) = A(p,p) - A(p,k).A(k,p)
```

   Here in this algorithm for $i = 1:n$, $A(i, i:n)$ is overwritten by $U(i, i:n)$ and $A(i+1:n, i)$ is overwritten by $L(i+1:n, i)$.

   But in **Banded Matrix Algorithm**, we didn't do any partial pivoting. If $A(k,k) \neq 0$, then we did general Gaussian Elimination operations. But those operations are done only on the banded part of the matrix A as rest of the elements of A are zero.

   Here the operations are,

```
     for i = k+1 : min{k+p , n}
         A(i,k) = A(i,k)/A(k,k)
    end
    for j = k+1 : min{k+q , n}
        for i = k+1 : min{k+p , n}
            A(i,j) = A(i,j) - A(i,k).A(k,j)
        end
    end
```

Here P & Q are lower and upper bandwidth. The basic difference between the 2 algorithms is,

- For Banded Matrix Algorithm, we didn't do any partial pivoting
- For Gaussian Elimination Algorithm, we did partial pivoting.

As in Banded Matrix Algorithm, we compute LU of band matrix, so, operations are done only on the banded part of this matrix. That is why number of operation count is lesser than the number of operation counts in Gaussian Elimination Algorithm.

Number of operation count for Banded Matrix Algorithm:
Number of operations in $k+1 : min\{k+p,n\} \approx p$
Number of operations in $k+1 : min\{k+q,n\} \approx q$
So, the total number of operation count for Banded Matrix Algorithm is,

$$(n-1)(p+2pq) = np + 2npq - p - 2pq \approx 2npq \tag{10}$$

---

4. Read and understand Sections 3.4.6 and 3.4.7 of G-L on complete pivoting and rook pivoting. Demonstrate the algorithms on a 4×4 matrix example. Compare the operation counts and comment on the stability of each.

Complete Pivoting :

According to the complete pivoting algorithm, It is required to check the modulus of the maximum element of the principle submatrices and then swap the rows and columns accordingly. I have implemented the algorithm in python for better understanding and then verified my solution using the code given below.

```
"""
Created on Sun May 23 21:15:41 2021

@author: shiuli Subhra Ghosh
"""

import numpy as np

def lu_factor(A):
    n = A.shape[0]
    X = np.matrix([[0,0,0,0]])

    for k in range(n-1):

        # piv
        index = np.where(abs(A[k:n,k:n]) == np.amax(abs(A[k:n,k:n])))
        ind = list(zip(index[0],index[1]))
        max_row_index = ind[0][0] + k
        max_col_index = ind[0][1] + k
        print(max_row_index ,max_col_index)
        A[[k,max_row_index]] = A[[max_row_index,k]]
        A[:,[k, max_col_index]] = A[:,[max_col_index,k]]
        print('The change of matrix is')
        print(A, max_row_index, max_col_index,k)


        # LU
        if A[k,k] == 0:
            return
        for i in range(k+1,n):
            A[i,k]=A[i,k]/A[k,k]
        for j in range(k+1,n):
            for i in range(k+1,n):
                A[i,j]-=A[i,k]*A[k,j]
            print('row operated matrix is :')
            print(A)

    return(np.copy(A))

A1=np.matrix([[1.0,2,5.0,3.0],[7.0,1.0,2.0,.0],[3.0,9.0,4.0,-11],[4.0,-5.0,8.0,7.0]])
mat = lu_factor(A1)

n = A1.shape[0]
l = np.zeros((n,n))
u = np.zeros((n,n))
for i in range(n):
    for j in range(n):
        if i <= j:
            u[i,j] = mat[i,j]
        if i >= j:
            l[i,j] = mat[i,j]
print(l,u)
```

Now I will explain the solution step by step what the code or the algorithm is doing.

$$\begin{pmatrix} 1 & 2 & 5 & 3 \\ 7 & 1 & 2 & 0 \\ 3 & 9 & 4 & -11 \\ 4 & -5 & 8 & 7 \end{pmatrix} \xrightarrow[C_4 \leftrightarrow C_1]{R_3 \leftrightarrow R_1} \begin{pmatrix} -11 & 9 & 4 & 3 \\ 0 & 1 & 2 & 7 \\ 3 & 2 & 5 & 1 \\ 7 & -5 & 8 & 4 \end{pmatrix}$$ (As the maximum element in the 4×4 array is -11 and the index of that element in the matrix is (3,4))

$$\begin{pmatrix} -11 & 9 & 4 & 3 \\ 0 & 1 & 2 & 7 \\ 3 & 2 & 5 & 1 \\ 7 & -5 & 8 & 4 \end{pmatrix} \rightarrow \begin{pmatrix} -11 & 9 & 4 & 3 \\ 0 & 1 & 2 & 7 \\ -0.2727 & 4.454 & 6.0909 & 1.818 \\ -0.636 & 0.727 & 10.5454 & 5.909 \end{pmatrix}$$ (first column pivot is fixed at -11 and applying row column operations)

Next, again we need to change the pivot rows and columns, because the absolute value maximum element in the square principle submatrix indexed by [2:4, 2:4] is 10.5454..

$$\begin{pmatrix} -11 & 9 & 4 & 3 \\ 0 & 1 & 2 & 7 \\ -0.2727 & 4.454 & 6.0909 & 1.818 \\ -0.636 & 0.727 & 10.5454 & 5.909 \end{pmatrix} \xrightarrow[C_3 \leftrightarrow C_2]{R_3 \leftrightarrow R_4} \begin{pmatrix} -11 & 4 & 9 & 3 \\ -0.6363 & 10.54545 & 0.7272 & 5.90909 \\ -0.2727 & 6.0909 & 4.454545 & 1.8181 \\ 0 & 2 & 1 & 7 \end{pmatrix}$$

Again applying row column operations,

$$\begin{pmatrix} -11 & 4 & 9 & 3 \\ -0.6363 & 10.54545 & 0.7272 & 5.90909 \\ -0.2727 & 6.0909 & 4.454545 & 1.8181 \\ 0 & 2 & 1 & 7 \end{pmatrix} \rightarrow \begin{pmatrix} -11 & 4 & 9 & 3 \\ -0.6363 & 10.54545 & 0.7272 & 5.90909 \\ -0.2727 & 0.577 & 4.034482 & -1.59482 \\ 0 & 0.189 & 0.8620 & 5.8793 \end{pmatrix}$$

Again applying the change of rows and columns, as the maximum element if the submatrix indexed by [3:4,3:4] is 5.8793.

$$\begin{pmatrix} -11 & 4 & 9 & 3 \\ -0.6363 & 10.54545 & 0.7272 & 5.90909 \\ -0.2727 & 0.577 & 4.03448 & -1.5948 \\ 0 & 0.189 & 0.8620 & 5.879 \end{pmatrix} \xrightarrow[C_3 \leftrightarrow C_4]{R_3 \leftrightarrow R_4} \begin{pmatrix} -11 & 4 & 9 & 3 \\ -0.636 & 10.54545 & 0.7272 & 5.90909 \\ 0 & 0.189 & 5.879 & 0.8620 \\ -0.2727 & 0.577 & -1.594 & 4.0344 \end{pmatrix}$$

Finally the last row operation,

$$\begin{pmatrix} -11 & 4 & 9 & 3 \\ -0.636 & 10.54545 & 0.7272 & 5.90909 \\ 0 & 0.189 & 5.8793 & 0.8620 \\ -0.2727 & 0.577 & -1.594 & 4.03448 \end{pmatrix} \rightarrow \begin{pmatrix} -11 & 4 & 9 & 3 \\ -0.6363 & 10.54545 & 0.7272 & 5.90909 \\ 0 & 0.189 & 5.8793 & 0.8620 \\ -0.2727 & -0.577 & -0.271 & 4.2683 \end{pmatrix}$$

$$U = \begin{pmatrix} -11 & 4 & 9 & 3 \\ 0 & 10.54545 & 0.7272 & 5.90909 \\ 0 & 0 & 5.8793 & 0.8620 \\ 0 & 0 & 0 & 4.2683 \end{pmatrix} \qquad L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -0.6363 & 1 & 0 & 0 \\ 0 & 0.189 & 1 & 0 \\ -0.2727 & -0.577 & -0.271 & 1 \end{pmatrix}$$

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

So, $PAQ = LU$.

Flops calculation for Complete Pivoting :

k	i	add-sub flops	mult-div flops	Comparison
1	2:n = $(n-1)$ rows	$(n-1)\times n$	$(n+1)\times(n-1)$	$n\times n$
2	3:n = $(n-2)$ rows	$(n-2)\times(n-1)$	$n\times(n-2)$	$(n-1)\times(n-1)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(n-1)$	n:n = 1 rows	1×2	1×3	2×2

Now, add the add-sub flops.

$\sum_{k=1}^{n-1}(n-k)(n+1-k) = \sum_{k=1}^{n-1}(n^2+n-2nk-k+k^2) = \frac{1}{3}n^3 - \frac{1}{3}n$

Now add the mult-div flops

$\sum_{k=1}^{n-1}(n-k)(n+2-k) = \sum_{k=1}^{n-1}(n^2+2n-2nk+2k+k^2) = \frac{1}{3}n^3 + \frac{5}{2}n^2 - \frac{17}{6}$

Now add the comparisons for the each iterations

$n^2+(n-1)^2+\cdots+2^2 = n^2+(n-1)^2+\cdots+2^2+1^2-1 = \frac{1}{3}n^3+\frac{1}{2}n^2+\frac{1}{6}n-1$

So, we can say that the total flops count for complete pivoting is $\frac{2}{3}n^3 + \mathcal{O}(n^3)$

Gaussian Elimination with Complete pivoting is stable

Rook Pivoting :

In the Rook Pivoting, we usually select the maximum element from the row and the column indexed by same number. Here I have also written a code for the Rook Pivoting algorithm and verified my solution using the code.

```
"""
Created on Sun May 23 19:44:24 2021

@author: shiuli Subhra Ghosh
"""
import numpy as np

def lu_factor(A):

    n = A.shape[0]
    X = np.matrix([[0,0,0,0]])
    piv_row = np.arange(0,n)
    piv_col = np.arange(0,n)
    for k in range(n-1):

        # piv
        max_row_index = np.argmax(abs(A[k:n,k])) + k
        max_col_index = np.argmax(abs(A[max_row_index,k:n])) + k
        piv_row[[k,max_row_index]] = piv_row[[max_row_index,k]]
        piv_col[[k,max_col_index]] = piv_col[[max_col_index,k]]
        A[:,[k, max_col_index]] = A[:,[max_col_index,k]]
        A[[k,max_row_index]] = A[[max_row_index,k]]
        print('The change of matrix is')
        print(A, max_row_index, max_col_index)

        # LU
        if A[k,k] == 0:
            return
        for i in range(k+1,n):
            A[i,k]=A[i,k]/A[k,k]
        for j in range(k+1,n):
            for i in range(k+1,n):
                A[i,j]-=A[i,k]*A[k,j]
            print('row operated matrix is :')
            print(A)

    return [A,piv_row, piv_col]

A=np.matrix([[8.0,0,7.0,5.0],[0.0,4.0,6.0,2.0],[0.0,8.0,2.0,2.0],[4.0,4.0,2.0,2.0]])

mat = lu_factor(A)

n = A.shape[0]
l = np.zeros((n,n))
u = np.zeros((n,n))
for i in range(n):
    for j in range(n):
        if i <= j:
            u[i,j] = mat[i,j]
        if i >= j:
            l[i,j] = mat[i,j]
print(l,u)
```

Now I will explain the algorithm step by step using a 4×4 matrix.

$$\begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 4 & 6 & 2 \\ 0 & 8 & 2 & 2 \\ 4 & 4 & 2 & 2 \end{pmatrix} \longrightarrow \begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 4 & 6 & 2 \\ 0 & 8 & 2 & 2 \\ 4 & 4 & 2 & 2 \end{pmatrix}$$
(As the maximum element in the 1st row and column is 8 and the index of that element in the matrix is (1,1). So, no change)

Now, making the first row operations,

$$\begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 4 & 6 & 2 \\ 0 & 8 & 2 & 2 \\ 4 & 4 & 2 & 2 \end{pmatrix} \rightarrow \begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 4 & 6 & 2 \\ 0 & 8 & 2 & 2 \\ 0.5 & 4 & -1.5 & -0.5 \end{pmatrix}$$

Now, again finding the largest absolute value in the 2^{nd} row and column of the principle submarix and make the swap operation.

$$\begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 4 & 6 & 2 \\ 0 & 8 & 2 & 2 \\ 0.5 & 4 & -1.5 & -0.5 \end{pmatrix} \xrightarrow{R_3 \leftrightarrow R_2} \begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 8 & 2 & 2 \\ 0 & 4 & 6 & 2 \\ 0.5 & 4 & -1.5 & -0.5 \end{pmatrix}$$ (As the maximum element in the 2nd row and column is 8 and the index of that element in the matrix is (3,2))

Again, making the necessary row operations,

$$\begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 8 & 2 & 2 \\ 0 & 4 & 6 & 2 \\ 0.5 & 4 & -1.5 & -0.5 \end{pmatrix} \rightarrow \begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 8 & 2 & 2 \\ 0 & 0.5 & 5 & 1 \\ 0.5 & 0.5 & -2.5 & -1.5 \end{pmatrix}$$

Next, there won't be any swap operations because the largest element in the 3rd row and column principle submatrix is 5 and its position is (3,3). So, in the 3rd loop, swap operations won't happen. Then it will directly move to the last row operation.

$$\begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 8 & 2 & 2 \\ 0 & 0.5 & 5 & 1 \\ 0.5 & 0.5 & -2.5 & -1.5 \end{pmatrix} \rightarrow \begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 8 & 2 & 2 \\ 0 & 0.5 & 5 & 1 \\ 0.5 & 0.5 & -0.5 & -1 \end{pmatrix}$$

$$U = \begin{pmatrix} 8 & 0 & 7 & 5 \\ 0 & 8 & 2 & 2 \\ 0 & 0 & 5 & 1 \\ 0 & 0 & 0 & -1 \end{pmatrix} \qquad L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0.5 & 1 & 0 \\ 0.5 & 0.5 & -0.5 & 1 \end{pmatrix}$$

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

So, $PAQ = LU$.

Flops calculation for Rook Pivoting :

k	i	add-sub flops	mult-div flops	Comparison
1	2:n = $(n-1)$ rows	$(n-1)\times n$	$(n+1)\times(n-1)$	$2n-1$
2	3:n = $(n-2)$ rows	$(n-2)\times(n-1)$	$n\times(n-2)$	$2n-3$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(n-1)$	n:n = 1 rows	1×2	1×3	3

Now, add the add-sub flops.

$\sum_{k=1}^{n-1}(n-k)(n+1-k) = \sum_{k=1}^{n-1}(n^2+n-2nk-k+k^2) = \frac{1}{3}n^3 - \frac{1}{3}n$

Now add the mult-div flops

$\sum_{k=1}^{n-1}(n-k)(n+2-k) = \sum_{k=1}^{n-1}(n^2+2n-2nk+2k+k^2) = \frac{1}{3}n^3 + \frac{5}{2}n^2 - \frac{17}{6}$

Now add the comparisons for the each iterations

$2n-1+2n-3+\cdots+3 = n^2-1$

So, we can say that the total flops count for rook pivoting is $\frac{2}{3}n^3 + \mathcal{O}(n^2)$

The Rook Pivoting is also as stable as Complete Pivoting for Gaussian Elimination.