APPM 4720/5720: HW#1 prob. 2

a) CPQR Given Tolerance

We aim to construct a function that calculates the CPQR of a given matrix to a desired accuracy, let's call such a function: CPQR_given_tolerance. The inputs are an m x n target matrix $\mathbf{A}$ and a tolerance, acc. The output will be a $\mathbf{QR}$ approximation of $\mathbf{A}$ along with an index vector ind such that $||\mathbf{A}(:, ind) - \mathbf{QR}||_F \leq acc$. To do this, we take advantage of a previously given function that calculates the rank k CPQR of $\mathbf{A}$, CPQR_given_rank. The proposed function follows this procedure:

Step 1: Initiate calculation of the rank $p = min(m, n)$ CPQR of $\mathbf{A}$ (using CPQR_given_rank)
Step 2: For each vector added to $\mathbf{Q}$ (say the j^{th}) compare the residual to acc (check: $||\mathbf{Q}(:, j+1:n)||_F$).
Step 3: Exit calculation if $||\mathbf{Q}(:, j+1:n)||_F \leq acc$ and return $\mathbf{Q}$, $\mathbf{R}$, ind.

Notes: The rank of $\mathbf{A}$ cannot exceed $p = min(m, n)$, as such the rank of our approximation should be less than or equal to p. Any matrix norm may be used to check error. We used the Frobenius norm because of its simplicity to calculate along with the resulting conservative estimates. During calculation, CPQR_given_rank overwrites the input $\mathbf{A}$ with $\mathbf{Q}$, revised iterates of the original matrix are stored in columns not yet orthonormalized hence why $||\mathbf{Q}(:, j+1:n)||_F$ is relevant (for a more concise, informative description, please see the course notes). There are many ways to implement this procedure, some better than others. We strive to minimize the movement of data and rely on BLAS2 and BLAS3 operations (see problem 3) as well as FLOP count. One could also use Householder relections or Givens rotations to calculate the QR factorization. A good discussion of this is given in Section 5.2 of *Matrix Computations, 4th edition* by Gene Golub and Charles Van Loan.

b) SVD Given Tolerance

We aim to retrieve the partial SVD of $\mathbf{A}$ to a given accuracy. The construction of this function relies on part (a). A nifty property of the error term from our $\mathbf{QR}$ factorization allows us to avoid needless computation.

Recall:
$\mathbf{A} = \mathbf{QRP}^* + (Error)\mathbf{P}^* \Rightarrow \mathbf{A} \approx \mathbf{QRP}^* + \mathbf{E}$, where $||\mathbf{E}||_F \leq acc$. (Since the inverse of a unitary matrix is equal to its adjoint, $\mathbf{P}^* = \mathbf{P}^{-1}$)

The SVD of $\mathbf{RP}^*$ is then calculated and $\mathbf{Q}$ is right multiplied by the left singular vectors of $\mathbf{RP}^*$. This does not change $\mathbf{E}$ in the approximation. Our program follows this procedure:

Step 1: Calculate CPQR to given specified tolerance using (a)
Step 2: Calculate the inverse permutation matrix $\mathbf{P}^*$
Step 3: Find the SVD of $\mathbf{RP}^*$, this forms $\hat{\mathbf{U}}$,$\mathbf{D}$,$\mathbf{V}$
Step 4: Matrix-Matrix multiply $\mathbf{Q}$ by $\hat{\mathbf{U}}$ (from the SVD in step 3), this forms $\mathbf{U}$
Step 5: Return $\mathbf{U}$,$\mathbf{D}$,$\mathbf{V}$

Notes: Be careful to calculate the adjoint matrix of $\mathbf{V}$ when required ($\mathbf{A} \approx \mathbf{UDV}^*$, $\mathbf{A} \not\approx \mathbf{UDV}$). To test your program, generate some large matrices and play with accuracy levels. It can be fun to watch how large, ill-conditioned (constructed to have a high condition number) matrices interact with the algorithms.