Rook Pivoting

We consider a matrix A of size $n > 0$, and rank $0 < r \leq n$.

We assume that we are able to execute r steps of the rook pivoting algorithm. Moreover, when iterating over rows and columns to find the pivot, we assume that we converge with O(1) (i.e., a few) iterations. This is the case in practice. The worst case scenario is different but it can be ignored for this homework.

Implement your $O(r^2 n)$ algorithm in Julia. The starter code is given below. For a matrix of size 64 and a rank of 32, report on the 2-norm of the difference between A and $P^{-1}LUQ^{-1}$ using your algorithm. The matrices L and U^T should be lower triangular and should have only r columns.

In [ ]:
```
using Random
using Printf
using LinearAlgebra

# Initialize the random number generator
rng = MersenneTwister(2018)
```

In [ ]:
```
function getrfRook!(A)
    n = size(A,1)
    P_row = collect(1,n); P_col = collect(1,n)
    for k = 1:n
        # LU only on kth col of the kth step
        if k == 1
            # simply start searching for the biggest
            row = 1; row0 = 0; col = 1; col0 = 0
            while row != row0 || col != col0
                row0, col0 = row, col
                row_A = abs.(A[row+k-1, k:end])
                col = findmax(row_A)[2] #biggest ele in row
                col_A = abs.(A[k:end, col+k-1])
                row = findmax(col_A)[2]
            end
            # swap and update values
            row += k - 1; col += k - 1
            P_row[k], P_row[row] = P_row[row], P_row[k]
            P_col[k], P_col[col] = P_col[col], P_col[k]
            for j = 1:n
                A[k,j], A[row,j] = A[row,j], A[k,j]
            end
            for i = 1:n
```

```
                A[i,k], A[i,col] = A[i,col], A[i,k]
            end
            #LU on col 1
            for i = k+1:n
                A[i,k] /= A[k,k]
            end
        #otherwise, for steps after step one
        else
            #firstly update the kth diagonal entry
            for i = 1:k-1
                A[k,k] -= A[k,i]*A[i,k]
            end

            row = 1; row0 = 0; col = 1; col0 = 0
            array = zeros(n-k,1)
            while row != row0 || col != col0
                row0,col0 = row, col
                array = A[row+k-1, k:n]
                #update
                for m = k:n
                    array[m-k] -= A[row+k-1]*A[k,m]
                end
                row_A = abs.array
                col = findmax(row_A)[2]

                array = A[k:n, col+k-1]

                #update
                for m = k:n
                    array[m-k] -= A[m,k]*A[k,col+k-1]
                end
                A[i,j] -= A[i,k]*A[k,j]
                col_A = abs.array
                row = findmax(col_A)[2]
            end

            #once located all index number of swap, perform swapping
            P_row[k], P_row[row] = P_row[row], P_row[k]
            P_col[k], P_col[col] = P_col[col], P_col[k]
            for j = 1:n
                A[k,j], A[row,j] = A[row,j], A[k,j]
            end
            for i = 1:n
                A[i,k], A[i,col] = A[i,col], A[i,k]
            end

            #LU on kth column
            for i = k+1:n
                A[i,k]/=A[k,k]
            end
        end
    end
end
```

Initialization

In [ ]:
```
n = 64 # Size of matrix
r = 32 # Rank

A = rand(rng,n,n)

F = lu(A)    # LU factorization
L = F.L      # Lower triangular part
U = F.U      # Upper triangular part
L = L[:,1:r] # Keep the first r columns
U = U[1:r,:] # Keep the first r rows


A = L*U      # Rank r matrix
A0 = copy(A) # Save a copy
;
```

Factorization

In [ ]:
```
P_row, P_col = getrfRook!(A0)
```

Test

In [ ]:
```
L0 = UniformScaling(1.0) + tril(A0,-1) # Extract L matrix
U0 = triu(A0) # Extract U matrix

L = zeros(n,r)
for i=1:n
    L[P_row[i],:] = L0[i,1:r] # Undo the row permutations
end

U = zeros(r,n)
for j=1:n
    U[:,P_col[j]] = U0[1:r,j] # Undo the column permutations
end

err = norm(L*U - A)
@printf "The error is %g" err # Test the accuracy
err < 1e-13 ? "PASS" : "FAIL"
```

In [ ]:
```

```