Practical assignment 4. Gram-Schmidt and QR-factorization

1) **Gram-Schmidt process for QR-factorization.** Write a Matlab function `qr_gs` which can be used as `[Q,R]=qr_gs(A)` and, for a given complex (and real) matrix A of the size m×n (m≥n) returns an m×n matrix Q with orthonormal columns and an n×n upper triangular matrix R such that $A = QR$ (i.e. thin, or reduced, QR factorization). Use modified Gram-Schmidt algorithm (see the algorithm below, here $X = A$).

- The QR factorization should be obtained by the Gram-Schmidt process applied to the columns of A.
- You can start with the following commands in your function

```
[m,n]=size(A);
if m<n
    error('must be m>=n')
end
Q=zeros(m,n);
R=zeros(n);

for j=1:n
  Q(:,j)=A(:,j);
end
```

  - Check that your function works correctly: 1) it produces QR-factorization of A: $A = QR$, 2) matrix Q is orthonormal and matrix R is upper-triangular.
  - Answer the question. Do the commands

```
[Q1,R1]=qr(A);
[Q2,R2]=qr_gs(A);
```

  produce the same result? Should they?
- How can the call to `qr` be modified so that it does produce the same result as `qr_gs`?
- Check what happens with Gram-Schmidt process, when some columns of the matrix are linear dependent. Compare with Matlab `qr` command.

ALGORITHM 1.1 *Gram-Schmidt*

1. Compute $r_{11} := \|x_1\|_2$. If $r_{11} = 0$ Stop, else compute $q_1 := x_1/r_{11}$.
2. For $j = 2, \ldots, r$ Do:
3. Compute $r_{ij} := (x_j, q_i)$, for $i = 1, 2, \ldots, j-1$
4. $\hat{q} := x_j - \sum_{i=1}^{j-1} r_{ij} q_i$
5. $r_{jj} := \|\hat{q}\|_2$,
6. If $r_{jj} = 0$ then Stop, else $q_j := \hat{q}/r_{jj}$
7. EndDo

ALGORITHM 1.2 *Modified Gram-Schmidt*

Define $r_{11} := \|x_1\|_2$. If $r_{11} = 0$ Stop, else $q_1 := x_1/r_{11}$.
For $j = 2, \ldots, r$ Do:
  Define $\hat{q} := x_j$
  For $i = 1, \ldots, j-1$, Do:
    $r_{ij} := (\hat{q}, q_i)$
    $\hat{q} := \hat{q} - r_{ij}q_i$
  EndDo
  Compute $r_{jj} := \|\hat{q}\|_2$,
  If $r_{jj} = 0$ then Stop, else $q_j := \hat{q}/r_{jj}$
EndDo

2) **Different QR-factorizations.** Try the following Matlab commands

```
A=rand(4,2), I=eye(4);
[Q,R]=qr(A)
A1=[A,I(:,3:4)], [Q1,R1]=qr(A1)
norm(A-Q*R,1), norm(A-Q1*R1(:,1:2),1) %R=R1(:,1:2)
```

Check that you have computed two different QR-factorizations of A.

3) **Solving linear system using QR-factorization.** Consider a linear system with an n×n random matrix A: $Ax = b$, when n=1000, 2000, 3000 and above.

- Generate these systems with the following Matlab commands:

```
n=1000;
A=rand(n,n); xexact=rand(n,1); b=A*xexact
```

- Solve the system in the following ways, reporting, for each case, the CPU time required, norm of the absolute error `norm(x-xexact,2)` and residual norm of the solution `r=b-A*x; norm(r,2)`
  1. Backslash or `mldivide: tic x=A\b; toc`
  2. QR-decomposition, using A=QR:
  a) `tic [Q,R]=qr(A); x=R\(Q'*b); toc`
  b) `[Q,R]=qr(A); tic x=R\(Q'*b); toc %neglect time to get QR`

- Explain the differences in the CPU times you observe for different solution methods as n increases. When do you obtain the smallest absolute error and residual? What are the advantages and disadvantages of each of the methods used to solve the linear systems?