Math 5553, Homework 3, Due on 3/13/2012

1. (12 points) Matlab command `rand(m,n)` generates an $m \times n$ random matrix. Here we always assume $m \geq n$. To get a random $m \times m$ unitary matrix, one can simply use `[Q,R] = qr(rand(m,n))`, where `qr` is the Matlab built-in routine for QR factorization, or `[Q, S, V] = svd(rand(m,n))`, where `svd` is the Matlab built-in routine for SVD. Then, `Q` returns an $m \times m$ unitary matrix.

   Now we can manually write an upper triangular matrix R and set $A = QR$. This gives us control of the diagonal entries of R in the QR factorization of A. An easy way to write an upper-triangular matrix is to use `diag(a) * triu(ones(m,n))`, where $a = [a_1, a_2, \dots, a_m]$ is an m-dim vector. This generates an $m \times n$ matrix which contains the first n columns of the upper triangular matrix

$$\begin{bmatrix} a_1 & a_1 & a_1 & \cdots & a_1 \\ 0 & a_2 & a_2 & \cdots & a_2 \\ 0 & 0 & a_3 & \cdots & a_3 \\ & & & \cdots & \\ 0 & 0 & 0 & \cdots & a_m \end{bmatrix}$$

   In the following experiment, we will set $a_i = 2^{-i}$, for $i = 1, ...m$.

   Matrix A can be used to test the stability of different methods for computing the QR factorization. We have covered three of them in class, the classical Gram-Schmidt, the modified Gram-Schmidt, and the Householder triangulization algorithms. Implement these three algorithms and test them on matrix A, with $m = n = 48$. For each algorithm, you get the computed QR factorization $\tilde{Q}$ and $\tilde{R}$. Calculate and report the matrix 2-norm of $I - \tilde{Q}^*\tilde{Q}$, using Matlab command `norm`. Notice that if $\tilde{Q}$ is exactly a unitary matrix, this norm should be 0.

   Draw the diagonal entries of $\tilde{R}$ using command `semilogy`. (Plot in discrete dots, circles, x-marks or whatever instead of continuous curve. Type `help plot` to learn more about line type options.) Compare the results with the diagonal entries of R. What do you observe for these three algorithms?

2. (8 points) Suppose a quantity $x(h) \approx ch^r$, where c and r are non-negative constants. In this case we usually say $x = O(h^r)$. For example, when h takes the value of $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}$ and x takes the value 0.803, 0.197, 0.051, 0.0124, we can easily see that r should be around 2. Or in other words, the given set of data satisfies $x(h) = O(h^2)$. The rule is obvious in this case since every time we reduce h by a factor of 2, x will be reduced approximately by a factor of 4.

   However, the value of r is much less obvious if we are given the following set of data:

$$h = \frac{1}{8}, \frac{1}{12}, \frac{1}{16}, \frac{1}{20}, \text{ and } x = 0.185,\ 0.0853,\ 0.0486,\ 0.0313$$

   Plotting this set of data using `loglog(h,x)`. The result is almost a straight line, which indicates that the $\ln h$-$\ln x$ relation is linear. Indeed, by taking nature log of $x \approx ch^r$, we have $\ln x \approx r \ln h + \ln c$. Clearly, for any given set of data, we can then use the least squares method (the line fitting) to find r and c. Compute r and c for the above given data.

   Then, test you algorithm on another set of data:

$$h = \frac{1}{8}, \frac{1}{12}, \frac{1}{16}, \frac{1}{20}, \text{ and } x = 0.12,\ 0.0672,\ 0.044,\ 0.0316$$