MATH 715 - Computational Mathematics II

HW #1

Due 2:30pm, Thursday Feb. 8

1. Download the image `UWlogo.jpg` from the course website. Convert the file to three matrices (R, G, B) using your favorite software. In MATLAB, for instance, you would type
"`A = imread('UWlogo.jpg');`", which results in a three-dimensional array. Each pixel of the image is converted to three 8-bit (1 byte) integers ranging from 0 to 255. For instance, $[0, 0, 0]$ is black, $[255, 0, 0]$ is red, and $[255, 255, 255]$ is white. Be sure you can visualize the full image; in MATLAB, you would type "`image(A)`".

(a) Determine the memory required to store this image with no compression (easy! multiply...).

(b) Use a built-in SVD to decompose each of the three matrices. You may need to convert the matrices to double precision first (`A = double(A)` in MATLAB). To verify completion, reconstruct the original matrix using the SVD's of the three matrices; in Matlab you must revert back to `uint8` to properly view the image. Plot the singular values on a semi-logarithmic scale.

(c) Construct and plot approximating images using only 1, 3, 5, and 50 singular values. For each case indicate the memory required to store the image if you only stored the singular values and their corresponding left- and right-singular vectors.

(Just turn in one page with the above requested plots — it's ok to print in black and white — and corresponding memory costs.)

2. Determine the SVDs for the following matrices (by hand)

$$(a)\ \begin{pmatrix} 3 & 0 \\ 0 & -2 \end{pmatrix}, \quad (b)\ \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}, \quad (c)\ \begin{pmatrix} 0 & 2 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad (d)\ \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}, \quad (e)\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}.$$

3. Suppose $A \in \mathbb{C}^{m\times m}$ has an SVD given by $A = U\Sigma V^*$. Find an eigenvalue decomposition $A = X\Lambda X^{-1}$ of the $2m \times 2m$ Hermitian matrix

$$\begin{pmatrix} 0 & A^* \\ A & 0 \end{pmatrix}.$$

4. If P is an orthogonal projector, then $I - 2P$ is unitary. Prove this algebraically, and give a geometric interpretation.

5. Let $x \in \mathbb{R}^m$, and let E be the $m \times m$ matrix that extracts the "even part" of an m-vector: $Ex = (x + Fx)/2$, where F is the $m \times m$ matrix that flips $(x_1, \ldots, x_m)^T$ to $(x_m, \ldots, x_1)^T$. Is E an orthogonal projector, an oblique projector, or not a projector at all? What are its entries?

6. Let A be a matrix with the property that columns 1, 3, 5, 7... are orthogonal to columns 2, 4, 6, 8,.... In a reduced QR factorization $A = QR$, what special structure does R possess? Assume that A has full rank.

7. Write a function that computes the reduced QR factorization of a matrix $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ using the classical Gram–Schmidt algorithm. Write a second function that computes the same but with the modified Gram–Schmidt algorithm. Follow through Experiment 2 in the handout from T&B at the end of this homework, and reproduce Fig. 9.1 using your code. This time, attach a copy of your code to your homework set.

Experiment 2: Classical vs. Modified Gram–Schmidt

Our second example has more algorithmic substance. Its purpose is to explore the difference in numerical stability between the classical and modified Gram–Schmidt algorithms.

First, we construct a square matrix A with random singular vectors and widely varying singular values spaced by factors of 2 between 2^{-1} and 2^{-80}.

`[U,X] = qr(randn(80));`	Set U to a random orthogonal matrix.
`[V,X] = qr(randn(80));`	Set V to a random orthogonal matrix.
`S=diag(2.^(-1:-1:-80));`	Set S to a diagonal matrix with exponentially graded entries.
`A = U*S*V;`	Set A to a matrix with these entries as singular values.

Now, we use Algorithms 7.1 and 8.1 to compute QR factorizations of A. In the following code, the programs `clgs` and `mgs` are MATLAB implementations, not listed here, of Algorithms 7.1 and 8.1.

`[QC,RC] = clgs(A);`	Compute a factorization $Q^{(c)}R^{(c)}$ by classical Gram–Schmidt.
`[QM,RM] = mgs(A);`	Compute a factorization $Q^{(m)}R^{(m)}$ by modified Gram–Schmidt.

Finally, we plot the diagonal elements r_{jj} produced by both computations (MATLAB code not shown). Since $r_{jj} = \|P_j a_j\|$, this gives us a picture of the size of the projection at each step. The results are shown on a logarithmic scale in Figure 9.1.

The first thing one notices in the figure is a steady decrease of r_{jj} with j, closely matching the line 2^{-j}. Evidently r_{jj} is not exactly equal to the jth singular value of A, but it is a reasonably good approximation. This phenomenon can be roughly explained as follows. The SVD of A can be written in the form (5.3) as

$$A = 2^{-1}u_1v_1^* + 2^{-2}u_2v_2^* + 2^{-3}u_3v_3^* + \cdots + 2^{-80}u_{80}v_{80}^*,$$

where $\{u_j\}$ and $\{v_j\}$ are the left and right singular vectors of A, respectively. In particular, the jth column of A has the form

$$a_j = 2^{-1}\overline{v}_{j1}u_1 + 2^{-2}\overline{v}_{j2}u_2 + 2^{-3}\overline{v}_{j3}u_3 + \cdots + 2^{-80}\overline{v}_{j,80}u_{80}.$$

Since the singular vectors are random, we can expect that the numbers $\overline{v}_{ji}$ are all of a similar magnitude, on the order of $80^{-1/2} \approx 0.1$. Now, when we take the QR factorization, it is evident that the first vector q_1 is likely to be

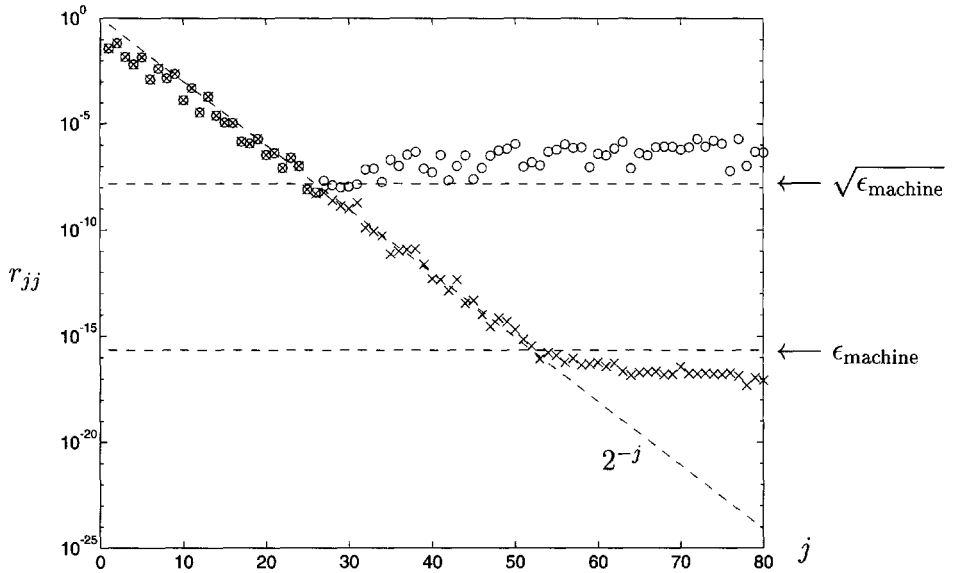

Figure 9.1. *Computed r_{jj} versus j for the QR factorization of a matrix with exponentially graded singular values. On this computer with about* 16 *digits of relative accuracy, the classical Gram–Schmidt algorithm produces the numbers represented by circles and the modified Gram–Schmidt algorithm produces the numbers represented by crosses.*

approximately equal to u_1, with r_{11} on the order of $2^{-1} \times 80^{-1/2}$. Orthogonalization at the next step will yield a second vector q_2 approximately equal to u_2, with r_{22} on the order of $2^{-2} \times 80^{-1/2}$—and so on.

The next thing one notices in Figure 9.1 is that the geometric decrease of r_{jj} does not continue all the way to $j = 80$. This is a consequence of rounding errors on the computer. With the classical Gram–Schmidt algorithm, the numbers never become smaller than about 10^{-8}. With the modified Gram–Schmidt algorithm, they shrink eight orders of magnitude further, down to the order of 10^{-16}, which is the level of *machine epsilon* for the computer used in this calculation. Machine epsilon is defined in Lecture 13.

Clearly, some algorithms are more stable than others. It is well established that the classical Gram–Schmidt process is one of the unstable ones. Consequently it is rarely used, except sometimes on parallel computers in situations where advantages related to communication may outweigh the disadvantage of instability.

Experiment 3: Numerical Loss of Orthogonality

At the risk of confusing the reader by presenting two instability phenomena in succession, we close this lecture by exhibiting another, different kind of