CS 421: Numerical Analysis
Fall 2004
Problem Set 3

Handed out: Wed., Oct. 6.

Due: Mon., Oct. 18.

1. Consider executing k outer loop iterations of left-looking MGS and then stopping. Explain how to extend the columns of Q_1 and R_1 computed so far to $m \times n$ and $n \times n$ matrices respectively, say $\bar{Q}_1$ and $\bar{R}_1$, such that $\bar{Q}_1$ is partly orthogonal (some columns have orthogonality relationships) and $\bar{R}_1$ is upper triangular, and such that $A = \bar{Q}_1\bar{R}_1$. Explain the orthogonality relationships that hold for the columns of $\bar{Q}_1$.

   Then repeat all of this for right-looking MGS.

2. Let A be an $m \times n$ matrix of rank n. Consider the function that maps $\mathbf{b}$, an arbitrary vector in $\mathbf{R}^m$, to $A\mathbf{x}^*$, another vector in $\mathbf{R}^m$, where $\mathbf{x}^*$ is defined to be the minimizer of $\|A\mathbf{x} - \mathbf{b}\|_2$ for the given $\mathbf{b}$.

   This function $\mathbf{b} \mapsto A\mathbf{x}^*$ is linear. Write down its corresponding $m \times m$ matrix (in terms of A). Argue that the matrix is symmetric. Also, show that the if one has a QR-factorization of A, then the matrix defined above depends only on Q (and not on R).

3. An *affine subspace* H of $\mathbf{R}^n$ is defined as a set that can be written in the form $H = \{\mathbf{x} \in \mathbf{R}^n : B\mathbf{x} = \mathbf{c}\}$, where B is a given $k \times n$ matrix of rank k and $\mathbf{c}$ is a given k-vector. (This is the "implicit" form.)

   It turns out that any such an H can also be expressed in the form $H = \{A\mathbf{u} + \mathbf{b} : \mathbf{u} \in \mathbf{R}^{n-k}\}$ where A is some $n \times (n-k)$ matrix of rank $n-k$ and $\mathbf{b}$ is some n-vector. (This is the "parametric" form.)

   Provide an algorithm, based on QR factorization, that takes as input $(B, \mathbf{c})$ and computes $(A, \mathbf{b})$. [Hint: Start with a QR factorization of B^T. Split the resulting Q into k columns and $n-k$ columns.]

4. On the course home page, you will find some jpeg files that contain images of a line. The files are as follows. File `exact.jpg` contains an exact plot of the line. Files `gauss1.jpg`, `gauss4.jpg`, `gauss16.jpg`, and `gauss64.jpg` contain images of the line with Gaussian noise added to the y-coordinate. The standard deviation of the noise is 1 pixel, 4 pixels, 16 pixels, and 64 pixels respectively. The Gaussian noise is clipped at the figure boundaries. Finally, files `outliers001.jpg`, `outliers005.jpg`, `outliers010.jpg`, and `outliers050.jpg` contain the same line, but 0.1%, 0.5%. 1.0%, and 5% (respectively) of the y-coordinates of the pixels have been changed to random data (outliers).

   Write m-files that read these images and try to figure out the slope of the line using a least-squares fit. Your program should print its guess for the slope for each of the

files. It should also plot the points on the same axis with the guessed-at line for each file. You should write at least four m-files: one called `line_lsfit` that takes as input an array of ordered pairs (x_i, y_i) and fits the best least-squares line to this data; one called `fit_jpeg` that processes the jpeg and invokes `line_lsfit` to fit the best line; one called `make_plot` that makes the requested plot; and one called `driver_q4`, a script that answers the question (i.e., computes all the slopes and makes all the plots for the nine examples by invoking the other m-files).

Some hints: for `line_lsfit`, solve the least-squares problem using the Matlab '\' operator. For an example of how to fit a parabola using least squares, see p. 108 of the text. (The problem under consideration, fitting a line, is simpler than fitting a parabola.)

For `fig_jpeg`, use the matlab function `imread` to read the jpeg. (Note: type `help imread` for more information. Note that jpeg files use lossy compression, so in addition to the noise intentionally added to the data, there is noise from jpeg compression.) Read and convert the image to a numerical matrix using the following statements:

```
im = imread(filename);
imr = double(im(:,:,1))+double(im(:,:,2))+double(im(:,:,3));
```

Compute a cutoff as follows: find the maximum entry in imr and also the minimum entry, and set the cutoff to be the average of these two. The data pixels are those that are less than the cutoff. Use as x-coordinates for the data-fitting problem the column index within `imr` of the data pixels, and use as y-coordinates the row within `imr`.

In `make_plot`, use the `cla` command to clear the axes in the current figure window, then display the image using `image(im)` (where `im` is the result of the `imread` function). Follow this with a call to `hold on` to hold the image on the axes, and then use a `plot` command to plot the fitted line.

Hand in: listings of all m-files, the nine requested plots, the nine computed slopes, and a paragraph of comments about how well least-squares performs for the two types of files.