18.085, PROBLEM SET 3, DUE 7/8 IN CLASS

Question 1. (40 pts.) This exercise will show you how to find the Singular Value Decomposition (SVD) of a matrix. Just follow the instructions step by step, no need to refer to your class notes about the SVD!

Recall the SVD of an n by n matrix A has $A = U\Sigma V^T$, where U and V are orthonormal matrices and Σ is a diagonal matrices, with non-negative entries σ_j, $j = 1, \ldots, n$. (Whatever below is in parentheses and small font is for your information - you do not have to prove these things, but hopefully you can understand them.) Let

$$A = \begin{pmatrix} 1 & 4 \\ 2 & 8 \end{pmatrix}.$$

a) Show, *using the definition of the SVD decomposition* of A, (that is, using *only* the fact that $A = U\Sigma V^T$ with U and V orthonormal and Σ diagonal) that $M = A^TA = V\Sigma^2V^T$. (In fact, the eigenvalue decomposition of the matrix $M = A^TA$ *is* $M = V\Sigma^2V^T$. Indeed, multiplying M by any column v_j of V, we get $Mv_j = V\Sigma^2V^Tv_j = V\Sigma^2e_j = V\sigma_j^2e_j = \sigma_j^2v_j$ (where e_j is the vector with all zero entries except the jth entry, which is 1). This means that v_j is an eigenvector of M with eigenvalue σ^2.)

b) Compute the matrix $M = A^TA$ and find the eigenvalues and eigenvectors of M. Make sure the eigenvectors you choose are orthonormal! (Indeed, the matrix $M = A^TA$ is obviously a symmetric matrix, hence its eigenvectors are a basis, and they can be chosen orthonormal.)

c) Order the eigenvalues λ_1, λ_2 you found in (a) in decreasing order, from largest to smallest. Then, let $\sigma_j = \sqrt{\lambda_j}$ for $j = 1, 2$. What are σ_1, σ_2? (You found in (a) the eigenvalue decomposition of M, but as we have just seen, the eigenvectors of M are also the columns of V in the SVD of A! We also know the singular values σ_j: they are the (positive) square roots of the eigenvalues of M!)

d) Now we find U: Since $A = U\Sigma V^T$ and V is orthonormal, then $AV = U\Sigma$, and so the columns of U are : $u_j = Av_j/\sigma_j$ for $j = 1, 2$. What is u_1? What prevents you from finding u_2 this way? Hint: A has rank 1, hence one of the singular values is 0.

e) To find u_2, we simply recall that U has to be orthonormal, hence its columns have to be an orthonormal basis of $\mathbb{R}^2$, so if we know the first column u_1, we should be able to find the second! What is u_2?

f) Verify, using Matlab, that your SVD is correct. To do this, construct the matrix A using the command "`A=[1 4; 2 8]`" then compute its SVD using the command "`[U S V]=svd(A);`", where S will be Σ. Explain any discrepancies between your SVD and Matlab's. Hint: for $A = U\Sigma V^T$, if we change the signs of entries in a column of V, can we change something in U or Σ so that the end result, $U\Sigma V$ still equals A? Recall that entries in Σ should be non-negative.

g) Write down the reduced SVD of A, $A = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, such that $\tilde{U}, \tilde{\Sigma}, \tilde{V}$ are as "small" as possible. Explain why the entries of U, Σ and V you got rid of were "useless".

h) Fortunately for you, you are more intelligent than Matlab! In particular, Matlab cannot recognize that A has rank 1 and hence has a reduced SVD, so you can't compare your result from (g). However, do enter your matrices $\tilde{U}, \tilde{\Sigma}, \tilde{V}$ in Matlab, multiply them together, and verify that you obtain the matrix A.

Question 2. (40 pts.) In this example, we will see that the matrix associated with polynomial interpolation on equispaced points has very bad condition number. Polynomial interpolation consists in constructing a degree $n-1$ polynomial $p(x)$ that *exactly* matches given data (x_j, y_j) for $j = 1, \ldots, n$. That is, we want

$$p(x_j) = y_j \qquad \text{for } j = 1, \ldots, n.$$

There is a unique such polynomial! Let

$$p(x) = \sum_{j=0}^{n-1} c_j x^j$$

be that polynomial of degree $n-1$. As in least squares, we want to find the coefficients c_j of this polynomial, but this time we want no error! So in

$$Ac = y,$$

c is the vector of coefficients c_j, y is the vector of target values of p at the given points, that is, y contains the y_j's, and we want to solve $Ac = y$ exactly. (You can really think of this as doing an "exact" least squares problem, which, well, wouldn't be called "least squares" then.) Finally, we will evaluate this polynomial at m other points, because we would like to know how the data would behave at those points.

a) Write down the entries in A (write down a few entries so I can see you found the pattern).

   Assuming A has full rank, we can now invert it to get the desired coefficients of p: $c = A^{-1}y$. Now that we have those coefficients c, we would like to evaluate p at some other points, say t_k for $k = 1, \ldots, m$, to obtain the value of p there, that is, we want to know $f_k = p(t_k)$. So in $Bc = f$, c is the same as previously, and f is the vector of desired values of p, the f_k's.

b) Write down what entries of B are (again, write down a few entries so I can see you found the pattern).

   Now let's do this in one step: instead of using the vector c as an intermediary, let's just find f from y: $f = Bc = BA^{-1}y$. So we multiply y by $M = BA^{-1}$ to get the answer f: f=My. Notice that M depends on the x_j's and t_k's! Let's build matrices A and B in Matlab now! Let $h = 1/(n-1)$, let x be the following vector of n equispaced points between 0 and 1: $x = 0, h, 2h, \ldots, n$. In Matlab, this is "`h=1/(n-1);x=(0:h:1);`". Then, in Matlab, build A column by column, that is, type "`A=zeros(n,n);for i=1:n,A(:,i)=x.^(i-1);end`". This should help you verify your answer to (a).

c) Let $n = 4$, and construct in Matlab the appropriate A as just explained. Also, let $t_k = h/2, 3h/2, \ldots, 1 - h/2$, so $m = n - 1$ (or "`t=h/2:h:1-h/2`" in Matlab), and construct B in a similar way as you did with A. Show your A and B, and your code for B. (Your code should work for generic n!)

   You can type "`format short e`" in Matlab to see less significant digits if you want. Now let's construct M!

d) Let $M = BA^{-1}$, or "`M=B*inv(A)`" in Matlab. What are the condition numbers of A, B, M? (Type "`cond(A)`", etc in Matlab.)

e) Let now $n = 10$, and construct A, B and M again. What are their condition numbers? (Don't show me the matrices, just their condition numbers.)

   You might notice that A and B are badly conditioned (high condition number) but M is not so bad: this is because Matlab is (sometimes) quite clever and knows that A and B are bad, so it slightly modified them to make them "nicer" before inverting A and multiplying it with B. So it didn't use the true A and B...

f) To convince ourselves Matlab "cheated", find the norm of the error between MA and B (those should be equal since we have $M = BA^{-1}$, so the error *should* be 0, or about 10^{-15} in Matlab... but it's not): "`norm(M*A-B)`". What is the norm of the error?

g) OK, so Matlab knows about bad conditioning! Try constructing A, B and M for $n = 20$ now. Report the warning message you get! (If you don't get an error, please tell me ASAP, it could be that different operating systems work slightly differently.)

h) And finally, to convince ourselves that even Matlab can't do anything against *awful* conditioning, report the condition numbers of A and B (they should be really high now, which is why you got a warning), and report again the norm of the error Matlab makes in computing M (yes, it computed some M even though it gave you a warning).

I hope you are now convinced of two things: first, worrying about the conditioning of your matrix can save you from making inaccurate calculations (i.e., lots of error). Second, interpolating data at equispaced points is a terrible idea. If you ever have to do this, try cubic splines; or use Chebyshev points (we will not discuss those techniques though, they are beyond the scope of this class); or do not *interpolate*, but *approximate* such as least squares does.

Question 3. (20 pts.) In this question, we will use eigenvalues to solve a system of masses connected by springs! Read on if you want to know the setup. Otherwise, start at equation (0.1). Again, just follow instructions!

Imagine you have 2 masses, both of mass m. The first is connected to a fixed wall by a spring with stiffness constant k_1 and the second mass is connected to another fixed wall by a spring with the same stiffness constant, k_1. Both masses are connected together by a third spring, of stiffness constant k_2. (If you don't know what any of this means, you can look up Hooke's law, but this is *not* necessary for the class, or the pset.) The masses and springs are constrained to lie along a line. Then, with x_1 and x_2 being the position along this line of the first and second masses respectively, and $x = (x_1, x_2)$, the equations of motion are:

$$\ddot{x} = \begin{pmatrix} -\frac{k_1+k_2}{m} & \frac{k_2}{m} \\ \frac{k_2}{m} & -\frac{k_1+k_2}{m} \end{pmatrix} x = \begin{pmatrix} -\beta & \alpha \\ \alpha & -\beta \end{pmatrix} x, \qquad \alpha = \frac{k_2}{m}, \qquad \beta = \frac{k_1+k_2}{m}.$$

Now, because we expect the solution x to start to oscillate if the springs are stretched and then released, we assume $x = ve^{i\omega t}$ where $v = (v_1, v_2)$ and the oscillation frequency ω do not depend on t. Now we have to find v and ω! We calculate that $\ddot{x} = -\omega^2 ve^{i\omega t} = \lambda ve^{i\omega t}$, with $\lambda = -\omega^2$. And so we have an eigenvalue problem:

$$\ddot{x} = \lambda ve^{i\omega t} = \begin{pmatrix} -\beta & \alpha \\ \alpha & -\beta \end{pmatrix} ve^{i\omega t},$$

or

$$\begin{pmatrix} -\beta & \alpha \\ \alpha & -\beta \end{pmatrix} v = \lambda v. \tag{0.1}$$

a) Solve equation (0.1). That is, find the eigenvalues λ (call them λ_1, λ_2) and eigenvectors v (call them v_1, v_2) of the matrix $\begin{pmatrix} -\beta & \alpha \\ \alpha & -\beta \end{pmatrix}$. Do not plug in values for α and β, work in general!

   You should have found 2 distinct, real eigenvalues, and two independent eigenvectors (we expect this because the matrix is symmetric - in fact you could make the vectors orthonormal, but don't bother.) Now let $k_1 = k_2 = m = 1$. Notice how the matrix in (a) now looks like a second order difference matrix! That's not a coincidence, but we won't go there.

b) Use your answer in (a) to find the frequencies of oscillation of the system, that is, say what are $\omega_1 > 0$ and $\omega_2 > 0$, where $-\omega_j^2 = \lambda_j$ for $k_1 = k_2 = m = 1$.

c) Use your answer in (a) to find the frequencies of oscillation of the system, that is, say what are $\omega_1 > 0$ and $\omega_2 > 0$, where $-\omega_j^2 = \lambda_j$ for $k_1 = .1$, $k_2 = 1000$, $m = 1$.

d) Would you say the matrix in (b) is badly conditioned? What about the matrix in (c)? Give their condition numbers, using your answers from (b) and (c). What does that imply for the ratio ω_1/ω_2 of the frequencies of oscillations for each case?

Bottom line is: when a system has two similar frequencies of oscillation as in (b), we can compute the solution to that system accurately. However, when a system has two very different frequencies of oscillation as in (c), then we start to make errors. Basically, it becomes very difficult for the computer to tell how fast is the very fast oscillation, and how slow is the very slow oscillation - it simply can't deal with the different scales, fast and slow.