

PROJECT 5

Math 441

Due: Thursday, October 23

All “by hand” calculations require that you show your work. All MATLAB code and output must be turned in.

- Matrix condition number and ill-conditioning:** Consider the system $A(x + \delta x) = b + \delta b$. In class we showed that $\frac{\|\delta b\|}{\|b\|} \leq \kappa(A) \frac{\delta x}{\|x\|}$. Use a similar argument to show that $\frac{\|\delta x\|}{\|x\|} \leq \kappa(A) \frac{\delta b}{\|b\|}$.
- Ill-conditioning of Polynomial Root Finding:** There are no formulas (like the quadratic formula) to compute the roots of a polynomial of order 5 or larger. Thus, in general, a program like MATLAB approximates the roots of a polynomial by some algorithm. In this problem, you will investigate this phenomenon. Download the file **ZoomInPoly.m** from the MATH441 website which plots the polynomial $f(x) = (x - 1)^6$.
 - Run this function in MATLAB by entering **ZoomInPoly(delta)**, where the variable **delta** is set to .01. Run again with **delta= .005**. Turn in the plots with your assignment.
 - Change **ZoomInPoly** so that now MATLAB calculates $f(x)$ using the expansion $f(x) = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$.
 - Run this new function in MATLAB, where the variable **delta** is set to .01. Run again with **delta= .005**. Turn in the plots with your assignment.
 - How are the the plots in #2c different from the ones in #2a? Why are these plots different?
- Singular Value Decomposition:** To answer the following questions, run **TSVDOneD** from the MATH441 website. When asked, enter 0 for the noise level and 0 for the truncation level. Now answer the following questions. Recall that **TSVDOneD** solves the system $Ax = b$ where A is a matrix modeling blur, x is the true 1-D image we want to find, and b is the blurred image. We know that any $m \times n$ matrix (we’ll assume that $m \geq n$) has a singular value decomposition

$$A = U\Sigma V^T$$

where U is an $n \times n$ orthogonal matrix, V is an $m \times m$ orthogonal matrix, and Σ is an $m \times n$ diagonal matrix with diagonal entries $\sigma_1 \geq \sigma_2 \geq \dots \sigma_n \geq 0$.

- In MATLAB, type **[U Sigma V]=svd(A)**, then plot the singular values: **plot(diag(Sigma))**. Use the relative error to check that **U*Sigma*V'** is equal to to A : **norm(A-U*Sigma*V')/norm(A)**. Notice that the singular values are given from largest to lowest.
- Note that a system $Ax = b$ can be solved by

$$x = V\Sigma^+U^Tb$$

where Σ^+ is $n \times m$ and has $\frac{1}{\sigma_i}$ (for the non-zero σ_i) on the diagonal, called the pseudo inverse of Σ . This gives the solution since multiplying both sides of $Ax = b$ by $V\Sigma^+U^T$ gives

$$(V\Sigma^+U^T)Ax = V\Sigma^+U^Tb$$

and the left hand side is $(V\Sigma^+U^T)Ax = (V\Sigma^+U^T)U\Sigma V^T x = x$ since U and V are orthogonal matrices. In MATLAB, find the solution to $Ax=b$: **x=V*pinv(Sigma)*U'*b** and plot it.

- When the smallest singular value σ_n is close to zero, then the condition number of A is large since

$$\kappa_2(A) = \frac{\|A\|_2}{\|A^{-1}\|_2} = \frac{\sigma_1}{\sigma_n}.$$

As we saw in class for the imaging problems, this is why $Ax = b$ is ill-conditioned. Find the condition number of A : **cond(A)**

- To solve $Ax = \hat{b} = b + \delta b$ for such ill-conditioned matrices, one can use the truncated singular value decomposition (TSVD), where only the k largest singular values are used to find an approximate solution to $Ax = b$. Now run **TSVDOneD** for some non-zero noise level δb (say .01), and use some non-zero truncation level. How many singular values did you keep at your chosen truncation level? Explain how you got your answer: the code, and/or the variables in MATLAB (type **whos** to see all variables in memory) and/or the plots.
- Let k be the number of singular values from (d). Now, in MATLAB, calculate the TSVD solution “by hand” by typing **x=V(:,1:k)*pinv(Sigma(1:k,1:k))*U(:,1:k)*b** and plot it. Turn in this plot with your solutions.

- (f) Give the Euclidean norm of the residual for your solution. Based on this residual, is the TSVD algorithm applied to this problem backward stable?
- (g) How is the TSVD solution you found in (d) different from the variable `xtsvd` which the program `TSVDOneD.m` computes? Or are they the same? You'll need to look at the code and see how `xtsvd` is computed.

4. **Interpolating with linear combinations of arbitrary functions:** Do problem 3.1.9.

- (a) Set up a Vandermonde matrix for a quadratic $A = [1 \ t \ t^2]$ to find the coefficients of $a_1 + b_1x + c_1x^2$.
- (b) To solve $Ax = b$, use the SVD as you did in #3b.
- (c) and (d) Use a Vandermonde type matrix $A = [1 \ e^t \ e^{-t}]$ to set up a system to find an interpolating function $a_2 + b_2e^t + c_2e^{-t}$.
- (e) Save off your solutions from #4b and (d), and do something in MATLAB like

```
t=0 : .01 : 4;
quad=a1 + b1*t + c1*t.^ 2;
myexp=a2 + b2*exp(t) + c2*exp(-t);
plot(t,quad)
hold on
plot(t,myexp)
```

5. Download `Eigenfaces.m` and the data `faces.zip` from the MATH441 website. In UNIX, outside of MATLAB, type `unzip faces.zip` to get the individual files out. You'll need to download `Eigenfaces.m` in the same directory where you unzip the data. Run `Eigenfaces` to perform a principle components analysis of the face-data. Because images take up a lot of computer memory (in this case each image is 112×112), we will project each of the images onto a few of the eigenvectors of the covariance matrix (as we've done before in different examples), but in this problem we will view what these "face projections" look like.

- (a) When prompted to enter "Percentage of total variance/power = " enter 50. How many eigenvectors were saved? Note that instead of having to store 112×112 numbers for each image ($112^2 = 12544$), MATLAB will now compress (project) each image into only 1 number (called a weight) for each eigenvector saved! That's a HUGE reduction in memory.
- (b) From the variables and/or code for `Eigenfaces.m`, find the r weights for the 20th individual. These numbers correspond to a projection of the data into the r eigenvectors from #5a. In MATLAB, the projections for all individuals will look something like this:

```
[W e]=eig(C); % Get the eigen-decomposition
W=W(:,size(e,1):-1:1); % This flips the eigenvectors around, so the e-vec for the biggest
                        % eigenvalue is now W(:,1)
Weights = W(:,1:r)*D % Get r weights for all 64 individuals
Projection = W(:,1:r)*Weights % Scale these weights along each of the r saved eigenvectors
```

Note that D is a $112^2 \times 64$ data matrix, where $D(:,i)$ is an image for the i^{th} individual, $Weights$ is $r \times 64$, and $Projection$ is $112^2 \times 64$. Notice, we need to work only with $Weights$ for all the individuals. But we don't need the big matrix $Projection$ unless we want to look at the compressed images.

- (c) In step 4 of the program, the program lets you view the "eigenfaces" which are the eigenvectors of the covariance matrix. Why don't these eigenfaces look like real faces?
- (d) In step 8 of the program, the recognition step, check out figures 7, 8 and 9. What is being shown in figure 7? How is the match in figure 9 being found?
- (e) Is the program recognizing images well? Explain.
- (f) Rerun `Eigenfaces.m`. This time, enter "Percentage of total variance/power = " 100. What do you expect the projections of the training data onto the eigenfaces to look like? Will the relative errors be larger or smaller? Why?
- (g) With "Percentage of total variance/power = " 100, do you expect the recognitions in the testing images to be better? Why? Are they better?