Numerical Analysis 1 – Class 4

Thursday, February 11th, 2021

Subjects covered

- The SVD and dimensionality reduction
- Applications of the SVD: Image compression, latent semantic indexing
- Solving a system of linear equations: Gaussian elimination.
- Matrix decompositions and linear systems: LU, Cholesky, QR.

Reading

- N. Kutz, Chapter 2.1 (Direct solvers for linear systems).
- C. Moler, Chapter 2 (linked on Canvas).
- "A Singularly Valuable Decomposition -- The SVD of a Matrix", D. Kalman (available on Canvas).
- “Using Linear Algebra for Intelligent Information Retrieval”, M. Berry, et al. (linked on Canvas).

Problems

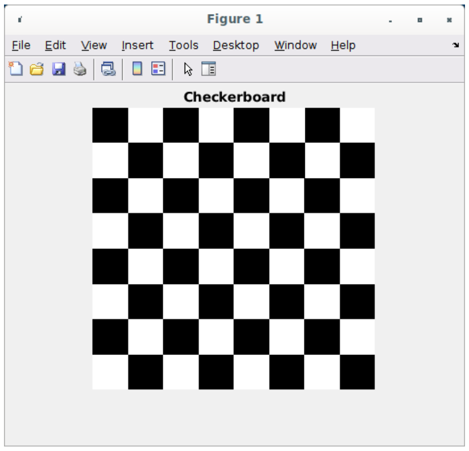
Most of the following problems require you to write a program. For each program you write, please make sure you also write a test which validates your program. Please use Canvas to upload your submissions under the “Assignments” link for this problem set.

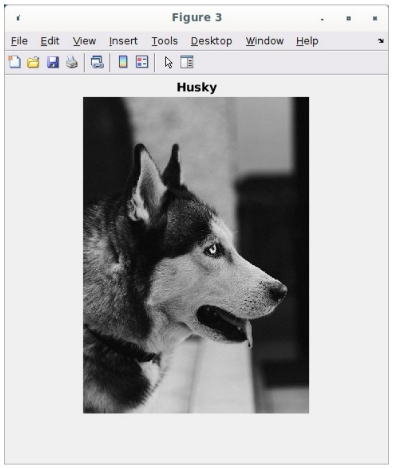
Problem 1

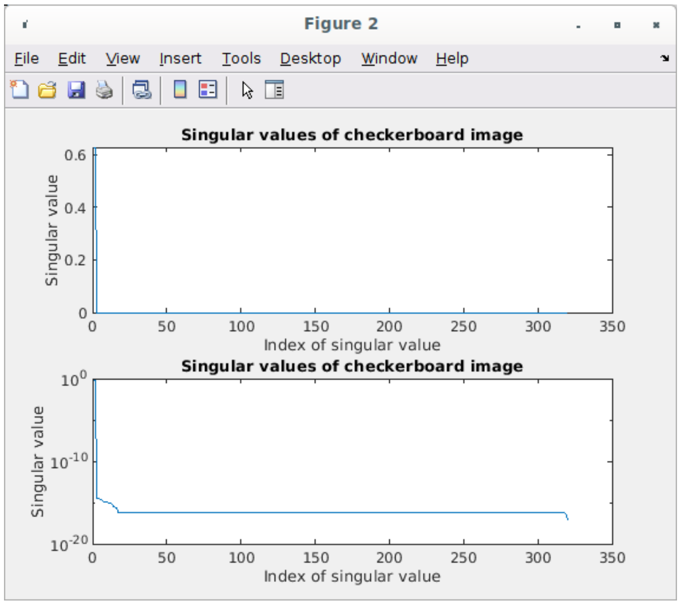
I have placed two black & white images onto Canvas. I saved the images as text (csv) files so you can see the numbers corresponding to each pixel. The files are called “checkerboard.csv” and “side_husky.csv”. Please write a program which does the following:

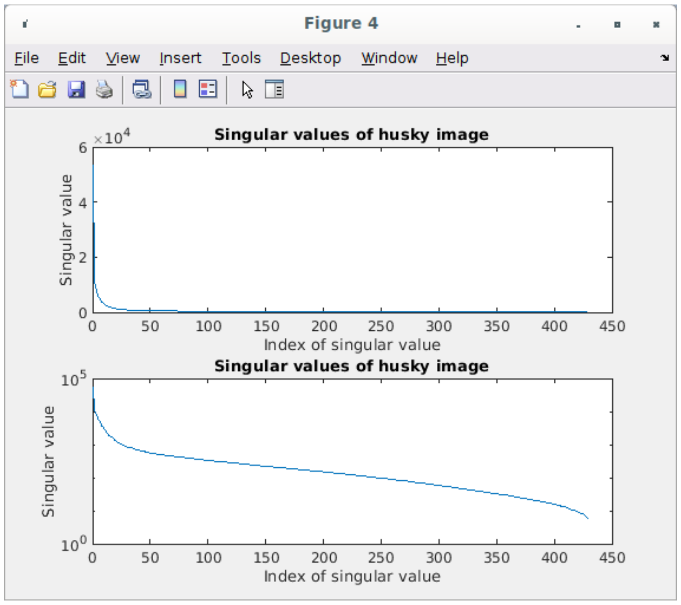
- Read in each image as a matrix. The Matlab function “readmatrix()” will do this.
- Show the image. The Matlab function “imshow()” will do this. My results are shown below.
- Compute the SVD of the image matrix. Then plot the singular values vs. index. My results are shown below. Note that I plot the singular values twice – once on a linear scale then again on a log scale. I do this so you can see all the singular values.
- Now answer the questions: 1. How many non-zero singular values does the checkerboard matrix have? (Note that many of the values are very, very small. They are round-off error so you can consider these values to be zero.) 2. What is the rank of the checkerboard matrix? 3. Why does the checkerboard matrix have this rank? 4. How many non-zero singular values does the husky matrix have? 5. What is the rank of the husky image? 6. Why does the husky matrix have this rank?

Regarding testing, the requested function is simple enough that you don’t need a test. Just supply a runner which will generate the plots.









Problem 2

In class we discussed Gaussian elimination as the basic method used to solve the dense linear system $A x = b$. There are two steps to the Gaussian elimination algorithm: 1. forward elimination and 2. backsubstitution. The goal of this problem is to write a program which implements backsubstitution given an upper triangular matrix *U* and column vector *b*, i.e. the function solves $U x = b$.

I have placed an "app note" on Canvas under Modules → Class 4 describing the exact algorithm useful to solve $U x = b$. Please refer to that note when doing this problem. Please do the following:

- Write a program using for loops to solve $U x = b$ for *x* via backsubstitution. Your program should take as inputs matrix *U* and vector *b*, and return the vector *x*.
- Write tests for your program. As a test I suggest you create a bunch of random matrices *A* of different sizes, then use Matlab's lu() function to generate an upper triangular *U* from *A* to feed your program. (You could also just write a looping program to generate the *U* if desired.) Compute the solution *x* using your program, then compute the residual $b - U x$ and verify it is small.

Depending upon how you generate your test matrix, you may find that the residual grows rapidly with matrix size. Therefore, as testing tolerance I suggest you employ the matrix condition number to generate a tolerance for each new matrix by `tol=eps(1)*cond(U)`.

Problem 3

In this problem we build an algorithm to compute the inverse of a lower triangular matrix *L*. We'll use the tautology $L^{-1}L=I$, where *I* is the identity matrix. We know the elements of both *L* and *I;* the goal is to compute the elements of L^{-1} . But first, what is the shape of the inverse matrix L^{-1} ? It turns out that if L is lower triangular then L^{-1} is also lower triangular. This is a provable theorem, but for this problem you can just accept it as a fact.

Please do the following:

- Knowing $L^{-1}L=I$, derive the equations which each element of L^{-1} must obey. This is a pencil and paper exercise. I suggest you use write down the relation in terms of 3x3 matrices and create a table of equations for the elements of L^{-1} . To help you I show a portion of my table. (In my table I have named the elements of L^{-1} $q_{j,i}$.)

j	**i**	**Equation derived from** $L^{-1}L=I$	**Solve for new unknown q**
3	3	$q_{33}l_{33}=1$	$q_{33}=1/l_{33}$
3	2	$q_{32}l_{22}+q_{33}l_{32}=0$	$q_{32}=-q_{33}l_{32}/l_{22}$
3	1	$q_{31}l_{11}+q_{32}l_{21}+q_{33}l_{31}=0$	$q_{31}=(-q_{33}l_{31}-q_{32}l_{21})/l_{11}$

Then from the table please deduce the general pattern for your algorithm.

- Write a program which implements your algorithm and test it using lower triangular matrices of different sizes. To test I suggest you compute the residual $r=\|L^{-1}L-I\|$ and verify it remains below a tolerance. Note that $\| * \|$ denotes the matrix norm which we learned about in class 3. By default, Matlab computes the "induced" matrix norm, i.e. the largest singular value. Depending upon how you generate your test matrix, you may find that the residual grows rapidly with matrix size. Therefore, as testing tolerance I suggest you employ the matrix condition number to generate a tolerance for each new matrix by `tol=eps(1)*cond(L)`.
- What is the time complexity of your algorithm?