CMSC 25400 Assignment 5

Rachel Hwang

March 8, 2014

1. My eigenvector decomposition program may be found at "hw5/eigP.c", drawing from a function module called "ml_utils.c". In "ml_utils.c", I have implemented the power algorithm, as well as the Jacobi method.

   The Power method I implemented precisely as specified in the assignment. It worked wonderfully for the calcuation of the first eigenvector. However, I noticed that even with careful and frequent re-orthogonalization, this method was extremely, prohibitively slow (as it depends on randomness to converge properly). For extra credit, I implemented the Jacobi method, which worked much better. Below are some sample inputs/outputs of "eigP.c". All input matrices are completely randomly generated with values from 0.0 to 0.9. My program generates a random symmetric matrix, writes the matrix to "test.mat", then after computing the eigenvals and eigenvectors, writes them to "eVals.data" and "eVecs.data" respectively.

   For a 3x3 matrix,

   Input Matrix 1

0.0	0.0	0.6
0.0	0.4	0.0
0.6	0.0	0.6

   Eigenvalues 1

0.970738	0.400000	-0.370738

   Eigenvectors 1

0.532390	0.0	0.846499
0.0	1.0	0.0
-0.846499	0.0	0.532390

   For a 5x5 matrix,

   Input Matrix 2

0.100000	0.300000	0.500000	0.700000	0.300000
0.300000	0.200000	0.600000	0.600000	0.100000
0.500000	0.600000	0.400000	0.500000	0.000000
0.700000	0.600000	0.500000	0.600000	0.200000
0.300000	0.100000	0.000000	0.200000	0.900000

   Eigenvalues 2

2.043884	-0.200963	-0.527093	0.010145	0.874026

   Eigenvectors 2

0.433119	0.426287	0.468098	0.588312	0.255853
-0.561518	0.694391	-0.397725	0.203964	0.052275
-0.659654	-0.443810	0.398474	0.445703	0.102252
0.246501	-0.330641	-0.623686	0.636554	-0.189022
0.034624	-0.172658	-0.273739	-0.091847	0.941075

   Below is the plot of the number of iterations my program took to converge at each epsilon and n value, over matrices with values from 1 to 100. As is intutitive, smaller (stricter) epsilon values require far more iterations; there is a negative correlation between epsilon and interations

required. The far more surprising relationship here is that between n value and the number of iterations required. The larger the matrix, the fewer the iterations required. This the related to margin of difference in value between the largest/first eigenvector and the next eigenvector. I compared my iterations graph with several classmates and found that this relatonship is determined by the distribution of values in the matrix, or how the random matrix is generated. Using a (1000 x 1000) matrix generated by a classmate (James Porter), "hw5/rachel.data", with epsilon = 10^{-6}, my program instead took 607 iterations to converge.

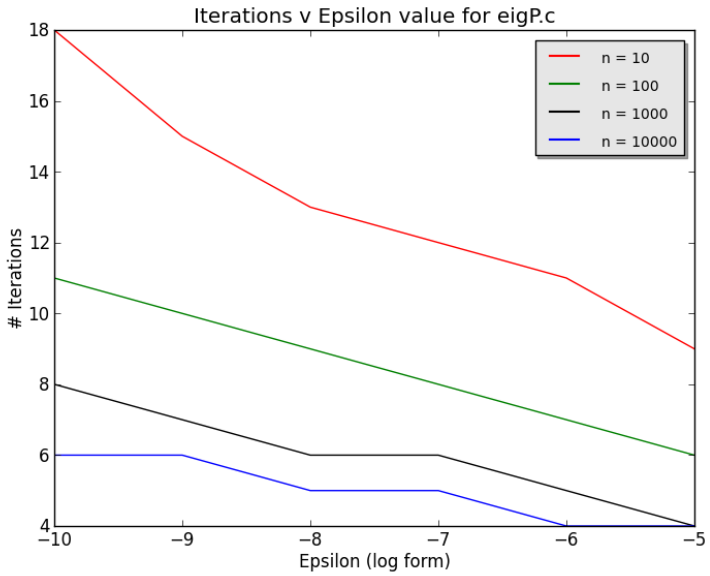

Figure 1:

2. Dear Grader, I cannot make my program work. I'm very sorry. I've been working on this problem set for in excess of 36 hours near continously, and due to my other pressing work concerns, I have to stop. I could not make my program generate the correct output, but please, please consider that Idid in fact implement the power method, the jacobi method, a sort for eigen vectors, and 99 percent of the svd algorithm. There is a small bug somewhere in my code, and I simply cannot find it, having walked through my code dozens of times, likely breaking othering things in the process. Having discussed this assignment extensively, with classmates and looked at $successful$ data, I understand the interaction of k and Frobenius error. Had I been able to collect my data successfully, I would see a pretty much strictly linear, negative correlation between k value and error rate. This is intuitively correct because the higher the k value, the more information we have with which to generate S, creating a more accurate diagonal matrix. This is turn generates a more accurate $A' = USV^T$, lowering the frobenius error rate. The forumla I used to generate S, in "ml_utils.c", is ($S = U^T AV, U[i] = normalized(Av_i), v_i$ = the ith eigenvector of $A^T A$). Once, again, I'm sorry I do not have actual data. Please read my code to see that I did do much of the work for the algorithms. On the bright side, I now have a pretty extensive library of linear algebra functions for my own use.

Best,
Rachel