Kevin Smith
Math 2506 G1
Jacobi Algorithm Project

What I did / Implementation / Graphing

My implementation consists of two main classes, Jacobi and JacobiApp. JacobiApp gives a graphical interface to choose between running the algorithm by sorting through the off diagonal elements for the largest value, or systematically running through the off diagonal elements until a diagonalized matrix is achieved. Once a choice is made, it creates 10 new Jacobi classes and runs the algorithm 10 times, each time adding the points to be graphed to a list. Once that is done, the points are then drawn onto the graph.

The graph provides a good idea for how well the algorithm is running. The x axis represents the kth iteration of the algorithm, while the y axis represents the natural log of the summation of all off diagonal elements squared. The black points on the graph represent these values produced by running the algorithm. The green points on the graph represent the values created by the line:

$$y = x\ln(9/10) + \ln(off(A))$$

Because we have the theoretical bound:

$$bk <= k\ln(9/10) + \ln(off(A))$$

we know that all the black points should fall below that line. At the top we show the original matrix for one of the algorithm runs, as well as the diagonalized matrix for that run on the bottom of the graph.

The Jacobi class is what does the real work, and actually runs the algorithm. When instantiated, a random mXn matrix is created and then made symmetric. Then we can run the algorithm either using the largest off diagonal elements or systematically choosing, and return a list of points to be plotted. When running the algorithm, the JAMA library is used to find the unitary matrix U from the Bij that is then used to find the givens matrix G. JAMA is also used in the creation of the original matrix, and in matrix multiplication to find our new B iteration with $B = G^t B G$

NOTE: I just read that I was supposed to return a list off off(b) from each iteration. I did not do this directly, but rather returned a list of points (k, bk). Because $bk = \ln(off(b))$, off(b) could be obtained from these points rather easily like this: $off(b) = e^{bk}$. I believe that the reasoning for returning this list is to see them get smaller and smaller as the algorithm progresses. This effect can still be seen by looking at the values bk.

Analysis

2] The actual data all runs below the theoretical bound. In fact, it quickly leaves the bound and bk goes much lower much faster then the bound does. This is good, because it means our algorithm is getting the job done and doing it quickly.

3] If we don't bother finding the largest off diagonal entries, but instead systematically “sweep through” the upper triangle entries, we get a bit of a different results. While it does work, it seems to take a bit longer and has to run through more iterations to finish. Not only that, but results tend to vary

a bit more. I've also noticed that points even sometimes go above the theoretical bound, which never happens when choosing the largest values. In other words, choosing the largest values produces faster and more reliable results and is an important part of the process, particularly if the matrix were to grow very large.

NOTE: Again, I just noticed I read the directions wrong. Instead of producing 10 different graphs, I ran the algorithm 10 times and and plotted the results from all ten runs onto the same graph. I also only show the Original matrix and the diagonlized matrix for the last run. But if the program is run through the terminal or through eclipse I will print out each iteration so they can be seen easily there as well as the value for off(B)