Project 1 for Calc III (Math 2605)

Gene Hynson, September 26th 2013

The Jacobi Algorithm

- **What you did and why?**
    - In my Jacobi algorithm program, I used Java to create an applet, perform basic matrix operations, and compute the Jacobi of a 5x5 matrix. When the program is run, a graph is shown along with three text boxes. The first text box shows the values in the original, random 5x5 matrix. The second box shows the values of the matrix after performing the Jacobi algorithm and sorting. The final, third, box shows the values of the matrix without performing the sorting operation. On the right side of the applet, a graph is shown with three unique lines. The red line represents the theoretical bound (i.e. *ln(9/10)*), the green line represents the actual data from the sorted matrix that went through the Jacobi algorithm (i.e. *xln(9/10)+ln(Off(A)) & ln(Off(B)),* and the black line represents the unsorted matrix. To compare multiple graphs, continue clicking the button inside the applet to create a new, random matrix and graph. I also included two counters as labels inside the applet to show the difference in efficiency of sorting versus not sorting a matrix.
    - Inside my code, I used both the Graph package and the JAMA package. The Graph package provides some very useful tools to create a beautiful graph inside the applet. The JAMA package allowed me to create Matrices inside my program instead of using double[][]. I also used the JAMA package to perform simple Matrix operations like finding the Transformation, Adding, Subtracting, Multiplying, etc. The object of the program is to diagonalize a square matrix until the diagonal matrix is filled with the eigenvalues approximated to 10e-9.
        - So, after initializing the graph my program generates a random symmetric matrix.
        - Next, we finds the off-diagonal entries and the largest of those entries.
        - In order to update our diagonalized matrix we must first diagonalize a 2x2 matrix filled with largest “off’s” and find its eigenvectors.

- This can be accomplished by first finding the eigenvalues using the quadratic formula. Eig1 = *((d+a) + sqrt((d-a)^2 + 4*b^2))/2* in the matrix,
  |a b|
  |c d|
- Now that we have the eigenvalues, we can simply use the equation, *A-(lambda*I) = x* to find the eigenvectors.
- Finally we normalize the eigenvectors and place them inside the rotational matrix. Now we use the find the diagonal by using the equation,
  $D = R^T * A * R$.

- Using D, we can check to see if off(D) < 10e-9.
- When drawing the non-sorting graph, the program doesn't sort through the matrix to find largest off diagonal entry.

- **How does the actual data compare with the theoretical bound?**
  - The actual data stays fairly close the theoretical bound for the first couple of steps and then begins to plummet downward until it reaches optimum diagonalization. The theoretical bound slowly slopes downward, eventually crossing over the same "y-values" as the actual data, but much further down the X-axis.
- **What do you conclude about the importance of sorting versus its expense?**
  - Sorting the matrix during the Jacobi algorithm is an important step in the overall efficiently of the algorithm. As shown in the program, sorting the matrix can easily cut off 5 to even 10 steps to arrive at diagonalization! This makes the sorting process easily worth its expense.
- **What happens if you do not sort the matrix?**
  - The results tend to follow the theoretical bound for several more steps than the actual data. This means that more steps will be required to bring the matrix to optimum diagonalization. It also does not produce a smooth, curved graph like the actual data does. This means that when the matrix is not being sorted, it lacks consistency in the completeness of the diagonalization process for every step. Some steps produce a much more diagonalized matrix and others do not. The actual data's graph is represented by a very smooth line meaning that each step is consistently producing a more accurate, diagonalized matrix.

Original Matrix

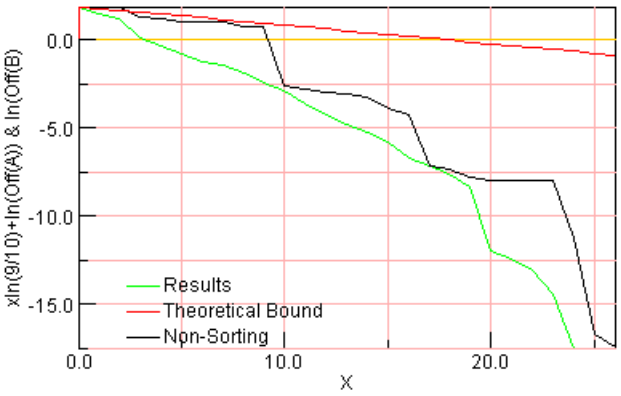
0.222	0.191	0.316	0.047	0.374
0.191	0.443	0.717	0.743	0.287
0.316	0.717	0.048	0.379	0.685
0.047	0.743	0.379	0.049	0.979
0.374	0.287	0.685	0.979	0.161

Diagnolized Matrix

-1.122	0.000	-0.000	0.000	0.00
0.000	0.229	-0.000	-0.000	0.00
-0.000	-0.000	0.059	0.000	-0.00
0.000	0.000	0.000	2.186	0.000
-0.000	0.000	-0.000	0.000	-0.42

Non-Sorted Matrix

2.186	-0.000	0.000	0.000	-0.00
-0.000	0.229	-0.000	-0.000	0.00
0.000	-0.000	0.059	0.000	0.000
0.000	-0.000	0.000	-0.428	0.00
-0.000	0.000	0.000	0.000	-1.12



Steps to diagnolization when sorting: 25

Steps to diagonlization when not sorting: 27

Run Again With New Matrix