Report on the implementation of the Jacobi Algorithm

My implementation of the Jacobi Algorithm is written in C#, a language I chose because of its high-level syntax and a fast running time. In my implementation, I have split the different steps of the algorithm into separate methods, such as Matrix multiplication, transposition, and Eigenvalue calculation. This algorithm was simple to implement because it requires only 2x2 matrices in its intermediate steps Instead of finding the characteristic polynomials and performing Elementary Row operations to obtain the Reduced Row Echelon Form, I wrote formulas that compute the Eigenvalues and Eigenvectors based on and off-diagonal elements of the matrix. This made the computation simpler and gave a faster calculation than would have been with traditional methods.

My Jacobi method runs in a loop, which is guaranteed to run at least once. After each iteration, it computes Off(), which is the sum of the squares of the off-diagonal elements. The loop only stops when the computed Off() after each loop is less than 10^-9. Apart from the actual Jacobi algorithm, I have another class that handles program control. In this Program class, a loop runs infinitely running the Jacobi algorithm with random matrices, and afterward prompting the user if he or she wants to compute the Eigenvalues of another matrix. The entire output of the program is shown in an Excel File, which include Graphs, original and resulting matrices, and computational data. The Excel file is a program in itself, because the columns that are graphed obtain their data based on the values of the 2nd column. To get those values, I have defined formulas for the cells, so that the columns stay updated with newer data. The excel report is itself divided into two worksheets, for data obtained from the sorting method, and that obtained from the symmetric method. Both worksheets are updated after each iteration of the program.

On average, my implementation of the Jacobi Algorithm completes within 25 iteration for the sorting method, and 60 without. The two approaches that I use differ by how sub-matrices are chosen. The sorting method looks for the indexes with the maximum absolute value in the Upper Triangular region of the matrix, while the unsorting method sequentially uses the next pair of indexes that are off the diagonal. My graphs contain a table of data, which consists of the value of Off() at the k’th iteration, the natural log of Off() at the k’th iteration, and the solution to the line

http://www.texify.com/img/%5CHuge%5C%21b%28k%29%20%3D%20k%20ln%289/10%29%20%2B%20ln%28Off%28A%29%29.gif

at the k’th iteration, where Off(A) is the value of the original matrix computed before any iterations of the Jacobi algorithm are executed.

Overall, my data has been very consistent. The linear fit of in(Off(A)) is always at a downward angle of bk, with all tests. A major difference between my two approaches is that Off() descends more smoothly with sorting, but tends to decreases only over short intervals without sorting. A reason for this may be that without sorting, elements that are already reduced to their shortest possible value are repeatedly simplified, even when further simplification of those elements is unnecessary. The sorting approach deals with this problem my ignoring shorter values in favor of larger values, reducing them before moving onto miniscule values. Another trivial, yet interesting, difference between the two methods is that the last two Eigenvalues computed are always flipped, for every case the data is tested for.

On average, the Jacobi algorithm completes within 25 iterations for the sorting method, and 60 for the unsorted method. The sorting method finds the element with the maximum absolute value, and uses its indices to locate the 2x2 matrix. The unsorted method uses the next off-diagonal element to find those indices, starting from the beginning after the last element is reached. Both curves look very similar in slope and shape, although the sorted method is less chunky. I believe that this is because it continiously reduces the elements that are not already reduced, while the unsorted method may try to reduce an element which is already below the threshold value. Since both matrices produce the same eigenvalues after the Jacobi Algorithm returns, I believe the data and algorithm work correctly.

A surprising result that I had not anticipated was that every matrix that I had tested converged in very few tests. Since the Jacobi algorithm is not guaranteed to always converge, I had thought that there might be cases when the algorithm would continue to run infinitely. However, no data ever exceeded 70 iterations. This project helped me understand that every symmetric matrix must have Eigenvalues that can be calculated with few calculations.