Conclusions from the Jacobi Method Project

Project by Tiff Zhang

For Math 2605 at Georgia Tech

Introduction

The purpose of this assignment was to help me better understand the process behind the Jacobi Algorithm by implementing the algorithm in a web application. In the process of debugging my program, I corrected a few of my misunderstandings about the Jacobi Algorithm, and in the process of completeing the comparison required by the assignment, I came to understand the importance of the sorting step in the algorithm.

A Brief Explanation of Jacobi's Algorithm

The purpose of Jacobi's Algorithm is to the find the eigenvalues of any mxm symmetric matrix. In general, two by two symmetric matrices will always have real eigenvaleus and those eigenvalues can be found by using the quadratic equation. Larger symmetric matrices don't have any sort of explicit equation to find their eigenvalues, so instead Jacobi's algorithm was devised as a set of iterative steps to find the eigenvalues of any symmetric matrix.

Jacobi's Algorithm takes advantage of the fact that 2x2 symmetric matrices are easily diagonalizable by taking 2x2 submatrices from the parent, finding an orthogonal rotation matrix that diagonalizes them and expanding that rotation matrix into the size of the parent matrix to partially diagonalize the parent. More specifically, the basic steps for Jacobi's Algorithm would be laid out like such:

1. Start with a mxm symmetric matrix A
2. Find the off-diagonal item in A with the largest magnitude
3. Create a 2x2 submatrix B based on the indices of the largest off-diagonal value
4. Find an orthogonal matrix U that diagonalizes B
5. Create a rotation matrix G by expanding U onto an identity matrix of mxm
6. Multiple G_transpose * A * G to get a partially diagonlized version of A
7. Repeat all steps on result from Step 7 until all of the off-diagonal entries are approximately 0

So, as long as you know Jacobi's Algorithm you candiagonalize any symmetric matrix! But, especially for large matrices, Jacobi's Algorithm can take a very long time with a lot of iterations, so it's something that we program computers to do. And that's why I made this program here: to have a computer do the heavy lifting of iterating through matrices.

Convergence and "Sorting"

A problem with the Jacobi's Algorithm is that it can get stuck in an infinite loop if you try to get all of the off-diagonal entries to exactly zero. So, when we do the Jacobi's Algorithm, we have to set a margin of error, a stopping point for when the matrix is close enough to being diagonal. For this project, the stopping rule we used was sum(offB^2) < 10e-9. Thus, when the program reached a point where the square of all the off diagonal entries added up is less than 10e-9, it would stop.
Other than picking an error though, we can change specific details in our implementation of Jacobi's Algorithm. Step 2 from my earlier list, where you find the largest off-diagonal entry of the matrix, is not strictly necessary because you can still diagonalize all of the parts of a matrix if you just iterate through the off-diagonal values.

That's what my simulation in the "Math 2605 Simulation" tab was all about. Starting with one set of the same 10 symmetric matrices, I ran two different variants of the Jacobi Algorithm: the first using the sorting step to find the largest off-diagonal value and the second just iterating through the values.

When I graphed the results, I found that for 5x5 matrices, Jacobi's Algorithm with the sorting step tended to converge in between 20-30 iterations while the algorithm without the sorting step tended to converge in about 30-40 iterations. When I ran similar tests on matrices of larger sizes, I found that Jacobi's Algorithm without the sorting step generally tended to take approximately 30% more iterations.

Conclusions

It's clear overall that the sorting step in Jacobi's Algorithm causes the matrix to converge on a diagonal in less iterations. But the reason we looked at the sorting step was that it can be slow for large matrices; after all, you have to go through all of the off-diagonal entries and find which one is largest. However, the iterations of the Jacobi Algorithm saved by the sorting step take time to process also. Since the sorting step significantly reduces the number of iterations of Jacobi's Algorithm needed to achieve a diagonal, it's clear that it's pretty useful. And it makes sense; by systematically applying Jacobi's algorithm to the off-diagonal elements furthest from zero, you're going to get all of the off-diagonal elements to approach zero the fastest.

So, in conclusion, this project shows that Jacobi's Algorithm is a rather handy way for a computer to figure out the diagonals of any symmetric matrices. With the diagonal of a matrix, we can find its eigenvalues, and from there, we can do many more calculations.