Eigenvalue Problem with Symmetric Matrices, Program 4

Colton Piper

March 9th, 2018

1 Executive Summary

We are implementing the implicit QR algorithm and analyzing it. This algorithm is used to approximate eigenvalues of a symmetric matrix. We will also be looking at an algorithm that transforms a symmetric matrix into a symmetric tri-diagonal matrix using householder reflectors to preserve the eigenvalues.

2 Statement of the Problem

The problem is implementing the implicit QR algorithm that finds the eigenvalues for a symmetric matrix that is both efficient and accurate. Thus we do not want not be wasteful in time complexity and do not want to store any redundant information. We will be implementing first an algorithm that transforms the symmetric matrix into a tri-diagonal matrix. This will not be wasteful in storage thus we will only pass in a step matrix that has the diagonal and below the diagonal elements of the symmetric matrix and then returns a 2D vector which will have the diagonal as the first vector and the sub/super-diagonal as the second vector. Then we will have an algorithm that uses the QR algorithm that finds the eigenvalues for a tri-diagonal matrix.

3 Description of the Mathematics

Now to transform a symmetric matrix into a tri-diagonal matrix we must use orthogonal transformations. This is because we want the transformed matrix to have the same eigenvalues as the original matrix. There is a theorem that tells us that tells us that if we have a matrix $A = S^{-1}TS$, then the matrices A and T are similar and have the same eigenvalues. We will be doing $T = HAH^T$ where T is a tri-diagonal matrix and H are householder reflectors. Now because householder reflectors are orthogonal matrices their transpose are their inverses. Thus T and A are similar matrices and have the same eigenvalues.

Now that we have our tri-diagonal matrix we will use Given's rotations to implement the implicit QR algorithm. This iterative scheme transfers the magnitudes of the sub/super-diagonals into the main diagonal. Thus the more iterations you use chasing the bulge using Given's rotations the closer to zero the sub/super-diagonals will be. We will be using Given's rotations such that $\tilde{\Lambda} = G^T T G$. One iteration of this is we will first create an element that is right off the sub/super-diagonal and then use Given's rotations to push this element down the sub-sub-diagonal until it is out of the matrix entirely.

4 Description of the Algorithms and Implementation

The first algorithm is using the implicit QR algorithm to approximate the eigenvalues of a symmetric tri-diagonal matrix. The way I implemented this program was explicitly wrote a general 4x4 symmetric tri-diagonal matrix with arbitrary elements, then saw how applying one Given's rotation to both the left and right of T affects the elements of T. A 4x4 matrix was the best case as you have a chance to see which elements transform due to the first Given's rotation and then which elements will transform for an i^{th} Given's rotation applied to T on both sides and lastly how the Given's rotation that pushes the bulge out of the matrix effects the elements of the symmetric tri-diagonal matrix. This then yielded explicit equations on which elements of the matrix get affected by the i^{th} Given's rotation.

Now that we had explicit equations we could import them into code which sounds easier then when done. The indexing proved to cause a lot of bugs and it took some time to finally get the code running properly. To do these updates without using parallel computing

we needed to use some temporary variables as many elements affect how many others will update and those affect how that element will update. All said and done we only get 5 storage. The eigenvalues will already be stored in the 2D-vector, symmetric tri-diagonal vector, that you passed into the function.

The complexity of the code really depends on how accurate you want the eigenvalues to be. But the structure of the code is an outer for loop that iterates n times for the amount of eigenvalues we need to find. Then a while loop that continues to iterate until the element of the sub/super-diagonal in check is below a certain tolerance to zero. Then inside that loop is another for loop that iterates $n-1-k$ times where k is the amount of approximated eigenvalues found.

The next code to implement was transforming a symmetric matrix into a symmetric tri-diagonal matrix using householder reflectors. The way I implemented this code was first found what HAH^T was and then transformed it into a 2-rank update to A. After distributing we get that

$$\begin{aligned} HAH^T &= A + a(x * y^T + z * x^T) \quad \text{where} \\ x &= v \pm ||v||_2 e_1 \\ a &= -2/||x||_2^2 \\ y &= A * x \\ w &= x^T * y \\ z &= y + awx \end{aligned}$$

and v is the vector used to create the householder reflector. We will create $n-2$ householder reflectors thus we will be calculating all these $n-2$ times. So first we will find the norms of v and x and that just takes a for loop then x norm squared we can explicitly achieve from the norm of v because x and v are so close to the same vector. Next we will solve for the vector y and as that is a matrix vector product it will take quadratic time to achieve. That already brings our complexity to on the order of n^3 because we have n^2 time for n times. Then we compute the dot product of for w using a simple loop and lastly a vector sum to create z.

Those are just the pre-processing steps then we can finally apply a 2-rank update to A. That turned out to be more difficult again because of indices and the fact that I only stored the vectors y and z as x we already new because it lied in A. Now remember that all x, y, and z are not $nx1$ vectors as they decrease by 1 row every time we introduce a new householder to make under the sub-diagonal zero. Thus each iteration we are only updating the schurr complement of A.

5 Description of the Experimental Design and Results

I found examples online to check my code with. I ran out of time doing this, thus I just have simple examples to check with. My examples I found from the website, http://mathfaculty.fullerton.edu/mathews/n2003/householdermod.html, had these three matrices and transformed them into 3 symmetric tri-diagonal matrices.

$$\begin{pmatrix} 4 & & & & & \\ 1 & 2 & & & & \\ 2 & 0 & 3 & & & \\ -5 & 4 & -1 & 1 & & \\ 1 & 5 & 2 & 5 & -2 & \\ 4 & 3 & 1 & 2 & 4 & 1 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 5 & & & & \\ 1 & 1 & & & \\ 2 & 2 & 0 & & \\ 2 & 1 & 2 & 1 & \\ 4 & 0 & 1 & 2 & 4 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} 4 & & & \\ 2 & -3 & & \\ 2 & 1 & 3 & \\ 1 & 1 & 1 & 2 \end{pmatrix}$$

My code transformed them into the following

$$\begin{pmatrix} 4 & -0.148936 & 1.26851 & 2.66491 & 3.35819 & -2.14267 \\ 6.85565 & 2.92443 & 4.75824 & -7.99442 & -1.75936 & \end{pmatrix} \quad \text{and}$$

$$\begin{pmatrix} 5 & 5.8 & -0.882353 & 1.37321 & -0.290861 \\ 5 & -0.824621 & 1.57787 & 1.27902 & \end{pmatrix} \quad \text{and}$$

$$\begin{pmatrix} 4 & 2 & -1.4 & 1.4 \\ -3 & -3.16228 & -0.2 & \end{pmatrix}$$

where the first row is the diagonal and the second row is the sub/super-diagonal. My numbers agreed with the website up to sign as they had $\pm||v||_2$ changing for x in the householder reflectors. My code can have that, but I did not see how the website was determining which sign it was to hold. In my code right now you can change the condition

to decide when to have it positive or negative, but I have it as a positive addition every time.

Then I used a website called www.symbolab.com to find the eigenvalues of these matrices and see if they agreed with the ones I got in my code. For my code I got the the following eigenvalues with using a tolerance of 0.000001 for the sub-diagonal element.

Size	Value 1	Value 2	Value 3	Value 4	Value 5	Value 6
4x4	6.64575	-3.64575	1.64575	1.35425		
5x5	10.4493	2.65205	-2.04336	-0.577672	0.519702	
6x6	12.3023	9.32815	-7.41126	-5.31112	2.2255	-2.13354

Table 1: This is a table of eigenvalues for the three matrices above using the implicit QR algorithm in double precision with a tolerance of 0.000001.

The last eigenvalues the website symbolab could not handle and thus I reverted to using WolframAlpha to find the eigenvalues. All three did match what the things gave us. One of the interesting features though is that after the last element of the sub-diagonal reached the tolerance of my choosing the rest of the sub-diagonals we already much closer to zero than the it thus we found all the eigenvalues were collected. The numbers are in the table below.

Size	Value 1	Value 2	Value 3	Value 4	Value 5
4x4	3.88828e-16	5.85832e-21	-9.11129e-07		
5x5	1.18677e-87	2.57683e-16	1.153e-80	-9.10075e-07	
6x6	1.66369e-38	1.26454e-31	2.92429e-47	1.03378e-122	9.79675e-07

Table 2: This shows what values of the sub/super-diagonals were after the last sub/super-diagonal element reached the tolerance of 0.000001.

I found that interesting and I am not sure if I might have implemented my code wrong or what. But I that generally the last sub-diagonal reaches the tolerance first than you work with and $n-1$ dimensional square matrix and continue up the chain. Then in the case that one of the middle sub-diagonals reaches below the tolerance first you could bring in deflation and split up the matrix to make the amount of computations less.

6 Conclusions

This was not the best analysis of the implicit QR Algorithm, but I learned how it worked and the code does work, but I am not sure that I implemented it correctly as after we find the first eigenvalues all the other sub-diagonals are already below the given tolerance level. Changing a symmetric matrix to a symmetric tri-diagonal matrix is a process that takes cubic time but only requires linear storage. While the implicit QR algorithm is more of a converge iterative scheme we will not talk about the complexity, but it does have two nested loops with-in. It does however only need constant storage to complete. As it only declares a few variables inside the function.