Project for Math 2605

Implementing the QR method

This project is concerned with the QR algorithm applied to the problem of finding one eigenvector of a matrix. You may do it using Maple, MATLAB, or Java. *You may do this project only if you also do the project on the power method.*

The QR algorithm is the power method in disguise. Does the disguise work better in practice? The aim of this project is to investigate this.

If we run the QR iteration, say, 10 steps and produce $A^{(10)}$, this will upper triangular only if there are large separations between all of the eigenvalues. And the first column of $U^{(1)}$ will be a good approximate eigenvector of A only if the largest eigenvalue (in absolute value) is much larger than the second. In general, we won't be so lucky, so we have to shift – just as in the power method.

The goal here is to use QR iteration to find one eigenvector of a matrix A. Here is how we will do it:

First, run 10 steps of QR and produce $A^{(10)}$, and $U^{(1)}$. If the first column of $A^{(10)}$ is nearly zero below the top entry, we may already have our eigenvector. Let $\mathbf{u}$ be the first column of $U^{(10)}$, and check to see if

$$|A\mathbf{u} - (\mathbf{u} \cdot A\mathbf{u})\mathbf{u}| < \epsilon \tag{1}$$

where ϵ is our tolerance level. For purposes of this project, take $\epsilon = 10^{-8}$. If so, we are done. If not, we shift by the upper left entry of $A^{(10)}$, which will be $c = \mathbf{u} \cdot A\mathbf{u}$. That is, replace A by $(A - cI)^{-1}$, and go back to the beginning with this new A.

If the first column of $A^{(10)}$ is not nearly zero below the first entry, let B be the 2×2 matrix in the upper left. Since B is 2×2, finding these eigenvalues is easy. Compute the eigenvalues of B, and let c be one of these eigenvalues – either the largest one if they are both real, or the one with the positive imaginary part if the are a complex conjugate pair. Replace A by $(A - cI)^{-1}$, and go back to the beginning with this new A. *Extra credit: In your write up, explain why this is a good thing to do.*

As before, once you have made a complex shift, you will be working in C^n, not R^n, and from this point forward, you must replace (1) by

$$|A\mathbf{u} - \langle \mathbf{u}, A\mathbf{u} \rangle \mathbf{u}| < \epsilon \tag{2}$$

and you should now use $\langle \mathbf{u}, A\mathbf{u} \rangle$ as the value to shift by.

Keep running this loop until either (2) or (1) is satisfied, depending on whether you made a complex shift or not.

How well does this work?

To examine this, generate 10 random 6×6 matrices. Say how you generate them – random integer entries from some interval, floats from some distribution, etc. Use any random matrix generator that gives "generic" matrices – we wouldn't want them symmetric for example.

Apply the method described above to try to find one eigenvector for each of these matrices. Show the matrices, and when it works, the eigenvector and corresponding eigenvalue, and answer the following questions:

(1) Does it work for all 10 of your matrices?

(2) How many times did you get a complex eigenvector and eigenvalue?

(3) How much shifting did you have to do? Keep track of what you shifted by in each case. Which example required the most shifts? Which required the fewest?

(4) Try replacing 10 QR iterations with some other number of iterations. How does this affect the efficiency of the method. Can you suggest some optimal policy for choosing the number based on your experience?

(5) Compare results with what you found using the power method *au natural*.

It will probably be most convenient to do the project up to here using Maple or MATLAB. The following optional extra credit part should be done in Java. This will give more than the usual extra credit. If you do both parts below, it counts as an entire second project. However, the second part can only be done one – for either this project, or the one on the power method. (This one is recommended if you want to do it for one).

• Write a small package that has a complex vector object, and a complex $n \times n$ matrix object. Write into the matrix object a class method for returning one (possibly) complex eigenvector.

Write an applet that uses your package to find eigenvectors. For still more extra credit:

• Give your vector object a class method for returning a Householder reflection matrix that reflects the given vector to a multiple of $\mathbf{e}_1$, and give your matrix object a class method for extracting the lower right $(n-1) \times (n-1)$ block. Using these, implement the Schur factorization as a class method of your matrix object: Write in a class method that returns the U and T with $U^*AU = T$. Put this in an applet that generates random matrices, and computes their Schur factorizations, and displays them. Fro the applet, fix $n = 5$, and give a button so every time you press it, you see a new A, T and U.

You may wish to display the real and complex parts of T and U separately.