Project for Math 2605

Implementing the power method

This project is concerned with the power method for finding one eigenvector of a matrix. You may do it using Maple, MATLAB, or Java.

If you take an $n \times n$ matrix A, and form, say, A^{16}, the first column of A will only be an approximate eigenvector of A provided that the largest (in absolute value) eigenvalue of A is a good bit larger than the next largest. If it is about twice as large, then since $2^{16} = 65,536$, the relative participation of the other eigenvectors in the first column of A^{16} will be reduced by a factor of $1/65,536$, washing them effectively away to this degree of accuracy.

If the ratio between the absolute values of the two largest eigenvalues is closer to 1, then a much higher power is required to wash the second eigenvector out. And if the largest eigenvector is one of a complex conjugate pair, so that there is an exact tie, no power will do, and we must shift. And in fact, we must shift by a complex number.

To shift, we choose choose a number c, possibly complex, and replace A by $(A-cI)^{-1}$, and apply the power method to that. If we have chosen c well, we will get extremely rapid convergence.

So here is how we will run the power method: First, compute some power, say A^{16}, of A. Let $\mathbf{u}$ be the unit vector obtained by normalizing the first column of A^{16}. Compute

$$\mathbf{u} \cdot A\mathbf{u} \ .$$

Then 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 (1) is satisfied, great – we have our approximate eigenvector $\mathbf{u}$. If not we have to shift.

How do we choose c? The project is an investigation of one approach.

Method for choosing the shift c: First compute some power of A, say A^{16}. Let $\mathbf{v}_1$ and $\mathbf{v}_2$ be the first two columns of A^{16}. There are two cases:

In the good case, $\mathbf{v}_1$ and $\mathbf{v}_2$ are nearly parallel – they are both nearly multiples of the same vector when one eigenvalue dominates. You recognize this case by

$$\mathbf{v}_1 \cdot \mathbf{v}_2 \approx \pm|\mathbf{v}_1||\mathbf{v}_2| \ . \tag{2}$$

(For a more quantitative version, let's say this is the case if the angle between $\mathbf{v}_1$ and $\mathbf{v}_2$ is less than one degree away from $0°$ or $180°$).

In this case we may hope that $\mathbf{v}_1$ itself is nearly an eigenvector, and we let $\mathbf{u}$ be the unit vector obtained by normalizing $\mathbf{v}_1$, and then

$$c = \mathbf{u} \cdot A\mathbf{u}$$

is our approximate eigenvalue, and we shift by c.

If $\mathbf{v}_1$ and $\mathbf{v}_2$ are not multiples of one another, we expect that $\mathbf{v}_1$ and $\mathbf{v}_2$ are linear combinations of the eigenvectors of A with the two largest eigenvalues. If so, then so are $\mathbf{w}_1 = A\mathbf{v}_1$ and $\mathbf{w}_2 = A\mathbf{v}_2$. Hence $\{\mathbf{v}_1, \mathbf{v}_2\}$ and $\{\mathbf{w}_1, \mathbf{w}_2\}$ are a basis for the same subspace. The 2×2 change of basis matrix B is defined by

$$W = VB$$

where $V = [\mathbf{v}_1, \mathbf{v}_2]$ and $W = [\mathbf{w}_1, \mathbf{w}_2]$. The formula for B is

$$B = (V^t V)^{-1} V^t W$$

as you see multiplying both sides of $W = VB$ by V^t.

Since B is 2×2, finding these eigenvalues is easy. Compute the eigenvalues of B, and define c to 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. *Extra credit: In your write up, explain why if $\{\mathbf{v}_1, \mathbf{v}_2\}$ and $\{A\mathbf{v}_1, A\mathbf{v}_2\}$, span exactly the same subspace, the the eigenvalues of B are eigenvalues of A. This fact is the motivation for the rule. This requires the choice of an appropriate similarity transform, but is not hard. Ask for a hint if you are stuck.*

Either way, we have our shift c defined, and now you replace A by $(A - cI)^{-1}$, and start over. 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

$$|A\mathbf{u} - \langle \mathbf{u} \cdot A\mathbf{u} \rangle \mathbf{u}| < \epsilon \tag{3}$$

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

Keeps running this loop until either (3) 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 16th powers with some other powers. How does this affect the efficiency of the method. Can you suggest some optimal policy for choosing the powers based on your experience?

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.

• 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.