**Laboratorio #12: VALORES CARACTERISTICOS**

**TABLE OF CONTENTS**

* [Introduction](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-12#INTRODUCTION)
* [The Rayleigh Quotient](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-12#THE_RAYLEIGH_QUOTIENT)
* [The Power Method](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-12#THE_POWER_METHOD)
* [The Inverse Power Method](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-12#THE_INVERSE_POWER_METHOD)
* [Using Shifts](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-12#USING_SHIFTS)
* [The QR Method](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-12#THE_QR_METHOD)
* [ASSIGNMENT](http://www.fim.utp.ac.pa/Members/fernando.castillo/metodos-numericos/laboratorio-parte-2/laboratorio-12#ASSIGNMENT)

**Introduction**

For any square matrix **A**, consider the equation **det(A-lambda\*I)=0**. This is a polynomial equation of degree **N** in the variable **lambda**. Hence there are exactly **N** complex roots that satisfy the equation. If the matrix **A** is real, then the complex roots occur in conjugate pairs. The roots are known as *eigenvalues* of **A**. Interesting questions include:

* how can we find one or more of these roots?
* when are the roots distinct?
* when are the roots real?

In textbook examples, the determinant is computed explicitly, the (usually cubic) equation is factored exactly, and the roots "fall out". This is not how a real problem will be solved.

If lambda is an eigenvalue of A, then A-lambda\*I is a singular matrix, and therefore there is at least one nonzero vector x with the property that (A-lambda\*I)\*x=0. This equation is usually written

**A \* x = lambda \* x**

Such a vector is called an *eigenvector* for the given eigenvalue. There may be as many as N linearly independent eigenvectors. In some cases, the eigenvectors are, or can be made, into a set of pairwise orthogonal vectors. Interesting questions about eigenvectors include:

* how do we compute an eigenvector?
* when will there be a full set of N independent eigenvectors?
* when will the eigenvectors be orthogonal?

In textbook examples, the singular system (A-lambda\*I)\*x=0 is examined, and by inspection, an eigenvector is determined. This is not how a real problem is solved.

Some useful facts about the eigenvalues lambda of a matrix A:

* A-1 has the same eigenvectors as A, and eigenvalues 1/lambda;
* A2 has the same eigenvectors as A, and eigenvalues lambda2;
* A+mu\*I has the same eigenvectors as A, and eigenvalues lambda+mu;
* if A is symmetric, all its eigenvalues are real;
* if B is invertible, then inv(B)\*A\*B has the same eigenvalues as A;

**The Rayleigh Quotient**

If a vector x is an exact eigenvector of a matrix A, then it is easy to determine the value of the associated eigenvalue: simply take the ratio of the I-th components of A\*x and x, for any index I that you like.

But suppose x is only an approximate eigenvector, or worse yet, just a wild guess. We could still compute the ratio of corresponding components for some index, but the value we get will vary from component to component. We might even try computing all the ratios, and averaging them, or taking the ratios of the norms. However, the preferred method for estimating the value of an eigenvalue uses the Rayleigh quotient.

The *Rayleigh quotient* of a matrix A and vector x is

**R(A,x) = (xT\*A\*x) / (xT\*x).**

The Rayleigh quotient is a scalar value whose magnitude generally lies between the smallest and largest magnitudes of the eigenvalues of the matrix. If the matrix A is symmetric, then all eigenvalues are real, and we may write the simpler statement:

**Lambdamin <= R(A,x) <= Lambdamax**

When x is an approximate eigenvector of A, the Rayleigh quotient is a very accurate estimate of the corresponding eigenvalue (if real) or the magnitude of the eigenvalue (if complex).

M File: write an M file called *rayleigh.m* which computes the Rayleigh quotient. It should have the form:

**function r = rayleigh ( A, x )**

Exercise: Copy the M file *eigen\_test.m* from the web page. This file contains 5 test problems. The matrix you get by calling A=eigen\_test(1) has eigenvalues 1/2, -1, and 2. Compute the value of the Rayleigh quotient for a variety of vectors x. Can you find a vector for which the quotient is exactly 2? More than 2? Less than -1?

x R(A,x)   
 [ 1, 2, 3] -0.57   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_

What happens to the Rayleigh quotient when you look at a sequence of vectors x, A\*x, A\*A\*x, ...?

The Rayleigh quotient R(A,x) will give us a way to approximate eigenvalues from an approximate eigenvector. Now we have to figure out a way to come up with an approximate eigenvector.

**The Power Method**

In many physical and engineering applications, the largest eigenvalue associated with a system represents the dominant and most interesting mode of behavior. For a bridge or support column, the eigenvalue might reveal the maximum load, and the eigenvector the shape of the object as it begins to fail under this load. For a concert hall, the dominant eigenvalue of the acoustic equation reveals the lowest resonating frequency. Hence, a simple means of approximating just one eigenvalue (the one of greatest magnitude) might be enough for some cases.

The *power method* is a simple iterative method for seeking the largest magnitude eigenvalue and its eigenvector. To discuss this method, let's begin with the matrix returned by A=eigen\_test(1). I constructed this matrix to have a particularly simple structure. The eig command will show you the eigenvectors as well as the eigenvalues, if you use the command in the form:

**[ V, lambda ] = eig ( A )**

The quantity V is actually a matrix, storing the N eigenvectors as columns. Record the eigenvalues and eigenvectors of A on this worksheet:

**Eigenvalue: \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_  
 Eigenvector  
 Component 1: \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_  
 Component 2: \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_  
 Component 3: \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_**

The *power method* tries to determine the largest magnitude eigenvalue, and the corresponding eigenvector, of a matrix, by computing (scaled) vectors in the sequence:

**xk+1 = A \* xk**

If we include the scaling, and an estimate of the eigenvalue, we get an algorithm that looks like this:

**Starting with any nonzero vector x\_old, repeat:  
Set x: = A \* x\_old              
Set r: = Rayleigh(A,x)               
Divide x by its L2 norm.    
If the maximum entry of x is negative, set x=-x**

The last step is only done to make convergence easier to detect.

M File: write a simple M file that takes one step of the power method. Your file should have the form:

**[x, r] = pow ( A, xold )**

Exercise: Using your power method code, try to determine the dominant eigenvalue, and corresponding eigenvector, of the 5 *eigen\_test* matrices. For consistency, use a starting vector of all 1's. Compare your results for matrix 1 with the results from the MATLAB eig command you recorded earlier.

**Matrix Eigenvalue Eigenvector  
 1 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 2 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 3 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 4 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
 5 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

One of these examples will probably give you slow convergence, and another may give you what seems like oscillatory behavior. Check the eigenvalues of these matrices. Check the norms of the eigenvalues. Can you make any observations?

**The Inverse Power Method**

The inverse power method reverses the iteration step of the power method. We write:

**A \* xk+1 = xk**

or, equivalently,

**xk+1 = A-1 \* xk**

In other words, this looks like we are just doing a power method iteration, but now for the matrix A-1. Leaving aside *why* we would want to do this, we can immediately assume that this process will tend to converge to the eigenvector associated with the largest eigenvalue of A-1. Interestingly enough, the eigenvectors of A and A-1 are the same. Moreover, if v is an eigenvector of A for the eigenvalue LAMBDA, then it is an eigenvector of A-1 for 1/LAMBDA and vice versa.

This means that we can freely choose to study the eigenvalue problem of either A or A-1, and easily convert information from one problem to the other. Is there any advantage to looking at A-1?

Well, the inverse power iteration will find us the biggest eigenvalue of A-1, but that's the eigenvalue of A that's closest in magnitude to zero.

The method looks like this:

**Starting with any nonzero vector x\_old, repeat:   
Solve A\*x: = x\_old  
Set r: = Rayleigh(A,x)   
Divide x by its L2 norm.    
If the maximum entry of x is negative, set x=-x**

M File: write a simple M file that takes one step of the inverse power method. Your file should have the form:

**[x, r] = inpow ( A, xold )**

Exercise: Using your inverse power method code, try to determine the smallest eigenvalue of the matrix *eigen\_test(5)*, starting from the vector of all 1's.

**Step x R(A,x)  
 0 [ 1.00, 1.00, 1.00 ] 4.33  
 1 [ 0.87, 0.22, 0.44 ]\_\_\_\_\_\_\_\_\_ 3.00\_\_\_\_\_\_  
 2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_  
 3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_  
 4 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_  
 5 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_**

**Using Shifts**

A major limitation of the power method was that it can only find the eigenvalue of largest magnitude. Similarly, the inverse power iteration seems only able to find the eigenvalue of smallest magnitude. But suppose we make what seems one tiny change. Instead of working with the matrix A, let's work with the matrix A+3\*I, where we have shifted the matrix by adding 3 to every diagonal entry. How would this shift affect the inverse power method?

Well, we already said the method finds the eigenvalue of smallest magnitude. Denote the smallest eigenvalue of A+3\*I by mu. But mu is an eigenvalue of A+3\*I because A+3\*I-mu\*I is singular. But that means A-(mu-3)\*I is singular. In other words, lambda=mu-3.0 is an eigenvalue of A, and is the eigenvalue of A that is closest to 3.

In general, the idea of shifting allows to focus the attention of the inverse power method on seeking the eigenvalue closest to any particular value we care to name. This also means that if we have an estimated eigenvalue, we can speed up convergence by using this estimate in the shift. A shifted inverse power method in which the shifts are adjusted on each step can be rapidly converging.

We are not going to worry here about the details of the expense of shifting and refactoring the matrix. (A real algorithm for a large problem would be very concerned about this!) Our simple-minded method looks like this:

**Starting with any nonzero vector x\_old and eigenvalue estimate r\_old, repeat:   
Solve (A-r\_old\*I)\*x: = x\_old   
Set r: = r\_old + Rayleigh(A-r\_old\*I,x)   
Divide x by its L2 norm.    
If the maximum entry of x is negative, set x=-x**

M File: write a simple M file that takes one step of the shifted inverse power method. Your file should have the form:

**[x, r] = inpows ( A, xold, rold )**

Exercise: Using your shifted inverse power method code, we are going to search for the "middle" eigenvalue of matrix *eigen\_test(5)*. Assume we know that this mising eigenvalue must be within the range [ 1.5, 3.5], and start with a vector of all 1's. Start with a shift of 2.5, and thereafter, adjust your shift to the value of R(A,x).

**Step x Shift or R(A,x)  
 0 [ 1.00, 1.00, 1.00 ] 2.5  
 1 [ 0.39, 0.91, -0.13 ]\_\_\_\_\_\_\_\_ 3.34\_\_\_\_\_\_  
 2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_  
 3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_  
 4 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_  
 5 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_**

Note that the cost of allowing the shift to vary is that we can't be sure that the code won't veer off to computing the largest or smallest eigenvalues. This could happen if the initial vector had a large component of the corresponding eigenvector. If you are interested, see if you can come up with a starting vector that causes your code to converge to the largest or smallest eigenvalue instead.

**The QR Method**

So far, the methods we have discussed seem suitable for finding one eigenvalue and eigenvector at a time. In order to find more eigenvalues, we'd have to do a great deal of special programming. Moreover, these methods are surely going to have trouble if the matrix has repeated eigenvalues, distinct eigenvalues of the same magnitude, or complex eigenvalues. The *QR method* is a more modern method which handles these sorts of problems in a uniform way, and computes all the eigenvalues and eigenvectors at one time.

The QR method repeats the following two steps:

* Given a matrix Ak, construct the QR factorization;
* Set Ak+1 = R \* Q.

The sequence of matrices Ak is orthogonally similar to the original matrix A, and hence has the same eigenvalues. Under fairly general conditions, the sequence of matrices Ak converges to

* a diagonal matrix if the A was symmetric
* an upper triangular matrix if the eigenvalues are all real
* an "almost" upper triangular matrix, where the main subdiagonal will have nonzero entries only when there is a complex conjugate pair of eigenvalues.

**The MATLAB command**

**[ Q, R ] = qr ( A )**

computes the QR factorization (as does our own *h\_factor* routine). The basic QR iteration is so simple that it's really not worth writing a routine to take a single step of it.

Exercise: Try out the QR iteration for the *eigen\_test* matrices 1, 2 and 3. For matrix 2, which has 4 distinct eigenvalues of equal norm, you may find that the process does not behave as we would hope.

Serious implementations of the QR method save information about each step, which defines the eigenvectors. The algorithm also uses shifts to try to speed convergence. Some eigenvalues can be determined early in the iteration, which speeds up the process even more.

**Assignment**

We are going to look at a simple version of the shifted QR method. To keep things distinct from the regular method, we'll call the matrix B here.

Let our initial matrix be B0. Step k of the shifted QR method works as follows:

* determine a shift sk;
* compute Q, R, the factors of Bk-1-sk\*I;
* define Bk=sk\*I + R \* Q.

 Assignment: compare 5 steps of the QR and shifted QR algorithms, starting with A0=B0=dif2(4), the difference matrix. For the shifted QR method, always use a shift of -1. At the end of 5 steps, record the 4 diagonal elements of your two matrices A5 and B5.

**diag(A5) diag(B5)  
 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_   
 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_**