Eigenvalue problems I: Introduction and Jacobi Method

Sam Sinayoko

Computational Methods 8

Contents

```
import numpy as np
import scipy as sp
import scipy.linalg
import matplotlib
from IPython.html.widgets import interact
from IPython.display import Image, YouTubeVideo
try:
    %matplotlib inline
except:
    # not in notebook
    pass
LECTURE = False
if LECTURE:
    size = 20
    matplotlib.rcParams['figure.figsize'] = (10, 6)
    matplotlib.rcParams['axes.labelsize'] = size
    matplotlib.rcParams['axes.titlesize'] = size
    matplotlib.rcParams['xtick.labelsize'] = size * 0.6
    matplotlib.rcParams['ytick.labelsize'] = size * 0.6
import matplotlib.pyplot as plt
```

1 Learning Outcomes

- Define an eigenvalue problems
- Give two examples of applications from the physical world involving the solution of eigenvalue problems.
- Solve a simple eigenvalue problem analytically.
- Explain in your own words the Jacobi algorithm for eigenvalue problems.

2 Introduction

Along with the differential equations we have covered to date, there is another class of problems that are important in science and engineering: eigenvalue problems. They are often associated with vibrations and they also

have some interesting and unique solvers which bring together a number of the technique and methods we have covered.

2.1 Eigenvalue problems and resonance problems

All mechanical structures have *natural frequencies* at which they like to vibrate. Those natural frequencies depends on the distribution of mass in the structure, as well as the stiffness of the elements that make up that structure. If the structure is excited at any of these natural frequencies, it will resonate: a small excitation will produce a very large response. This response can be large enough to to lead to a break up of the structure.

For example, a continuous tonal sound at the same frequency as one of the natural frequencies of a glass can make it break.

```
YouTubeVideo('17tqXgvCN0E')
```

It is therefore crucial to design engineering structures so that the natural resonances will not be excited. Failure to predict such resonances can lead to catastrophic collapse, as in the well known case of the Tacoma Narrows Bridge Collapse

```
YouTubeVideo('XggxeuFDaDU')
```

Another recent exemple is provided by the Millenium bridge, which had to be closed shortly after its inauguration because of a resonance problem.

```
YouTubeVideo('eAXVa__XWZ8')
```

Note that, for each natural frequency, the response follows a particular shape called the *mode shape*. The natural frequencies correspond to the *eigenvalues* of the system, while the mode shapes correspond to *eigenvectors*.

2.2 Eigenvalue problems and wave phenomena

2.2.1 Acoustics

In addition to vibration problems, eigenvalue problems are ubiquitous in acoustics or optical phenomena. For example, when sound waves propagate through a pipe, they take the form of particular duct modes, corresonding to the eigenvectors of the Helmhotz equation. Similarly, sound field in a closed room fluctates according to the acoustic modes of the room. The sound quality of a concert hall therefore depends on the acoustic modes of the room.

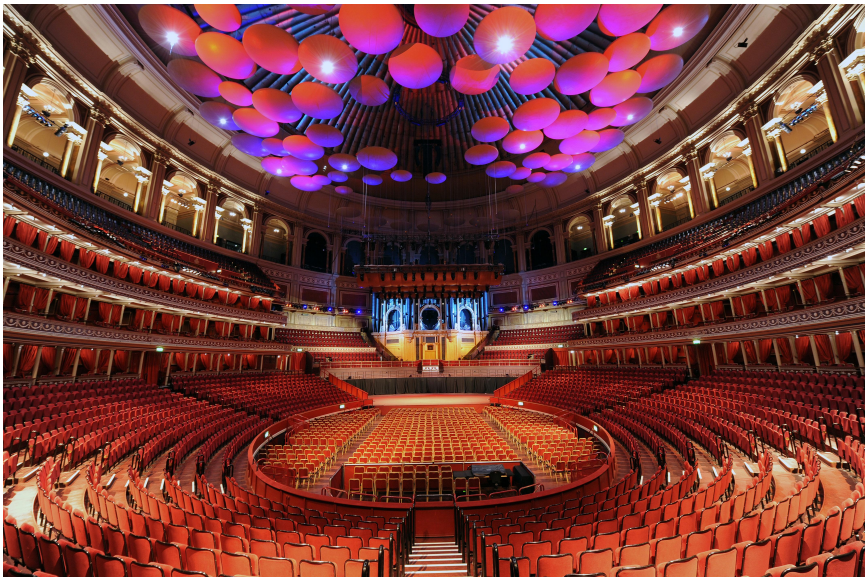

Figure 1: Picture of the Royal Albert Hall, showing circular pads hanging off the roof to improve the sound quality.

2.2.2 Optics

Other examples of eigenvalue problems in optics include the scattering of light by the wings of butterflies, the feathers of peacocks and the surface of opals. This produces colors that are due to structural coloring rather than pigmentation. The stunning colors they produce depend on the refraction index of these optical systems, which can be obtained by solving an eigenvalue problem.

3 Eigenvalue problem definition

The genrealised eigenvalue problem for two $N \times N$ matrices A and B is to find a set of *eigenvalues* λ and *eigenvectors* $x \neq 0$ such that

$$Ax = \lambda Bx. \tag{1}$$

If B is the identity matrix, this equation reduces to the eigenvalue problem:

$$Ax = \lambda x. \tag{2}$$

Figure 2: Opals exhibit stunning colors without pigments but via structural coloring.

We will focus on the latter equation as techniques to solve the generalised problem are an extension of solving the reduced problem and can be found in the literature [1]

4 Analytical solution

4.1 Theory

We can solve the eigenvalue problem as follows. If there exists $x \neq 0$ such that

$$Ax = \lambda x, \tag{3}$$

then

$$(A - \lambda I)x = 0. \tag{4}$$

Figure 3: Peacocks feathers are another example of structural coloring.

This must be true for each eigenvalue-eigenvector pair. Since x must be non-zero, the above equation has a solution if and only if

$$\det(A - \lambda I) = 0. \tag{5}$$

This leads to a polynomial equation of order n, called the *characteristic polynomial*, whose solutions are the eigenvalues of matrix A. It is then generally easy to find an eigenvector for each of the eigenvalue λ by finding a solution x satisfying $(A - \lambda I)x = 0$

4.2 Example

Let

$$A = \begin{pmatrix} 3 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 3 \end{pmatrix}. \tag{6}$$

We can find the eigenvalues by solving

$$\det(A-\lambda I)=\begin{vmatrix}3-\lambda & -1 & 0\\ -1 & 2-\lambda & -1\\ 0 & -1 & 3-\lambda\end{vmatrix}=0. \tag{7}$$

This gives

$$(3-\lambda)\begin{vmatrix}2-\lambda & -1\\ -1 & 3-\lambda\end{vmatrix}-(-1)\begin{vmatrix}-1 & -1\\ 0 & 3-\lambda\end{vmatrix}+(0)\begin{vmatrix}-1 & 2-\lambda\\ 0 & -1\end{vmatrix}=0. \tag{8}$$

Thus we have the characteristic polynomial

$$(3-\lambda)(2-\lambda)(3-\lambda)-(3-\lambda)-1(3-\lambda)=0 \tag{9}$$

$$-\lambda^3+8\lambda^2-21\lambda+18-3+\lambda-3+\lambda=0 \tag{10}$$

$$-\lambda^3+8\lambda^2-19\lambda+12=0. \tag{11}$$

This can be solved to give the solutions

$$\lambda_1=1, \qquad \lambda_2=4, \qquad \lambda_3=3. \tag{12}$$

Solving $(A-\lambda I)x=0$ for each of these eigenvalues yields the corresponding eigenvectors, for example

$$x_1=\begin{pmatrix}1\\2\\1\end{pmatrix}, \qquad x_2=\begin{pmatrix}1\\-1\\1\end{pmatrix}, \qquad x_3=\begin{pmatrix}1\\0\\-1\end{pmatrix}. \tag{13}$$

Note that the eigenvectors are not unique and can be multiplied by arbitrary constants.

Alternatively, we can also use the SymPy module in Python (or other symoblic mathematical software such as Mathematica or Maple).

```
import sympy as sy
L = sy.symbols('L')
M = sy.Matrix([[3 - L, -1, 0],
               [-1, 2 - L, -1],
               [0, -1, 3 - L]])
print 'Characteristic Polynomial: P(L) = ', M.det()

print 'Eigenvalues: ', sy.solve(M.det())
```

```
Characteristic Polynomial: P(L) =  -L**3 + 8*L**2 - 19*L + 12
Eigenvalues:  [1, 3, 4]
```

But writing out the characteristic polynomial and solving it is generally not a good way to solve as the matrix gets larger.

Clearly if the matrix was of the form

$$D = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}, \tag{14}$$

then we would find the eigenvalues trivially since we would have

$$\det(D - \lambda I) = 0 \tag{15}$$

$$\begin{vmatrix} a-\lambda & 0 & 0 \\ 0 & b-\lambda & 0 \\ 0 & 0 & c-\lambda \end{vmatrix} = 0 \tag{16}$$

$$(a-\lambda)(b-\lambda)(c-\lambda) = 0, \tag{17}$$

so the the eigenvalues are a, b, c. Thus, our first strategy for finding the eigenvalues will be to attempt to transform the matrix into a diagonal matrix.

If P is the transformation matrix that transforms A into a diagonal matrix D , then

$$D = P^{-1}AP, \tag{18}$$

which can be easily shown by noting that the transformation matrix transforms a vector x expressed in the original basis, into a vector x' expressed in the new basis, as

$$x = Px'. \tag{19}$$

It can also be shown easily from the above equation that the columns of the matrix transformation give the coordinates of the eigenvalues in the original basis.

5 Numerical solution: the Jacobi Method fo Eigenvalues

5.1 Algorithm

This method is a fairly robust way to extract all of the eigenvalues and eigenvectors of a symmetric matrix. Whilst it is probably only appropriate

to use for matrices up to 20 by 20, the principles of how this method operates underpin a number of more complicated methods that can be used more generally to find all of the eigenvalues of a matrix (assuming that finding such eigenvalues is actually a well-posed / stable / sensible problem).

The method is based on a series of rotations, called Jacobi or Givens rotations, which are chosen to eliminate off-diagonal elements while preserving the eigenvalues. Whilst successive rotations will undo previous et zeros, the off-diagonal elements get smaller until eventually we are left with a diagonal matrix. By accumulating products of the transformations as we proceed we obtain the eigenvectors of the matrix. See for example [2, 3] for additional details.

Consider the transformation matrix $P(p, q, \theta)$ of the form

$$P = \begin{pmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & c & & s & & & \\ & & & & 1 & & & & \\ & & & -s & & c & & & \\ & & & & & & 1 & & \\ & & & & & & & \ddots & \\ & & & & & & & & 1 \end{pmatrix}, \tag{20}$$

where all diagonal eleents are unity apart from two elements c in rows p and q, and all off-diagonal elements are zero apart from the elements s and $-s$ (in rows and columns p and q). Thus, if e_p and e_q are the \$p\$$^{\text{th}}$ and \$q\$$^{\text{th}}$ vectors of an orthonormal basis, and if

$$c = \cos\theta, \qquad\qquad s = \sin\theta, \tag{21}$$

then P represents a rotation of angle θ in the (oriented) (e_p, e_q) plane, which leaves all other basis vectors unchanged.

Applying this transformation matrix to the symmetric matrix A yields

$$\widetilde{A} = P^T A P, \tag{22}$$

which is also a symmetric matrix, and whose eigenvalues are the same as those of A. Furthermore, this operation preserves the Frobenius norm, i.e.

$$\sum_{i,j} \widetilde{A}_{ij}^2 = \sum_{i,j} A_{ij}^2, \tag{23}$$

so our rotations are not changing the "size" of A, only redistributing the coefficients.

Most importantly, for each column q in A, a matrix $P(p, q, \theta)$ can be defined such that $A_{p,q} = 0$ (see e.g. [2, 3] or Wikipedia). A desirable side effect is that

$$\widetilde{A}_{qq}^2 + \widetilde{A}_{pp}^2 \geq A_{qq}^2 + A_{pp}^2, \tag{24}$$

i.e. the diagonal elements increase in magnitude when going from A to $\widetilde{A}$. This effect is especially large if row p is chosen such that A_{pq} is the largest coefficient in the \$q\$$^{\text{th}}$ column of A. Thus, we can rotate our matrix A in such a way that the large off-diagonal coefficients become zero, while the on-diagonal coefficients increase in magnitude.

If we carry on this process iteratively, selecting p, q such that A_{pq} is the largest off-diagonal element, we will eventually obtain a matrix D that is diagonal to machine precision:

$$D = P^T A P, \tag{25}$$

where

$$P = P_1 P_2 P_3 \cdots P_m, \tag{26}$$

is the product of the successive Jacobi rotation matrices.

As explained in the previous section, the diagonal coefficients in D represent the eigenvalues, while the columns of P give the coordinates of the eigenvectors in our original basis.

Sample code for this algorithm can be found in [3].

5.2 Implementation

```
def maxelem(a):
    """Return (amax, k, l), the value and indices of the off-diagonal
    element in 2D array a
    """
    n = len(a)
    amax = 0.0
    for i in range(n-1):
        for j in range(i+1,n):
            if abs(a[i,j]) >= amax:
                amax = abs(a[i,j])
                k = i
                l = j
    return amax,k,l
```

Define a function for rotating the matrix in place:

```
def rotate(a, p, k, l):
    """Rotate input matrix a in pla
    transformation matrix p
    """
    n = len(a)
    aDiff = a[l,l] - a[k,k]
    if abs(a[k,l]) < abs(aDiff)*1.0
        t = a[k,l]/aDiff
    else:
        phi = aDiff/(2.0*a[k,l])
        t = 1.0/(abs(phi) + np.sqrt
        if phi < 0.0:
            t = -t
    c = 1.0/np.sqrt(t**2 + 1.0); s
    tau = s/(1.0 + c)
    temp = a[k,l]
    a[k,l] = 0.0
    a[k,k] = a[k,k] - t*temp
    a[l,l] = a[l,l] + t*temp
    # Case of i < k
    for i in range(k):
        temp = a[i,k]
        a[i,k] = temp - s*(a[i,l] +
        a[i,l] = a[i,l] + s*(temp -
    # Case of k < i < l
    for i in range(k+1,l):
        temp = a[k,i]
        a[k,i] = temp - s*(a[i,l] +
        a[i,l] = a[i,l] + s*(temp -
    # Case of i > l
    for i in range(l+1,n):
        temp = a[k,i]
        a[k,i] = temp - s*(a[l,i] +
        a[l,i] = a[l,i] + s*(temp -
    # Update transformation matrix
    for i in range(n):
        temp = p[i,k]
        p[i,k] = temp - s*(p[i,l] +
        p[i,l] = p[i,l] + s*(temp -
```

Implement the Jacobi eigenvalue algorithm

```
def jacobi_eig(a, tol=1.0e-9): # Jacobi met
    """ lambda, x = jacobi_eig(a, tol=1.0e-

    Solution of std. eigenvalue problem [a]
    by Jacobi's method. Returns eigenvalues
    and the eigenvectors as columns of matr
    """
    n = len(a)
    # Set limit on number of rotations
    nbrot_max = 5*(n**2)
    # Initialize transformation matrix
    p = np.identity(n)*1.0
    # Jacobi rotation loop
    for i in range(nbrot_max):
        amax, k, l = maxelem(a)
        if amax < tol:
            return np.diagonal(a),p
        rotate(a, p, k,l)
        # # Extra debug
        # print 'Step:', i
        # print a
        # print'----------------'
    print 'Jacobi method did not converge'
```

5.3 Testing

Test our implementation:

```
#Create full matrix
A= np.array([[3.,-1,0], [-1,2,-1] , [0 , -1 , 3]])
#A= np.array([[8.,-1,3,-1], [-1,6,2,0] , [3 , 2 , 9, 1],
w,v = jacobi_eig(A,tol =  1.0e-9)

# Now sort them into order with argsort
idx = w.argsort()
w = w[idx]
v = v[:,idx]

# Normalize the eigenvectors so the first coordinate equa
v = v / v[0].reshape(1, -1)
print 'Eigenvalues:\n', w
print ''
print 'Eigenvectors:\n', v
```

```
Eigenvalues:
[ 1.  3.  4.]

Eigenvectors:
[[  1.00000000e+00   1.00000000e+00   1.00000000e+00]
 [  2.00000000e+00  -3.66732745e-10  -1.00000000e+00]
 [  1.00000000e+00  -9.99999999e-01   1.00000000e+00]]
```

6 Self study

Solve the eigenvalue problem for matrix A below using an analytical approach (e.g. with SymPy), and using the Jacobi eigenvalue method.

$$A = \begin{pmatrix} 8 & -1 & 3 & -1 \\ -1 & 6 & 2 & 0 \\ 3 & 2 & 9 & 1 \\ -1 & 0 & 1 & 7 \end{pmatrix} \tag{27}$$

7 Conclusions

- Eigenvalue problems: given a matrix A, find scalars λ, called eigenvalues, and non-zero vectors x, called eigenvectors, such that $Ax = \lambda x$.

- Give two examples of applications from the physical world involving the solution of eigenvalue problems: vibrations (bridge dynamics) and the design of concert halls (acoustic modes).
- Solve a simple eigenvalue problem analytically: calculate the characteristic polynomial $\det(A - \lambda I)$ and find the eigenvalues, then find an eigenvector for each eigenvalue by solving $(A - \lambda I)x = 0$.
- Explain in your own words the Jacobi algorithm for eigenvalue problems

8 References

1. G. H. Golub and C. F. Van Loan, Matrix Computations, John Hopkins University Press; third edition (1996) ISBN-10:0801854148
2. W. H. Press, S.A. Teukolsky et al, Numerical Recipes in C.
3. J. Kiusalaas, Numerical Methods in Engineering with Python, Cambridge University Press (2010).