Eigenvalue problems II: 1D Helmholtz Equation

Sam Sinayoko

Computational Methods 9

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

By the end of this lecture, you should be able to

- Transform the 1D wave equation into the 1D Helmholtz equation.
- Discretise the Helmholtz equation using the finite difference method to obtain an eigenvalue problem.
- Use the Jacobi eigenvalue method to solve the eigenvalue problem and plot the first few modes / eigenvectors.

2 Introduction

The previous lecture introduced eigenvalue problems, in which given a matrix A, we seek the eigenvalues λ such and the (non-zero) eigenvectors x such that

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

We showed how the Jacobi eigenvalue method can be used to obtaining the eigenvalues and eigenvectors when A is a symmetric matrix. The Jacobi method is an iterative method that uses rotation matrices, or so-called Jacobi or Givens matrices, to transform A into a diagonal matrix.

In this lecture, we will show how some Partial Differential Equations lead to eigenvalue problems by solving the 1D Helmholtz equation.

3 1D Helmholtz equation: the vibrating string

3.1 Partial Differential Equation

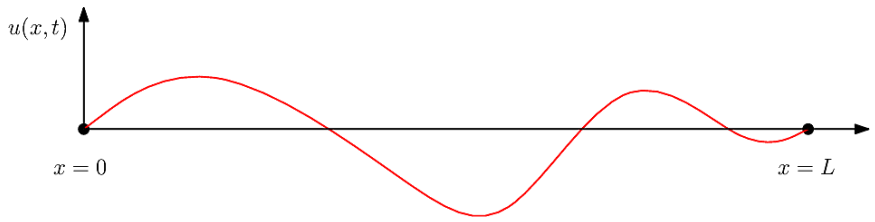

Figure 1: Vertical displacement $u(x,t)$ of a string of length L clamped at both ends.

Consider a guitar string clamped at both ends. When the string is displaced vertically by a small amount and then released, its displacement $u(x,t)$ satisfies the wave equation (see Wikipedia's article on the vibrating string for a derivation based on Newton's second law):

$$\frac{\partial^2}{\partial x^2}u(x,t) - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}u(x,t) = 0, \tag{2}$$

where c is a constant that depends on the material of the string and on its tension. As we discussed in lecture 07 (c.f. hyperbolic equations section), this equation indicates that u takes the form of waves propagating along the string at speed c.

This equation is hyperbolic and somewhat challenging to solve, but if we express the displacement in the form

$$u(x,t) = \mathrm{Re}\left(U(x)e^{i\omega t}\right) = U(x)\cos(\omega t) \tag{3}$$

where U is a real function of x, and ω is an arbitrary frequency, then the wave equation reduces to

$$U''(x) + k^2 U(x) = 0, \tag{4}$$

where $k = \omega/c$ is called the wavenumber. We have used the *separation of variable* technique which is a powerful way of solving PDEs.

This equation is called the Helmholtz equation and it is parabolic. Thus, for every frequency ω, there is a function $U(x)$ such that the displacement $u(x,t)$ satisfies the wave equation. It seems that ω must be related to an eigenvalue of the system, and that $U(x)$ must be the corresponding eigenvector. Let's demonstrate that this is indeed the case by discretising the Helmholtz equation using the Finite Difference Method.

3.2 Difference equation

As usual, let's start with only a few unkowns. Figure 2 represents the string using 5 uniformly spaced nodes, with 2 nodes at the left and right boundaries where the displacements are 0, and 3 nodes in between, U_0, U_1 and U_2, where the displacement in unkown.

The term U'' is our familiar Laplace operator in 1D, which we first discretized in lecture 04 using the first central approximation for the second derivative:

$$U''(x) = \frac{U(x-h) - 2U(x) + U(x+h)}{h^2} + O(h^2). \tag{5}$$

This leads to the following difference equation, accurate to order 2:

$$U(x-h) - 2U(x) + U(x+h) + h^2k^2U(x) = 0, \tag{6}$$

which can be put in the form

$$-U(x-h) + 2U(x) - U(x+h) = \lambda U(x), \qquad \text{where} \quad \lambda \equiv h^2k^2. \tag{7}$$

If we can solve the above equation for U and λ, then we can recover the wavenumber from $k = \sqrt{\lambda}/h$.

3.3 Discretized equation

We can now use our difference equation for each (inner) node in our domain to obtain a system of equations, which we will put in matrix form. For our three unkowns U_0, U_1 and U_2, we have

$$\begin{cases} -0 + 2U_0 - U_1 = \lambda U_0, \\ -U_0 + 2U_1 - U_2 = \lambda U_1, \\ -U_1 + 2U_2 - 0 = \lambda U_2, \end{cases} \tag{8}$$

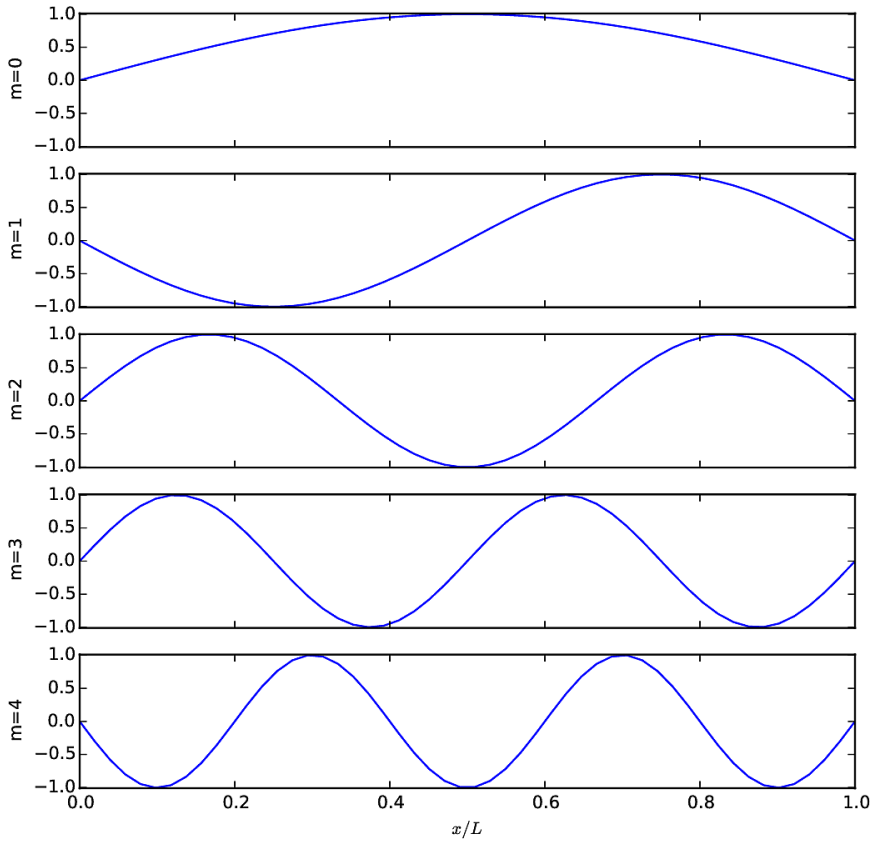

Figure 2: Vertical displacement $U(x)$ over a uniform mesh of guitar string of length L clamped at both ends, with unknown 3 mesh points.

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} U_0 \\ U_1 \\ U_2 \end{pmatrix} = \lambda \begin{pmatrix} U_0 \\ U_1 \\ U_2 \end{pmatrix}. \tag{9}$$

As expected, this is an eigenvalue problem and each eigenvalue / eigenvector pair gives us a solution to the Helmholtz equation, from which we can retrieve a solution to the original wave equation. This solution is called an acoustic mode. For a string with N degrees of freedom, you can find N modes. The physical solution would be a linear combination of all these modes.

We recognise our tri-diagonal Laplacian matrix, with 2s on the diagonal and -1s on the first upper and lower diagonals. So for a string with N degrees

of freedom, the problem would look like

$$\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix} \begin{pmatrix} U_0 \\ U_1 \\ \vdots \\ \vdots \\ U_{N-2} \\ U_{N-1} \end{pmatrix} = \lambda \begin{pmatrix} U_0 \\ U_1 \\ \vdots \\ \vdots \\ U_{N-2} \\ U_{N-1} \end{pmatrix} \tag{10}$$

```
def get_A(N):
    """Return our 2nd order Laplacian matrix"""
    offset = 1
    upper = -np.diag(np.ones(N - 1), offset)
    lower = -np.diag(np.ones(N - 1), -offset)
    diag = 2 * np.diag(np.ones(N))
    A = diag + upper + lower
    return A

N = 10
A = get_A(N)
print 'A = \n', A
```

```
A =
[[ 2. -1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [-1.  2. -1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. -1.  2. -1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. -1.  2. -1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. -1.  2. -1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -1.  2. -1.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. -1.  2. -1.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. -1.  2. -1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. -1.  2. -1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. -1.  2.]]
```

3.4 Numerical Solution

What do these modes look like? Let's solve the above eigenvalue problem. Since the matrix is symmetric, we can use our Jacobi code from lecture 08, which you can download from snippets tab on the course

website (lecture08.py). Here we will implement an additional function `jacobi_eig_sorted` that sorts the eigenvalues from small to large and reorders the columns in the matrix of eigenvectors accordingly. This is easily done using the NumPy's `argsort` array method.

```
from lecture08 import jacobi_eig
def jacobi_eig_sorted(a, tol=1e-9):
    """Solve the eigenvalue problem aw = v.w and return sorted
    eigenvalues v and eigenvectors w."""
    eigvals, eigvecs = jacobi_eig(a, tol)
    idx = eigvals.argsort()
    s_eigvals = eigvals[idx]  # sorted
    s_eigvecs = eigvecs[:,idx] # sorted
    return s_eigvals, s_eigvecs

eigvals, U = jacobi_eig(A)
# eigenvalues are in vector eigvals and the eigenvectors are the columns of U
mode = 0
print 'eigenvalue #%d: \n\t' % mode, eigvals[mode]
print 'eigenvector #%d: \n\t' % mode, U[:, mode]
```

```
eigenvalue #0:
        3.30972146789
eigenvector #0:
        [ 0.3222527  -0.42206128  0.23053002  0.12013117 -0.38786839  0.38786839
 -0.12013117 -0.23053002  0.42206128 -0.3222527 ]
```

We can use an embedding function as in lecture 06 to add the boundary points where the displacement is 0.

```
def embed(u):
    """Add a 0 at the start and end of the array"""
    return np.concatenate([(0,), u, (0,)])
# Test
print embed([1,2,3])
```

```
[0 1 2 3 0]
```

Let's compute the matrix A and its eigenvalues, and plot the vertical displacement of our string for various modes:

```
N = 50
L = 1.0
A = get_A(N)
eigvals, eigvects = jacobi_eig_sorted(A)

def plot_mode(mode=0, xlabel=False):
    lbda = eigvals[mode]
    U = embed(eigvects[:, mode])
    h = 1.0 / (N + 1.0)  # beware of the indivision bug
    k = np.sqrt(lbda) / h

    x = np.linspace(0, 1.0, N + 2)  # x / L
    plt.plot(x, U/max(U))
    plt.ylim([-1.01, 1.01])
    if xlabel:
        plt.xlabel(r'$x/L$')
# Test
if LECTURE:
    plot_mode(mode=1)
```

```
plt.figure(1)
plt.clf()

if LECTURE:
    i = interact(plot_mode, mode=(0, 10))
else:
    # Plot the first few modes
    nbmodes = 5
    fig, axes = plt.subplots(5,1, sharex=True, figsize=(8, 8))
    for mode in range(nbmodes):
        plt.sca(axes[mode]) # set the current axis to 'ax'
        plot_mode(mode)
        plt.ylabel('m=%d' % mode)
    plt.xlabel(r'$x/L$')
    plt.savefig('fig09-02.pdf')
```

3.5 Interpretation

Using the above approach, we find the discrete list of eigenvalues (λ_i) (and eigenvectors) for our string. Since $\omega/c = k = \sqrt{\lambda}/h$, this translates into a finite number of frequencies (ω_i). These are precisely the natural frequencies that we discussed in the previous lecture.

If the string is excited at one of these natural frequencies ω_i, it will vibrate with the shape defined by the associated eigenvector $U_{\omega_i}(x)$, or, in the time domain, $u(x,t) = U_{\omega_i}\cos(\omega_i t)$. This is beautifully illustrated in this YouTube video.

```
if LECTURE:
    YouTubeVideo('-gr7KmTOrx0')
```

3.6 Self Study

1. You can animate the mode shapes by plotting the $u(x,t)$ instead of $U(x)$ for a given eigenvector, to reproduce the standing waves illustrated in the above video. Add a keyword parameter `omega_t=0` to the `plot_mode` function and use it to plot the displacement $u(x,t)$ along the string as a function of x, for a fixed non-dimensional time `omega_t`. Modify the call to interact in the above cell to visualise the evolution of the mode shape with time over a period ($0 \leq \omega t \leq 2\pi$).

2*. (Hard) Think about how you could generalise this lecture to 2D or 3D. In 2D a similar problem would be to obtain the acoustic modes in a rectangular or a cylindrical duct. Have a look at my MSc dissertation for an analytical solution to the cylindrical problem, which you can use to validate your code.

4 Conclusions

- Use a Fourier expansion of the form $u(x,t) = U(x)\cos(\omega t)$ to transform the wave equation into a Helmholtz equation.
- Use the 2nd order Laplacian stencil to discretise the Helmholtz equation into an eigenvalue problem.
- Use the Jacobi matrix to solve the problem and plot the mode shapes.

Next week we will look at alternatives to the Jacobi eigenvalue method for solving eigenvalue problems.

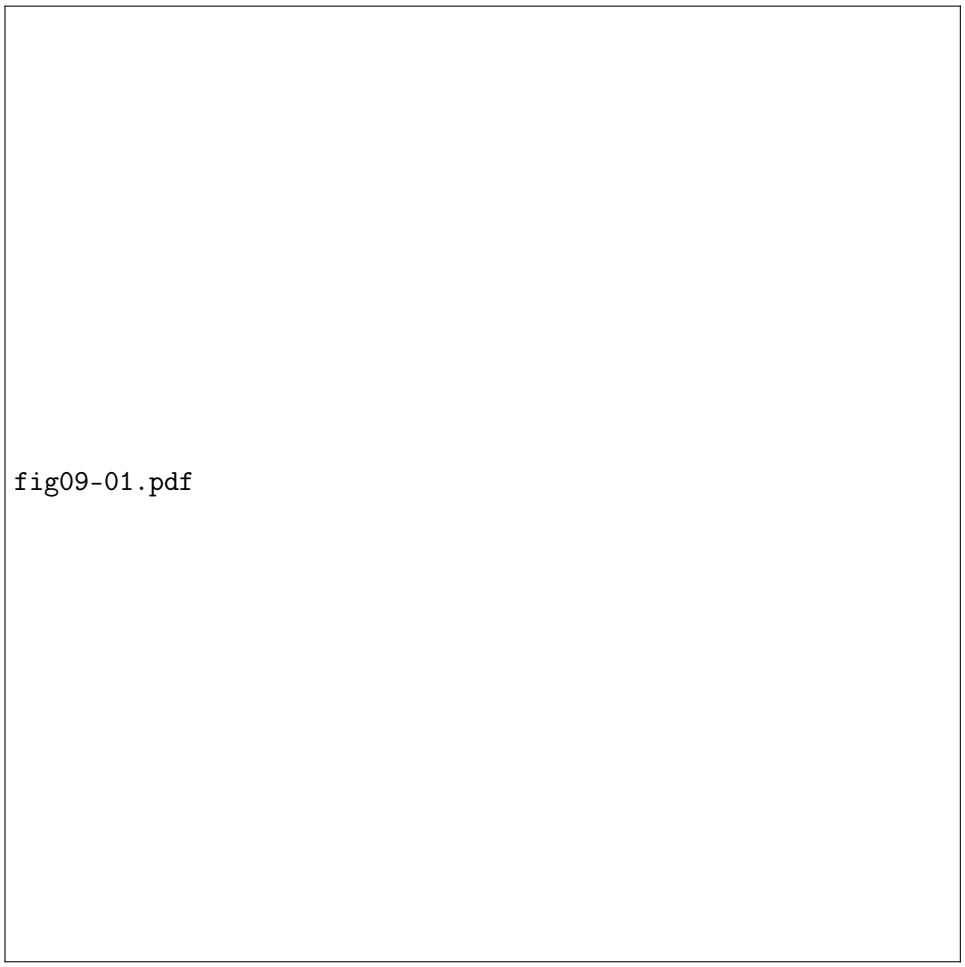

Figure 3: Vibration modes along a guitar string of length one metre, computed numerically using the Jacobi eigenvalue method and the finite difference method, with the first central approximation to the second derivative.