Lecture 11: Eigensystems and empirical orthogonal functions

Recap

We've been looking at one flavor of minimization problem, where we fit data to a model. Now we're going to shift gears to a different flavor of problem, in which we don't assume a prior model but instead look for compact representations of data.

Separately, I'll provide a review of linear algebra as a pre-recorded lecture that you can catch up on in your own time.

Projection

Let's start by discussing the concept of projection. When I correlate two data sets, I'm projecting one into the other (with a normalization). Formally, the regression coefficient tells me the projection. For example, to project $\mathbf{y}$ into $\mathbf{x}$, we compute

$$\hat{y} = \alpha x \tag{1}$$

with

$$\alpha = \frac{\langle xy \rangle}{\langle xx \rangle} \tag{2}$$

which is just the least-squares solution for a very simplified least-squares problem.

We can project $\mathbf{y}$ into multiple functions. For example, if I least-squares fit data using an $N \times M$ matrix $\mathbf{G}$, with orthogonal columns, then each fitted coefficient will tell me the projection into the relevant column of $\mathbf{G}$. (If the columns of $\mathbf{G}$ aren't orthogonal, then our fit will have to partition between them.) We can write, for example

$$y_n = \mathbf{w}^T \mathbf{x}_n, \tag{3}$$

where $\mathbf{w}$ is a vector of coefficients, and $\mathbf{x}_n$ consists of the nth values of multiple vectors $\mathbf{x}$, so that $\mathbf{x}_n = [x_{n1}, x_{n2}]$. In matrix form, this becomes

$$\mathbf{y} = \mathbf{w}^T \mathbf{X}, \tag{4}$$

where $\mathbf{X}$ consists of column vectors of x_{ni}.

The Fourier transform is a classic example of projecting data into orthogonal functions. We taken N data points and project them into N sines and cosines ($N/2$ sines and $N/2$ cosines). In the Fourier transform, we turn N pieces of information in the time/space domain into N with N pieces of information in the frequency/wavenumber domain. If the data set is complicated, the Fourier transform might look just as complicated as the original data set. But use the projection into Fourier space often hoping that it will provide us with a more compact representation of our information—something that is easier to understand.

Now we're going to look at cases where we haven't decided a priori what function to project into. We're simply going to look for the most compact representation of our data set. We did something like this earlier, when we looked for a variance ellipse that would allow us to rotate observations into orthogonal coordinates.

Eigensystems

Let's start by thinking about eigensystems. An eigensystem is defined by the equation

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \tag{5}$$

where $\mathbf{A}$ is a square matrix, $\mathbf{x}$ is a vector, and λ is a scalar. In other words, the transformation $\mathbf{A}\mathbf{x}$ results in a simple scaling of $\mathbf{x}$. Given a **normal matrix A** (a class of matrix that includes symmetric and orthogonal matrices), we can always find a set of λ's (eigenvalues) and a corresponding set of $\mathbf{x}$'s (eigenvectors).

The eigenvectors are equivalent to modes of physical systems. Consider the transverse oscillations of beads on a string (Figure 1). The two beads have mass m, and are separated by flexible strings of length l when at equilibrium. Suppose displacements x_n of the beads are so small that the tension T in the strings can be taken to be constant. The angle of each string to the horizontal is θ_n as illustrated in the figure. The equation of motion for the displacement x_1 of the first bead is

$$m\frac{d^2x_1}{dt^2} = -T\sin\theta_1 + T\sin\theta_2. \tag{6}$$

Under the assumption that the displacements are small, $\sin\theta_n$ may be approximated as $\tan\theta_n$, so

$$m\frac{d^2x_1}{dt^2} = -T\frac{x_1}{l} + T\frac{x_2 - x_1}{l}. \tag{7}$$

Rearranging produces

$$m\frac{d^2x_1}{dt^2} = \frac{T}{l}\left(x_2 - 2x_1\right). \tag{8}$$

Similarly, the equation of motion for the second bead is

$$m\frac{d^2x_2}{dt^2} = \frac{T}{l}\left(x_1 - 2x_2\right). \tag{9}$$

Proceed by assuming a solution of the form

$$x_n = X_n e^{i\omega t} \tag{10}$$

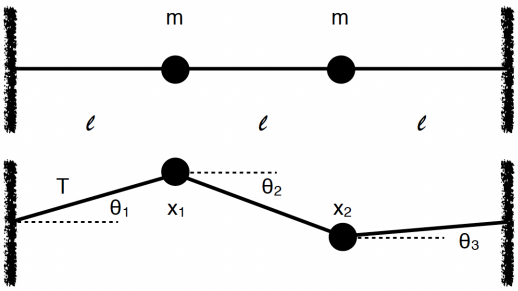

Figure 1: Beads on a string at equilibrium (top) and displaced (bottom).

Then the pair of equations to be solved are

$$(2-\lambda)X_1 - X_2 = 0 \tag{11}$$

$$-X_1 + (2-\lambda)X_2 = 0 \tag{12}$$

where $\lambda = \omega^2 ml/T$ are eigenvalues. For a nontrivial solution to this problem, we must have

$$\det \begin{bmatrix} 2-\lambda & -1 \\ -1 & 2-\lambda \end{bmatrix} = 0 \tag{13}$$

So

$$(2-\lambda)^2 - 1 = 0 \tag{14}$$

which has the two solutions: $\lambda = 1, 3$. For $\lambda = 1$,

$$X_1 - X_2 = 0 \tag{15}$$

which says simply that $X_1 = X_2$. Writing this solution as a normalized vector

$$\mathbf{x} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{16}$$

which is the eigenvector for the eigenvalue $\lambda = 1$. For $\lambda = 3$,

$$-X_1 - X_2 = 0 \tag{17}$$

with the normalized solution

$$\mathbf{x} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{18}$$

The two solutions, commonly referred to as modes, are shown in Figure 2.

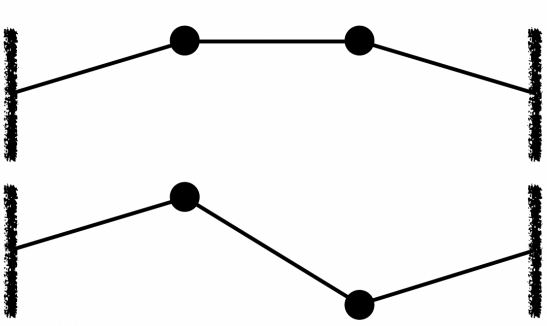

Figure 2: Modes of a two-beaded string. Mode 1 (top) has both beads moving in phase, and mode 2 (bottom) has the beads out of phase.

Given an initial specification of bead positions in terms of modes we can predict the evolution of the system. While equations of motion of the beads (8-9) are coupled, the equations for the modes are uncoupled. That is, the modes evolve independently of each other, and the evolution of the system is a linear combination of the two modes.

Note that the equation we solve was in the form (5) with

$$\mathbf{A} = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix} \tag{19}$$

So what we were doing in solving the the problem of beads on a string, was precisely the solution of an eigensystem. In linear vector space language, the modes are the most convenient coordinates. The eigensystem can be written

$$\mathbf{AP} = \mathbf{PD} \tag{20}$$

where $\mathbf{D}$ is diagonal with the eigenvalues along the diagonal, and the columns of the orthogonal matrix $\mathbf{P}$ are the eigenvectors.

A few standard relations are

$$\mathbf{D} = \mathbf{P}^T\mathbf{AP} \tag{21}$$
$$\mathbf{A} = \mathbf{PDP}^T \tag{22}$$
$$\mathbf{A}^{-1} = \mathbf{PD}^{-1}\mathbf{P}^T \tag{23}$$

Given the eigenvalue decomposition of a matrix, (23) gives an easy way of determining invertibility simply by determining whether any of the eigenvalues are zero. The ratio of the smallest to largest eigenvalue, referred to as the condition number, is an indication of the stability of the inversion to numerical error.

The *condition number* of the matrix $\mathbf{A}$ is the ratio of the largest to the smallest eigenvalue and is an indication of the stability of the inversion to numerical error.

Principal component analysis

In a more generic sense, if we want to find a compact coordinate system to represent our data, we'll want to rotate to a coordinate system that maximizes variance, using as small a number of dimensions as possible. Assume that we demean our data:

$$\sigma_y^2 = \frac{1}{N}\sum_{n=1}^{N} y_n^2 \tag{24}$$
$$= \frac{1}{N}\sum_{n=1}^{N} (\mathbf{w}^T\mathbf{x}_n)^2 \tag{25}$$
$$= \frac{1}{N}\sum_{n=1}^{N} \mathbf{w}^T\mathbf{x}_n\mathbf{x}_n^T\mathbf{w} \tag{26}$$
$$= \mathbf{w}^T\left(\frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n\mathbf{x}_n^T\right)\mathbf{w} \tag{27}$$
$$= \mathbf{w}^T\mathbf{C}_{xx}\mathbf{w}, \tag{28}$$

where $\mathbf{C}_{xx}$ is the covariance matrix of $\mathbf{x}$.

How do we find $\mathbf{w}$? We want to maximize the size of σ^2, subject to a constraint that $\mathbf{w}^T\mathbf{w} = 1$, so we can define a cost function:

$$L = \mathbf{w}^T\mathbf{C}_{xx}\mathbf{w} - 2\lambda(\mathbf{w}^T\mathbf{w} - 1). \tag{29}$$

Taking the derivative with respect to $\mathbf{w}$ yields

$$\frac{\partial L}{\partial \mathbf{w}} = 2\mathbf{C}_{xx}\mathbf{w} - 2\lambda\mathbf{w} = 0, \tag{30}$$

so

$$\mathbf{C}\mathbf{w} = \lambda\mathbf{w}, \tag{31}$$

meaning that $\mathbf{w}$ consists of the eigenvectors of the covariance matrix. So if we compute the covariance matrix of our data, we can find eigenvalues and eigenvectors and determine patterns of variability. But this is potentially more work than we need to do.....

Assessing dominate modes of variability

An empirical orthogonal mode decomposition provides us with a framework for assessing the dominant patterns of variability in any data set, provided that we can represent the data in a matrix form—typically with time in one matrix dimension and space in the other dimension. In class, we looked at examples of checkerboard patterns to gain insight about compact representation of data.
Example 1: What will the EOF decomposition look like for a checkerboard pattern, as shown in Figure 3?

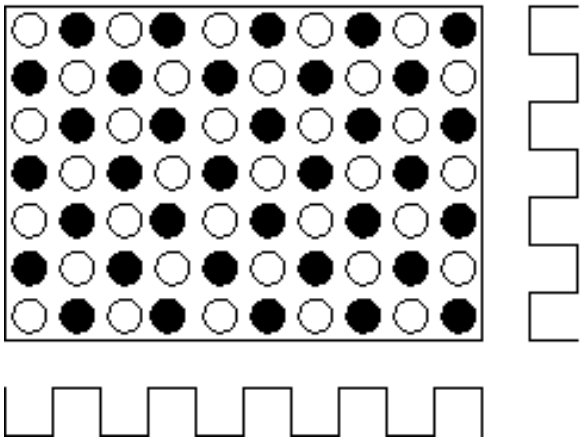

Figure 3: Checkerboard pattern. You could imagine this as a data set consising of alternating values of +1 and -1.

A checkerboard has clear alternating patterns. Every row is easily represented by as either +1 or -1 times the other rows. Such as system has one non-zero eigenvalue, one meaningful vector $\mathbf{u}_1$, and one meaningful vector $\mathbf{v}_1$.
Example 2: What happens when the pattern is more complicated, as in Figure 4?

In this case, although the patterns are not predictability repeated, each row can be represented as either +1 or -1 times the other rows, and each column is +1 or -1 times the other columns. Again the system has one non-zero eigenvalues, one meaningful vector $\mathbf{u}_1$, and one meaningful vector $\mathbf{v}_1$.
Example 3: What happens for a propagating signal, as in Figure 5?

In this case, the rows and columns are not all identical. There are effectively two types of rows and two types of columns, so 2 non-zero eigenvalues. This is our reminder that propagating patterns necessarily are represented by an eigenvalue decomposition as paired orthogonal patterns (a sine mode and a cosine mode).
Example 4: What happens for a a highly unstructured signal, as in Figure 6?

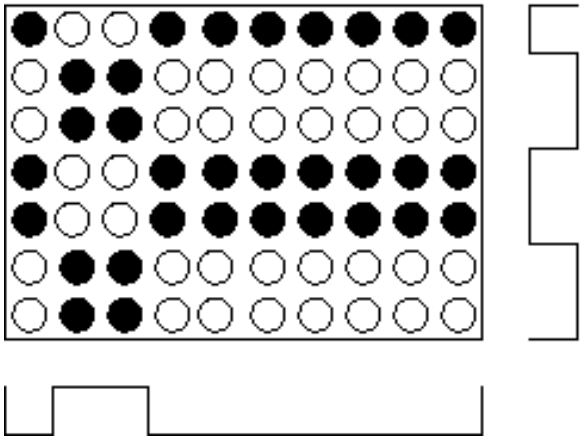

Figure 4: Structured pattern of variability.

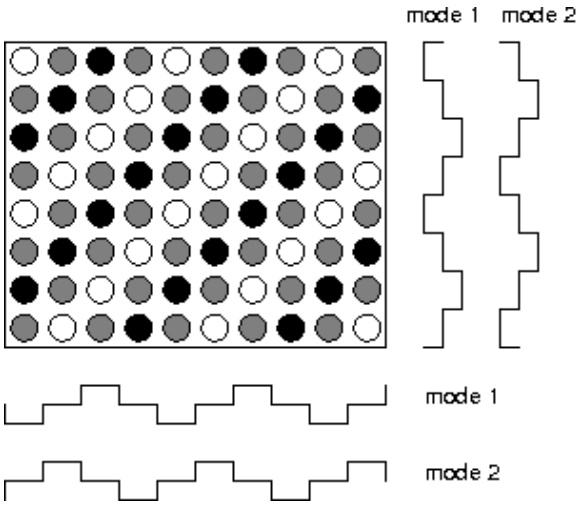

Figure 5: Propagating signal. You can imagine this to consist of values of +1, 0, and -1, depending on color.

In this case, rows and columns are not trivially represented as linear combinations of each other, and the singular value decomposition will have a high rank (and little compactness in representing observed variability).

A simple example: One mode

To put this to work, let's consider a simplified example, of a data set comprising one pattern in u and one pattern in v:

```
utrue=randn(1,10);
vtrue=randn(20,1);
```

or

```
import numpy as np
import scipy
```

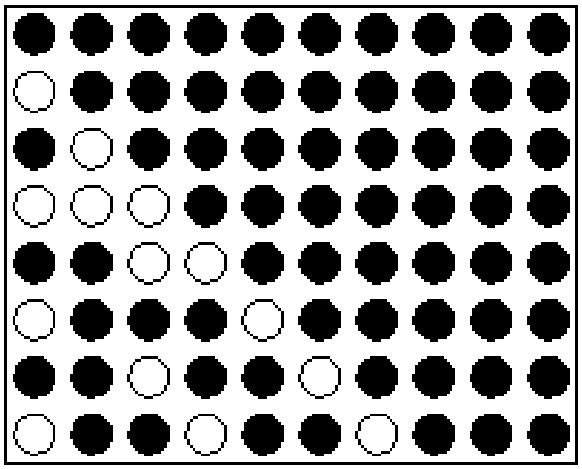

Figure 6: Unstructured signal.

```
import xarray as xr
import cmocean as cmo
import matplotlib.pyplot as plt
from numpy import linalg as LA

utrue=np.random.normal(size=[1,10])
vtrue=np.random.normal(size=[20,1])
```

We multiply these to form a full matrix of variability, and then add noise:

```
data_pure=10*vtrue*utrue
data=data_pure + randn(20,10);
```

or

```
data_pure=10*np.matmul(vtrue,utrue)
data=data_pure+np.random.normal(size=[20,10])
```

We can visualize these two patterns, and you'll see that we haven't added a lot of noise:

```
subplot(1,2,1)
imagesc(data_pure)
subplot(1,2,2)
imagesc(data)
```

or

```
plt.subplot(121)
plt.pcolormesh(data_pure)
plt.subplot(122)
plt.pcolormesh(data)
```

If we want to find eigenvalues we need a square symmetric matrix, so we compute a covariance and then find its eigenvalues:

```
cxx=data'*data;

[V,D]=eig(cxx);
```

```
cxx=np.matmul(data.T,data)
D,V=LA.eigh(cxx)
```

The eigenvalues are sorted in Matlab from smallest to largest:

```
plot(diag(D))
```

and "eigh" does the same in python:

```
plt.plot(D)
```

Then we can plot our original data and the corresponding eigenvector: First for "utrue":

```
plot(1:10,V(:,10),1:10,data(5,:),1:10,utrue)
```

and in python

```
plt.plot(V[:,-1],label='eigenmode')
plt.plot(utrue.flatten(),label='original data')
plt.plot(data[5,:]/10,label='sample data row, arbitrary scaling')
plt.legend()
```

and then for "vtrue", which is reconstructed by projecting the data onto the eigenvectors

```
1:20,data*V(:,10), 1:20, data(:,5)*-5,1:20,vtrue)
```

and in python

```
plt.plot(np.matmul(data,V[:,-1])/np.sqrt(D[-1]),label='reconstructed eigenmode')
plt.plot(vtrue.flatten()/2,label='original data, scaled')
plt.plot(data[:,4]/10,label='sample data column, arbitrary scaling')
plt.legend()
```

In both these cases, the scaling is arbitrary, but we can see that the shape is reconstructed (taking the noise into account).