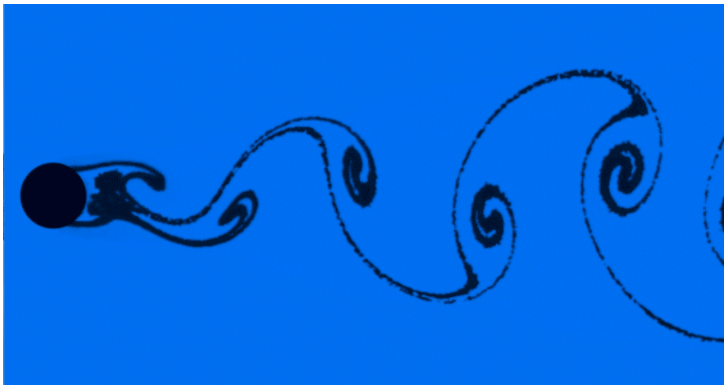
Lecture 14
Principal Component Analysis and Proper Orthogonal Decomposition

Jason J. Bramburger

In this lecture were are going to discuss Proper Orthogonal Decomposition (POD). This is not a new method to us though. That is, it is exactly the same as PCA (and therefore just an application of SVD). If you try to type POD into wikipedia, you get redirected to PCA. So why the new name? POD came out of fluid dynamics, where it remains in use to this day. You can find it in other areas of math too, but POD remains primarily used in fluid dynamics.



Use in Fluids

Let's say you have some sort of fluid flowing around a cylinder. Then, for each point in space and time we can associated a velocity to the fluid that exists for every point (x, y, z) in space and at each time:

$$\begin{bmatrix} u(x,y,z,t) \\ v(x,y,z,t) \\ w(x,y,z,t) \end{bmatrix}.$$

In practice, we can only measure at finitely many locations, but if we want to capture all the behaviour we are going to require A LOT of data (millions or billions of points). As you can guess, it would be helpful if we could reduce the dimensionality of our gathered data. This results in looking for **coherent structures**, which are patterns that just evolve over time.

Function Expansions

For the purposes of this lecture, we will consider only one spatial dimension to keep things simple. So, let's consider a function, $f(x,t)$, of both space (1D) and time. We would like to expand this function into the sum of some basis functions with time-dependent coefficients:

$$f(x,t) = \sum_{n=1}^{\infty} a_n(t)\phi_n(x).$$

We can then approximate the function by truncating to a finite number of terms in the series:

$$f(x,t) \approx \sum_{n=1}^{N} a_n(t)\phi_n(x).$$

You've seen this before in a few different contexts. For example, take $\phi_n(x) = x^n$ to arrive at

$$f(x,t) = \sum_{n=1}^{\infty} a_n(t)x^n$$

the Taylor expansion of f in x about $x = 0$ with time-dependent coefficients. Note that if f is independent of time t then we just have the usual Taylor series. Similarly, if $\phi_n(x) = \mathrm{e}^{inx}$ we get a Fourier series expansion in space. Again, we would have time-dependent coefficients. You could do the same thing with $\phi_n(x)$ as a wavelet.

If we know the spatial basis functions $\phi_n(x)$, how do we find the coefficients $a_n(t)$? Let's work through how we found them in the case of Fourier series (see Lecture 1 for full details). Writing

$$f(x,t) = \frac{a_0(t)}{2} + \sum_{k=1}^{\infty}[a_k(t)\cos(kt) + b_k(t)\sin(kt)]$$

with $x \in [-\pi, \pi]$, we saw that we could isolate a single cosine or sine term in the sum by multiplying $f(x,t)$ by that term and integrating (in x) over the spatial domain. This gives, for example,

$$a_k(t) = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x,t)\cos(kx)\mathrm{d}x.$$

Recall that the reason we could do this trick was because of the orthogonality of the $\cos(kt)$ and $\sin(kt)$, i.e.

$$\int_{-\pi}^{\pi} \sin(nx)\cos(mx)\mathrm{d}x = 0, \quad \forall n, m$$
$$\int_{-\pi}^{\pi} \cos(nx)\cos(mx)\mathrm{d}x = \begin{cases} 0, & n \neq m \\ \pi, & n = m \end{cases}.$$
$$\int_{-\pi}^{\pi} \sin(nx)\sin(mx)\mathrm{d}x = \begin{cases} 0, & n \neq m \\ \pi, & n = m \end{cases}$$

It is worth noting that the $1/\pi$ term simply comes from the fact that we didn't normalize the sines and cosines to have integral with itself equal to 1. If we had done so, the formula would not have that constant.

In general, we want a set of functions $\{\phi_n(x)\}_{n=0}^{\infty}$ for which you can expand $f(x,t)$ in terms of these functions and

$$\int_a^b \phi_n(x)\phi_m(x)\mathrm{d}x = \begin{cases} 0, & n \neq m \\ 1, & n = m \end{cases}$$

where we are assuming $x \in [a, b]$ for generality. Such a set of functions is called an *orthonormal basis*. The *ortho-* part comes from orthogonal, meaning the above integral proprety. The *-normal* part means that the integral of the square of the functions is 1 (case $n = m$ above). If the $\phi_n(x)$ form an the orthonormal basis we can obtain the coefficients in the same way we got the Fourier coefficients, which gives

$$a_n(t) = \int_a^b f(x,t)\phi_n(x)\mathrm{d}x.$$

This is what makes orthonormal bases great: You can uniquely recover the coefficients in the series. This wouldn't necessarily be the case if we didn't have the orthogonality condition on the $\phi_n(x)$.

Choosing Basis Functions

The next question we should ask ourselves is: What basis functions should I be using? You can expand a function in a lot of different ways - infinitely many in fact. Sometimes it is useful to choose basis functions that have an intuitive meaning. This is what we did with Fourier basis functions representing frequencies. Another choice would be to choose basis functions in such a way that we get the best approximation of the function with the fewest number of terms in the truncation. So, if we keep just the first term in the infinite sum, we get the best possible one-term approximation of the function. If we keep two terms, we get the best possible two-term approximation. And so on. The optimal choice of basis functions to achieve this is are called **proper orthogonal modes**. Expanding a function in these proper orthogonal modes leads to the **proper orthogonal decomposition (POD)**.

Relation to the SVD

The above should sound familiar to you! We are basically describing the low-rank approximations from the SVD. In fact, this is exactly how you find POD modes in practice. Start by discretizing in space and time:

$$x_1, x_2, \ldots, x_m$$
$$t_1, t_2, \ldots, t_n.$$

These discretizations lead to an $m \times n$ matrix with the values of the function at these discrete values in space and time. The rows correspond to a position in space and the columns correspond to a point in time:

$$\begin{bmatrix} f(x_1,t_1) & f(x_1,t_2) & \cdots & f(x_1,t_n) \\ f(x_2,t_1) & f(x_2,t_2) & \cdots & f(x_2,t_n) \\ \vdots & \vdots & \ddots & \vdots \\ f(x_m,t_1) & f(x_m,t_2) & \cdots & f(x_m,t_n) \end{bmatrix}.$$

The columns are sometimes called **snapshots**. Now that we have a matrix, we can apply the SVD. The left singular vectors will provide basis functions and the right singular vectors will characterize how the coefficients change over time.

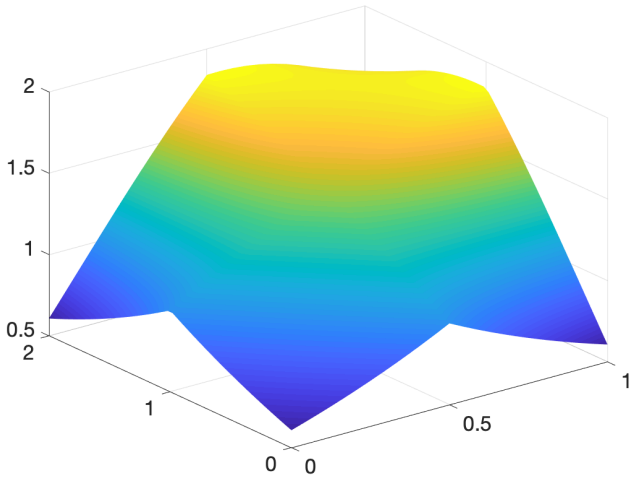
Example 1

To see how this works in practice, let's define the function

$$f(x,t) = \mathrm{e}^{-|(x-0.5)(t-1)|} + \sin(xt), \quad x \in [0,1], \quad t \in [0,2].$$

Start by discretizing in space and time and then plot it in MATLAB.

```
x = linspace(0,1,25);
t = linspace(0,2,50);
[T, X] = meshgrid(t,x);
f = exp(-abs((X - 0.5).*(T - 1))) + sin(X.*T);
figure(1)
surf(X,T,f)
shading interp
set(gca,'fontsize',16)
```

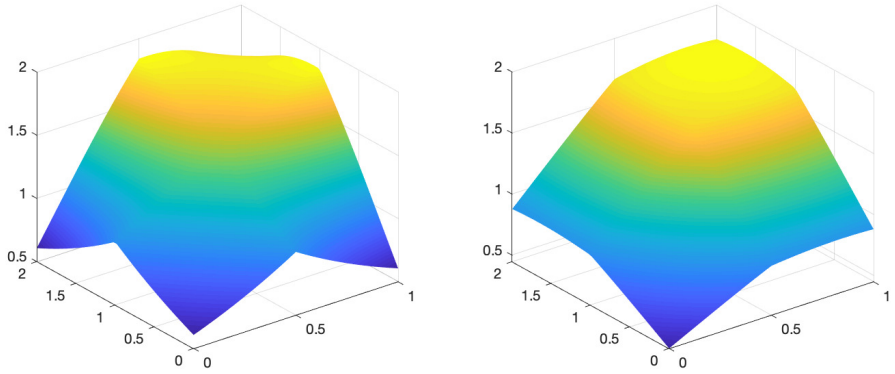


Let's use the SVD to compute the rank-1 approximation:

```
[U,S,V] = svd(f,'econ');
```

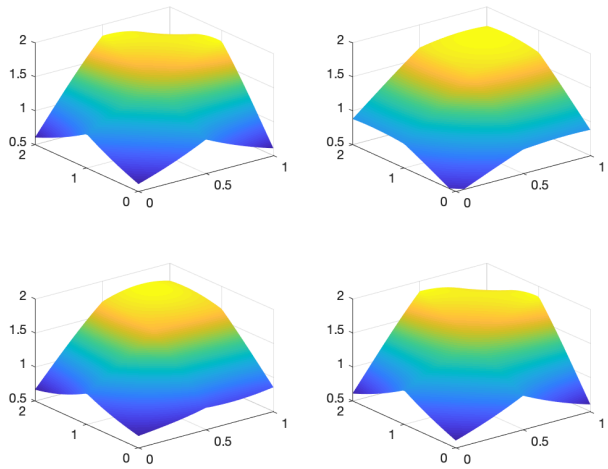
```
f_rank1 = U(:,1)*S(1,1)*V(:,1)';

figure(2)
subplot(1,2,1)
surf(X,T,f)
shading interp
subplot(1,2,2)
surf(X,T,f_rank1)
shading interp
```



Let's compare the rank-1, rank-2, and rank-3 approximations:

```
figure(3)
subplot(2,2,1)
surf(X,T,f), shading interp
for j = 1:3
    ff = U(:,1:j)*S(1:j,1:j)*V(:,1:j)';
    subplot(2,2,j+1)
    surf(X,T,ff), shading interp
    set(gca,'Zlim',[0.5 2])
end
```



It should be clear that as you use more POD modes the approximation gets better. The question is: how much better? That is, how do we quantify how much better the approximation gets by using more modes. We will measure this by determining how much **energy** from the full system is contained in each mode. Assuming

the full matrix is rank r, we measure the energy contained in the rank-N approximation by:

$$\text{energy}_N = \frac{\sigma_1^2 + \sigma_2^2 + \ldots \sigma_N^2}{\sigma_1^2 + \sigma_2^2 + \ldots \sigma_r^2} = \frac{\|X_N\|_F^2}{\|X\|_F^2}.$$

You may notice that this formula for energy is different from the one given in the textbook. In the textbook, they just sum the singular values without squaring them. We will use the version presented above for two reasons. First, it has a connection with the Frobenius norm, as you can see above. Second, we will use the word energy a lot while talking about the SVD, even when we are taking the SVD of an image. Thinking about the 'energy' of an image is weird, but when you're approximating something physical like fluid flow, you can really talk about the (kinetic) energy based on the velocities. The above formula actually corresponds to the physical energy for these types of problems.
Let's calculate the energies for our low-rank approximations:

```
sig = diag(S);
energy1 = sig(1)^2/sum(sig.^2)
energy2 = sum(sig(1:2).^2)/sum(sig.^2)
energy3 = sum(sig(1:3).^2)/sum(sig.^2)
```

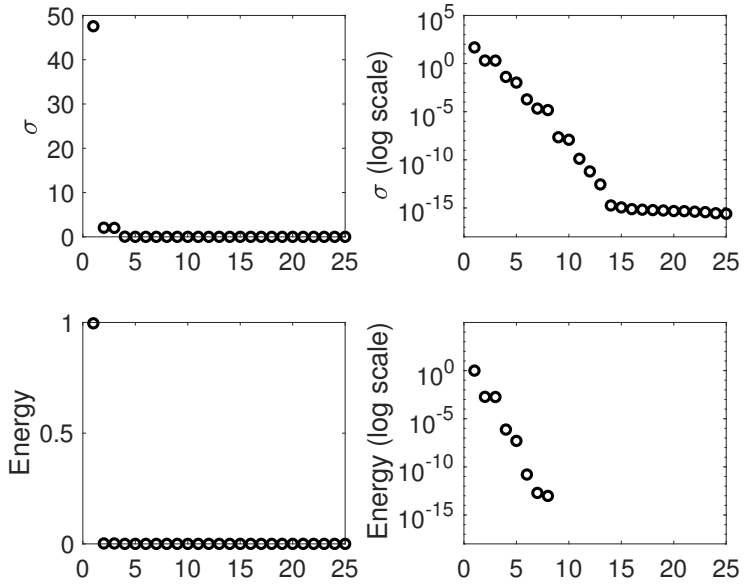
The values of the energy are 0.9963, 0.9982, and 1.000 for the rank 1 through 3 approximations, respectively. Hence, the rank-1 approximation contains more than 99% of the energy, while the first three modes capture all of the energy (to rounding error). We can create a plot that shows the singular values and the energies captured by each mode.

```
figure(4)
subplot(2,2,1)
plot(sig,'ko','Linewidth',2)
axis([0 25 0 50])
ylabel('\sigma')
set(gca,'Fontsize',16,'Xtick',0:5:25)
subplot(2,2,2)
semilogy(sig,'ko','Linewidth',2)
axis([0 25 10^(-18) 10^5])
ylabel('\sigma (log scale)')
set(gca,'Fontsize',16,'Xtick',0:5:25,'Ytick',logspace(-15,5,5))
subplot(2,2,3)
plot(sig.^2/sum(sig.^2),'ko','Linewidth',2)
axis([0 25 0 1])
ylabel('Energy')
set(gca,'Fontsize',16,'Xtick',0:5:25)
subplot(2,2,4)
semilogy(sig.^2/sum(sig.^2),'ko','Linewidth',2)
axis([0 25 10^(-18) 10^5])
ylabel('Energy (log scale)')
set(gca,'Fontsize',16,'Xtick',0:5:25,'Ytick',logspace(-15,0,4))
```

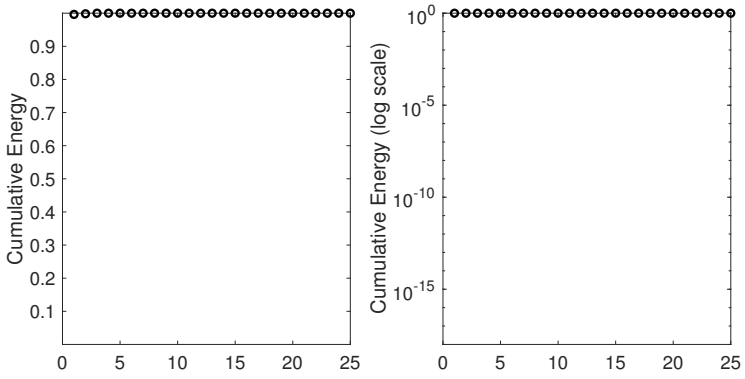
Notice how much larger the first singular value is to the rest of the singular values. The second and third singular values are also large in comparison to the rest. Since the difference in scales is so pronounced, it is better to plot on a log scale. We we plot energy, the squaring of the singular values means that differences become more stark. Plots like this can be used to determine what the appropriate value for N in your low-rank approximation should be. For example, if 4 singular values are much larger than the rest, you can use a 4 mode approximation.

Let's look at the cumulative energy in the first N modes:

```
figure(5)
subplot(1,2,1)
plot(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',2)
axis([0 25 10^-(18) 1])
```



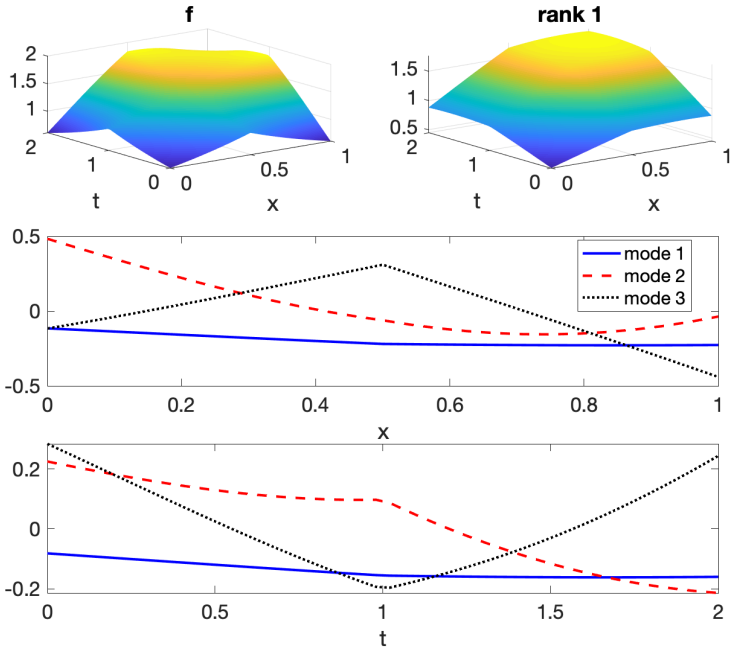

```
ylabel('Cumulative Energy')
set(gca,'Fontsize',16,'Xtick',0:5:25)
subplot(1,2,2)
semilogy(cumsum(sig.^2)/sum(sig.^2),'ko','Linewidth',2)
axis([0 25 10^-(18) 1])
ylabel('Cumulative Energy (log scale)')
set(gca,'Fontsize',16,'Xtick',0:5:25,'Ytick',logspace(-15,0,4))
```



Recall from above that the left and right singular vectors have a lot of meaning regarding the expansion of the function in an orthonormal basis. That is, we can plot the columns of U to see the modes themselves, while plotting the columns of V to see how these modes evolve in time.

```
figure(6)
subplot(3,2,1)
surf(X,T,f), shading interp
xlabel('x')
ylabel('t')
title('f','Fontsize',16)
set(gca,'Fontsize',16)
subplot(3,2,2)
surf(X,T,f_rank1), shading interp
xlabel('x')
ylabel('t')
title('rank 1','Fontsize',16)
```

```
set(gca,'Fontsize',16)
subplot(3,1,2)
plot(x,U(:,1),'b',x,U(:,2),'--r',x,U(:,3),':k','Linewidth',2)
xlabel('x')
set(gca,'Fontsize',16)
legend('mode 1','mode 2','mode 3','Location','best')
subplot(3,1,3)
plot(t,V(:,1),'b',t,V(:,2),'--r',t,V(:,3),':k','Linewidth',2)
xlabel('t')
set(gca,'Fontsize',16)
```



Example 2

Now we will consider the function

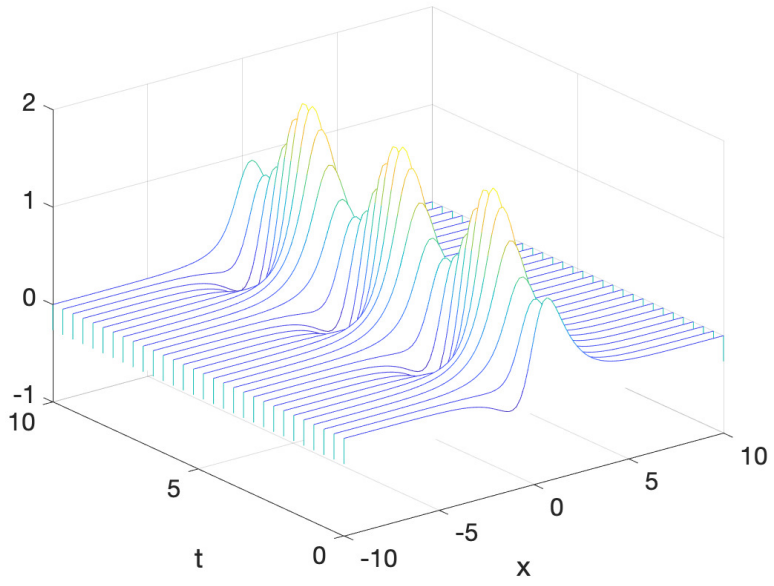
$$f(x,t) = [1 - 0.5\cos(2t)]\text{sech}(x) + [1 - 0.5\sin(2t)]\text{sech}(x)\tanh(x).$$

We are going to use the x values as rows and the t values as columns so that we can use a waterfall plot to represent the data. Don't worry though, we will work with the transpose for the SVD so everything will be the same.

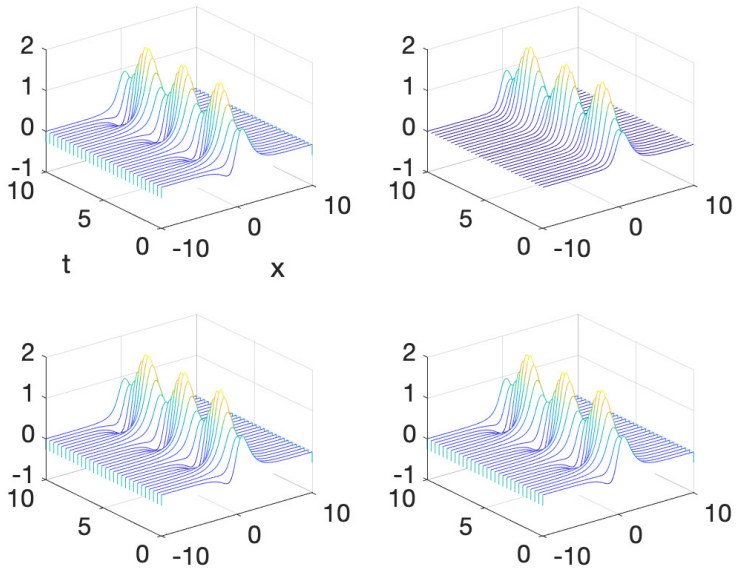
```
clear all; close all; clc

x = linspace(-10,10,100);
t = linspace(0,10,30);
[X,T] = meshgrid(x,t);
f = sech(X).*(1-0.5*cos(2*T))+(sech(X).*tanh(X)).*(1-0.5*sin(2*T));
figure(7)
waterfall(X,T,f)
xlabel('x')
ylabel('t')
set(gca,'Fontsize',16,'Zlim',[-1,2])
```

```
[U,S,V] = svd(f','econ');
figure(8)
subplot(2,2,1)
waterfall(X,T,f)
xlabel('x')
```



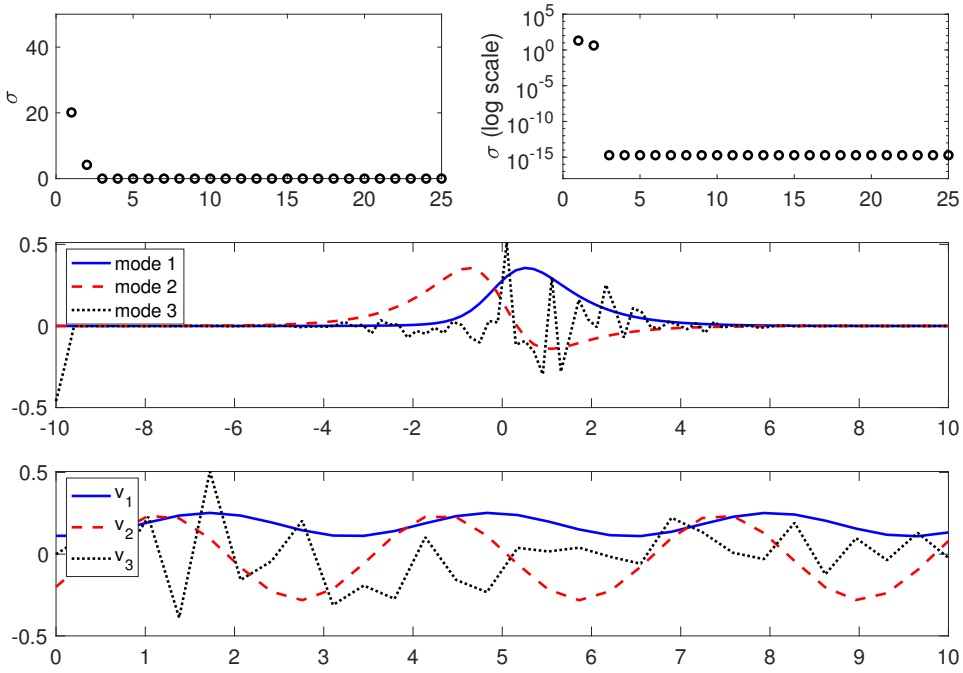

```
ylabel('t')
set(gca,'Fontsize',16,'Zlim',[-1,2])
for j=1:3
    ff = U(:,1:j)*S(1:j,1:j)*V(:,1:j)';
    subplot(2,2,j+1)
    waterfall(X,T,ff')
    set(gca,'Fontsize',16,'Zlim',[-1,2])
end
```



Let's conclude this example by plotting the singular values and the POD modes:

```
sig = diag(S);
figure(9)
subplot(3,2,1)
plot(sig,'ko','Linewidth',2)
axis([0 25 0 50])
```

```
ylabel('\sigma')
set(gca,'Fontsize',16,'Xtick',0:5:25)
subplot(3,2,2)
semilogy(sig,'ko','Linewidth',2)
axis([0 25 10^-(18) 10^5])
ylabel('\sigma (log scale)')
set(gca,'Fontsize',16,'Xtick',0:5:25,'Ytick',logspace(-15,5,5))
subplot(3,1,2)
plot(x,U(:,1),'b',x,U(:,2),'--r',x,U(:,3),':k','Linewidth',2)
set(gca,'Fontsize',16)
legend('mode 1','mode 2','mode 3','Location','northwest')
subplot(3,1,3)
plot(t,V(:,1),'b',t,V(:,2),'--r',t,V(:,3),':k','Linewidth',2)
set(gca,'Fontsize',16)
legend('v_1','v_2','v_3','Location','northwest')
```



We can see that only two POD modes are important. Even on the log scale, all of the rest of the singular values are essentially zero. Practically speaking, we could chalk this up to rounding error. This explains why the first two POD modes are nice smooth functions while the third one looks like noise. The u vectors (left singular vectors) are the spatial basis functions for POD, while the v vectors (right singular vectors) describe the time evolution of those basis vectors.

A Note on the Relation to PCA

Recall that with PCA we subtracted by the mean for each column, but here we didn't do that. Beyond just subtracting out the mean, it is typical with PCA to scale each variable so that its variance is 1 before doing the SVD. Why? This is to deal with data on different scales. Take for example our weight and height data. If we measure weight in megagrams and height in millimetres, we would have small values for weight but large values for height. Similarly, if we measure weight in grams and height in kilometres we would have large values for weight and small values for height. If we just subtract the mean, it will help, but the spread is still determined by the scales. Since the units we choose to measure in are arbitrary, it makes sense to scale the variance because that would scale out the units.

So, why didn't we subtract off the mean and scale the variance to 1 with our data above? First, if we made the variance 1 in Example 2 it would mean that we are blowing up the values near the edges (where things are

essentially zero) to matter as much as the middle. Second, if we subtract off the mean, we lose track of which values are large compared to others across x values. This might be okay in some cases where we are just interested in how much things vary in different locations, but for the types of problems that POD is used for, we usually want to preserve what is large.

To be frank, the main reason we didn't worry about this kind of scaling is because all of the measurements were made with the same units. So, we didn't have to worry about the same thing as the weight/height example. If you are comparing things with incomparable units, you should do scaling. But overall, you have to think about the context. I realize this is frustrating to hear since it doesn't translate into a concrete set of rules to follow, but unfortunately that's how data science works. You have to have a deep understanding of the particular problem and its needs. The way you typically have to approach these things is through trial and error by going back over things if they don't work right and correcting appropriately.