Lecture 11:

Recap

Last time we looked at methods for computing spectra by filtering in the frequency domain, and we looked at the use of the covariance to compute spectra. We had a few loose ends on the covariance approach, so we'll revisit that now.

Autocovariance in discrete form.

Last time we went through the representation of the autocovariance using an integral form. Let's rewrite this in terms of the discrete Fourier transform. In this case, the mean of our data is:

$$\langle x \rangle = \frac{1}{2T} \int_{-T}^{T} x(t) e^{i0} \, dt = a_0. \tag{1}$$

and the variance is

$$\langle x * x \rangle = \frac{1}{2T} \int_{-T}^{T} x^*(t) x(t) \, dt - |a_0|^2. \tag{2}$$

We use the complex conjugate here, just in case $x(t)$ is represented as a complex number, since this will give us the sum of the squares. Notice that we've remembered to subtract out the mean (our frequency zero Fourier coefficient).

In similar notation, we can write the covariance (for finite record length $2T$) as:

$$R(\tau) = \frac{1}{2T} \int_{-T}^{T} x^*(t) x(t+\tau) \, dt - |a_0|^2. \tag{3}$$

This lets us write out an expression for the variance R in terms of the discrete Fourier coefficients:

$$\begin{aligned} R(\tau) &= \frac{1}{2T} \int_{-T}^{T} \left[\sum_{n=-\infty}^{\infty} a_n^* e^{-i2\pi f_n t} \sum_{m=-\infty}^{\infty} a_m e^{i2\pi f_m (t+\tau)} \right] dt - |a_0|^2 & (4) \\ &= \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} a_n^* a_m e^{+i2\pi f_m \tau} \frac{1}{2T} \int_{-T}^{T} e^{i(-2\pi f_n + 2\pi f_m)t} \, dt - |a_0|^2 & (5) \\ &= \sum_{m=-\infty}^{\infty} |a_m|^2 e^{+i2\pi f_m \tau} - |a_0|^2 & (6) \end{aligned}$$

where we used a Kronecker delta (δ_{nm}) to eliminate the integral with $e^{2\pi i(-f_n + f_m)t}$ except when $n = m$, and we subtracted a_0^2 at the end to match our original definition. In setting this up, recall (from lecture 5) that the Fourier transform uses $e^{-i2\pi ft}$, and the inverse transform uses $e^{+i2\pi ft}$. We're using the inverse transform here (though the signs reversed when we had the complex conjugate. This tells us that the Fourier transform of the autocovariance can be expressed by the squared Fourier coefficients. (So we could avoid the Fourier transform completely and just work with the auto-covariance.)

In this form, Parseval's theorem simply says that

$$\begin{aligned} R(0) &= \frac{1}{2T} \int_{-T}^{T} x^*(t) x(t+0) \, dt - |a_0|^2 & (7) \\ &= \sum_{m=-\infty}^{\infty} |a_m|^2 - |a_0|^2 & (8) \end{aligned}$$

meaning that the variance of x is the sum of magnitudes of the Fourier coefficients.

In the case that we considered last time, with white noise, let's consider the autocovariance:

$$R_{xx}(\tau) = \int_{-\infty}^{\infty} x(t)x(\tau + t)dt = \begin{cases} 0 & \text{for } \tau \neq 0 \\ 1 & \text{for } \tau = 0 \end{cases} \tag{9}$$

This is true, because white noise is uncorrelated except at zero lag.

Alternatively, if I consider red noise, then the noise will be correlated from point to point, and the autocovariance will have a bit of spread. We can test this out:

```
% define red data with autoregressive process
a=randn(10000,1);
b(1)=a(1);
for i=2:length(a)
  b(i)=b(i-1)+a(i);
end

BB=xcorr(b,b)/max(xcorr(b,b));
BB_unbiased=xcorr(b,b,'unbiased')/max(xcorr(b,b,'unbiased'));
plot(-999:999,[BB' BB_unbiased'],'LineWidth',3)
```

Biased vs unbiased estimators.

Notice that we could consider both the 'biased' and the "unbiased" estimator. There are arguments for either choice. The difference depends on how we normalize our discrete autocovariance. In the unbiased case, we're computing

$$R(\tau)_{unbiased} = \frac{1}{N-m} \sum_{n=1}^{N-m} x(t_n)x(t_{n+m}). \tag{10}$$

In the biased case, we change how we normalize:

$$R(\tau)_{biased} = \frac{1}{N} \sum_{n=1}^{N} x(t_n)x(t_{n+m}), \tag{11}$$

which means that as the number of values we consider becomes smaller, we constrain the magnitude of the autocovariance by continuing to divide by N. Emery and Thomson note that the biased estimator acts like a triangle window.

Variance preserving spectra

Last time we said, that one of the virtues of the properly normalized spectrum is that the area under the curve should represent the signal variance within a specific frequency band:

$$\text{variance in a band} = \int_{f-\Delta f/2}^{f+\Delta f/2} |X(f)|^2 \, df \tag{12}$$

This sounds good as a concept, but in log-log space, it's challenging to figure out what the area under the curve really represents.

That might not bother you, but sometimes that visual representation might seem helpful. Consider a spectrum $S_{xx}(f)$ derived from our Fourier amplitudes $X(f)$. In log space, the area under our curves is a pseudo-variance s_*^2:

$$s_*^2 = \int_{f-\Delta f/2}^{f+\Delta f/2} \log(S_{xx}(f))\, df \tag{13}$$

If we want to plot something that is more directly representative of variance, we can try this:

$$s^2 = \int_{f-\Delta f/2}^{f+\Delta f/2} S_{xx}(f)\, df = \int_{f-\Delta f/2}^{f+\Delta f/2} f S_{xx}(f) d[\log(f)] \tag{14}$$

This means that instead of plotting log$S$ vs log(f), we could plot $fS_{xx}(f)$ in linear space vs f in log space. This is especially useful for features that have strong peaks not exactly at the lowest frequency.

Here's an example, using the red noise that we started with earlier:

```
bb=reshape(b,500,10000/500);
fbb=fft(detrend(bb));
amp=abs(fbb(1:251,:)).^2 / 500;
sbb=mean(amp,2);
sbb(2:250)=sbb(2:250)*2;

subplot(2,1,1)
loglog(0:250,sbb,'LineWidth',3)
xlabel('Frequency','FontSize',14)
ylabel('(units^2)/frequency','FontSize',14)

subplot(2,1,2)
semilogx(0:250,(0:250)'.*sbb,'LineWidth',3)
xlabel('Frequency','FontSize',14)
ylabel('(units^2)','FontSize',14)
```

Multi-tapers and spectral peaks

Spectra can come in two flavors. Some have distinct single peaks, associated with tides. Some have large-scale structure associated with the general red structure of the ocean. If we want to find exactly the right peaks, then we can try different strategies to what we use when we want to find the general structure.

When we have narrow peaks, they aren't always easy to differentiate, particularly if our sampling is a bit coarse compared with the signals we'd like to detect. Consider the following case of a sinusoidal cycle that might or might not be well sampled, depending how long our instruments survive:

```
time=1:.5:120;
A=2*cos(2*pi*time/30)+cos(2*pi*time/60);
B=A(1:200);
C=[A(1:200) zeros(1,40)];
```

```
plot(time,A,time(1:200),B,'LineWidth',3)
set(gca,'FontSize',16)
xlabel('time','FontSize',16)
ylabel('amplitude','FontSize',16)

fA=fft(A);
fB=fft(B);
fC=fft(C);

frequency1=(0:120)/120;
frequency2=(0:100)/100;

loglog(frequency1,abs(fA(1:121)).^2,frequency2,abs(fB(1:101)).^2,...
  frequency1,abs(fC(1:121)).^2,'LineWidth',3)
set(gca,'FontSize',16)
xlabel('frequency','FontSize',16)
ylabel('spectral density','FontSize',16)
legend('full record','truncated record','zero padding')
```

When you look at this example, you might conclude that without perfect sampling of full sinusoidal cycles, we'll never find the correct spectral peaks. In essence this is a windowing problem.

If we don't have adequate resolution what are our options?

1. Possibility 1. Pad the short record with zeros to make it as long as we want. Since resolution is $f = 1/(N\Delta t)$. In this case, we'll see the impact of a sinc function bleeding into the frequencies that we'd like to resolve. Clearly this doesn't fully solve our problem.

2. Possibility 2. Obtain a longer record. This will be critical if we really want to resolve our signal.

Even if our record is norminally long enough, we also need to figure out how to optimize our detection of spectral peaks. Last week we looked at the impact of windows, and examples from the Harris (1978) study showed how much impact a good windowing strategy can have in identifying spectral peaks. (For continuous spectra, windowing approaches work well.)

Formally, you'll recall that we can represent our record length problems using a convolution of our data with a finite width filter:

$$\hat{X}(f_n) = \int_{-\infty}^{\infty} X(f_m) W(f_n - f_m)\, df_m, \tag{15}$$

where

$$W(f) = \frac{\sin(2\pi f T)}{2\pi f T} = \text{sinc}\,(2\pi f T) \tag{16}$$

This means that the spectrum is essentially convolved with $W(2\pi f T)^2$. But as we noted earlier, we can switch from a boxcar window to a triangle window or something a bit more Gaussian than that and cut down on the sidelobes in our window to obtain a cleaner spectrum, although we have to widen the central peak of the window in the frequency domain, which means de-emphaizing the beginning and end of the data series.

What if we want to improve our resolution. Consider Rob Pinkel's example of a record equivalent to

$$x = 100\cos(2\pi 20.5/10001)+80\cos(2\pi 30.4/100001)+100\cos(2\pi 40.8/100001)+10\cos(2\pi 50.3/100001) \quad (17)$$

The quality of our spectral estimate will depend on the length of our record. (Why is that? The resolution is the lowest resolved frequency.) So what can we do to improve resolution. One strategy would be to pad our record with zeros to make it as long we want. That buys us something, but it gives us plenty of spectral ringing.

If we want to optimize resolution, we can try a multitaper approach. (See for example Ghil et al, Reviews of Geophysics, 2001). In a multitaper approach, we replace our single window with a set of tapers. The tapers are designed to minimize spectral leakage, and they are referred to as "discrete prolate spheroidal sequences" or "Slepian" tapers (after Slepian, who studied them). Tapers are used like windows—they pre-multiply the data, Fourier transforms are computed, and then the spectrum is computed as a weighted sum of all of the squared Fourier transforms. This effectively averages over an ensemble of windows to minimize variance. This is very effective for extracting narrow peaks that would otherwise be undetectable. Matlab has a multi-taper method package ('pmtm'), but if you really want this to work, you probably want to dig into the guts of the algorithm a bit further. Here's the Matlab example, modified slightly to make a longer record:

```
fs = 1000;                  % Sampling frequency
t = (0:3*fs)/fs;              % One second worth of samples
A = [1 2];                  % Sinusoid amplitudes
f = [150;140];              % Sinusoid frequencies

xn = A*sin(2*pi*f*t) + 0.1*randn(size(t));
pmtm(xn,4,[],fs)
```

This produces an impressive two spectral peaks. Of course this example isn't too tricky. Here's what we get if we take the same data and split them into 6 non-overlapping segments, even with no windowing or detrending:

```
fa=fft(reshape(xn(1:3000),500,6));
semilogy(mean(abs(fa(1:250,:)').^2))
```

These are reassuringly similar results.