Lecture 7:

Reading: Bendat and Piersol, Ch. 8.5.4

Recap

Last time we took another look at Fourier coefficients to verify that the coefficients that we assert should represent our data really do work, and in doing so, we defined the Kronecker delta, δ_{nm}. Then we talked about 3 really key concepts for Fourier transforms:

1. First derivatives in the time domain (or the space domain) can be represented in the frequency (or wavenumber) domain by simple multiplication (e.g. by $i2\pi f$).

2. Convolution, which is what we do when we filter data, can be carried out in the frequency (or wavenumber) domain through simple multiplication. The convolution can be written:

$$x * y(\tau) = \int_{-\infty}^{\infty} x(t)y(\tau - t)\, dt, \tag{1}$$

   and its Fourier transform is:

$$X(f)Y(f) = \int_{-\infty}^{\infty} x * y e^{-i2\pi ft}\, dt. \tag{2}$$

3. Parseval's theorem: Total variance in the time domain equals total variance in the frequency domain

$$\int_{-\infty}^{\infty} x^2(t)dt = \int_{-\infty}^{\infty} |\hat{x}(f)|^2\, df. \tag{3}$$

   It's worth noting that if we worked with $\sigma = 2\pi f$ rather than f, we'd have to normalize by 2π:

$$\int_{-\infty}^{\infty} x^2(t)dt = \frac{1}{2\pi}\int_{-\infty}^{\infty} |X(\sigma)|^2\, d\sigma \tag{4}$$

   In thinking about the time domain vs the frequency domain, one thing to keep in mind is the distinction between integrating over all time (on the left in the above equation) and integrating over all space (on the right). This implies that we're going to need to keep track of our frequency inforamtion carefully. In essence the Fourier coefficients in X (e.g. $|a_m|^2$) do not have the same units as the time domain values in x^2, because x is integrated in time and $|a_m|$ is integrated in frequency. If the total integral of x^2 is equal to the total integral of $|X|^2$, then we're going to need to adjust by factors of δf, and this will influence how we label our axes.

We also considered the terminology for describing spectra as red, white, or blue, and we discussed how we from a plot we can identify the Nyquist frequency (the highest frequency resolved, representing one cycle every 2 points, so it tells us the sampling interval) and the duration of the record (based on the lowest resolved frequency, which is one cycle for the full record.)

Computing spectra

We've covered a lot of territory—we can Fourier transform, we know some of the properties of the Fourier transform, we've looked at Parseval's theorem. Now we need to stop beating around the bush and produce some spectra of our own.

How do we take our data and produce a meaningful measure of the power per unit frequency? Here's a basic approach:

1. First, we know we're going to need to Fourier transform our data, and plot the squared amplitudes. We'll only need to analyze the first $N/2+1$ of the Fourier coefficients, and we'll look at the amplitudes of these values. (The second half of the Fourier coefficients are complex conjugates of the first half of the record and correspond to negative frequencies.) The frequencies corresponding to the first $N/2+1$ coefficients will run from 0 cycles per N points to $N/2$ cycles per N points. So a first step is to compute:

```
a=fft(data)
N=length(data);
amp=abs(a(1:N/2+1).^2; % for even N
amp=abs(a(1:(N+1)/2).^2; % for odd N
```

2. Second, since we're only taking half the record, we've thrown out half the energy in the original data (except at frequency 0), so we'll need to put that back in.

```
amp(2:end-1)=2*amp(2:end-1); % for even N
amp(2:end)=2*amp(2:end); % for odd N
```

3. Having gotten this far, we'd better check that Parseval's theorem is working for us. We'll need to see that the energy in the initial record matches the energy in the Fourier transform. If we don't worry about units, then in Matlab we just need to divide by N to make our normalization work.

```
amp=amp/N;
```

   If we want to make this make sense for the temporal and spatial sampling of our observations we might need to scale our results to reflect real time and frequency units. (Fortunately multiplicative scalings won't influence the shape of our spectra, so we can always delay sorting out the details.)

4. We can plot what we have. Typically we use a loglog axis (or at least semilogy). We'll need to compute our frequencies thoughtfully. So for example if we have measurements every 361 seconds, then we might want to convert to cycles per hour by scaling the frequency appropriately.

```
scale = 1/361 * 3600; % to convert from 1 cycle/2 pts to 0.5 cycles
                      % per 361 s to 0.5 cycles * 3600/361 /hr
frequency=(0:N/2)/N * scale; % for N even
frequency=(0:N/2-1)/N * scale; % for N odd
loglog(frequency,amp)
```

5. Now, we have a problem in that our results are way too noisy. We'll have a hard time distinguishing signal from peak. So clearly we're going to need more realizations. To do this, one common practice is chop our data into multiple segments. As a first step, we can just cut the data into M segments of N/M points each. For example, here's a brute strenght approach:

```
N=length(data); M=6; p=N/M;
data=reshape(data,N/M,M); % this gives us an array with N/M points
          % per column and M columns
b=fft(data);  % this computes the fft for each column
amp_b=abs(b(1:p/2+1,:)).^2; amp_b(2:p/2,:)=2*amp_b(2:p/2,:)/p;
loglog(frequency,mean(amp_b,2))
```

Here you can debate whether you should be plotting the sum (to conserve energy) or the average (to represent a mean spectrum for a data set of length p. In most cases you'll want the average.

Now the critical question? How many degrees of freedom does this record have? Is this M to represent M segments? Maybe you can think of it that way, but by convention, we get one degree of freedom for the real part and one for the imaginary part, so 2 per segment. We'll need this to compute error bars, but let's start by noting that our error bars are not the same as the standard error of the mean. We're computing the sum of M squared quantities, and that's going to depend on something that looks like a χ^2 distribution.

When we compute spectra from segments, clearly there are tradeoffs: if I have N data points total, I can have lots of segments with few points in each segment, or few segments with more points per segment. The Nyquist frequency will be the same whatever I choose, since that's determined by the interval between observations. But the low-frequency limit will differ, as will the increment between frequencies (which is determined by the lowest resolved frequency). There's no rule for how to handle this, and your decisions will depend whether you want small uncertainties or high resolution in frequency space.

Spectral Uncertainties

The problem with all of the spectra that we've computed so far (the amplitude of the Fourier Transform) is that we have not yet identified how to evaluate the uncertainty. By eye we can see that the spectra are fairly noisy. We know from computing means that the error of the mean decreases as we average more quantities together.

How do we incorporate more data into our spectra? You might imagine that you could improve your spectrum by extending the input time series from N to $2N$ data points for example. Unfortunately, although adding data points will change your spectrum, it won't reduce your noise or make the spectrum more precise at any individual frequency. Instead it will increase the number of frequencies for which you obtain results from $N/2$ to N.

Error bars for spectra rely on a similar principle. Our uncertainties in our spectra decrease as we average more spectra together. The challenge is to figure out how to obtain more spectra that can be averaged together. Typically what we do is to break our time series into segments, compute spectra for each of the segments, and average these to get a mean spectrum. Then we can rely on the fact that uncertainties in spectra are distributed like χ^2 to estimate the uncertainties.

If the unknown true spectrum is $X(f)$ and our estimate of the spectrum is $\tilde{X}(f)$, then we can consider the ratio $\tilde{X}(f)/X(f)$, where we use $N = \nu/2$ data segments. Formally, the probability that the estimated spectrum should be close in value to the true spectrum is:

$$P(\chi^2_{\nu,1-\alpha/2} < \nu\tilde{X}(f)/X(f) < \chi^2_{\nu,\alpha/2}) = 1 - \alpha \tag{5}$$

so if we want to find a 95% significance level, we set α to 0.05.

This error formulation differs from the usual error bars that we're used to seeing where we say for example that the true temperature should be This error formulation differs from the usual

error bars that we're used to seeing where we say for example that the true temperature should be the measured temperature plus or minus an uncertainty: $T = \hat{T} \pm \delta_T$. We can develop a similar expression for the true spectrum: $X(f)$ is in the range between $\nu\tilde{X}(f/\chi^2_{\nu,\alpha/2}$ and $\nu\tilde{X}(f)/\chi^2_{\nu,1-\alpha/2}$, where ν is twice the number of segments. This expression isn't very easy to interpret, since it varies as a function of frequency, and the estimated value $\tilde{X}(f)$ is not at the mid-point of the range.

Instead we'll make use of the ratio $\tilde{X}(f)/X(f)$ which does not depend on frequency. On a log plot, error bars defined by the range between $\nu/\chi^2_{\nu,\alpha/2}$ and $\nu/\chi^2_{\nu,1-\alpha/2}$ are the same size at all frequencies, so we can easily compare spectral peaks at different frequencies.

Some statistics books include look-up tables for χ^2, but we can compute it directly in Matlab. For $N/2$ data segments, the error limits are:

```
err_low = N/chi2inv(.05/2,N);
err_high = N/chi2inv(1-.05/2,N);
```

We can plot these values as:

```
semilogy([f f],[err_low err_high]*A);
```

where we set the frequency f and the amplitude A, so that the error bar ends up positioned in a convenient spot on the plot.

Now to have $N/2$ data segments, we have to split our long data record into shorter segments. We can do this by taking M data points at a time:

```
N=length(data);
M=segment_length; % define this value
for n=1:floor(N/M)
 d=data((n-1)*M+1:n*M); %select data for the nth segment
 fd(:,n)=fft(d);        % compute fft
end
sd=sum(abs(fd(1:M/2+1,:)).^2,2)/N; % sum over all spectra
%                                  (sum over 2nd index)
sd(2:end)=sd(2:end)*2;

nu=2*floor(N/M);
err_low = nu/chi2inv(.05/2,nu);
err_high =  nu/chi2inv(1-.05/2,nu);

semilogy(0:M/2,sd,[M/4 M/4],[err_low err_high]*sd(M/4))
```