Edge effects in spectra: detrending, windowing and pre-whitening

One of the big challenges in computing spectra is to figure out how to deal with edge effects. Edge effects matter no matter what the length of your record, so let's take this up from the point after you have decided how to segment your data and how to make sure that your spectra will resolve the frequencies/wavenumbers that you want to study. Fourier transform algorithms assume that you have an infinite data record. This means that in essence, from the perspective of an FFT algorithm, your N-point data segment will actually look like an infinitely repeating N-point data segment. Consider a linearly sloping data record computed as:

```
data=randn(200,1)+(1:200)'*.1;
```

as shown in Figure 1. Computationally this will be treated like a saw-tooth pattern by the FFT.
And as we know, the Fourier transform of a saw tooth pattern spreads energy over all frequencies, as indicated in Figure 3. We don't want that. To deal with edge effects, we have a number of options, including detrending, pre-whitening, and windowing.

Making red noise. Figures 1–3 were generated with fake data, but before we do any further calculations, let's create a second type of fake data that we can analyze in detail. Most geophysical quantities have red spectra, which imply more low-frequency variability than we'd find simply in a linear trend. (Red spectra are good for observing systems, because most of the energy is at low frequencies/wavenumbers that we are able to detect. If spectra were white or blue, we'd have to worry about the aliased effects of highly energetic high-frequency or high-wavenumber processes that we were not properly sampling.) For discussion purposes, let's consider data with a red spectrum. We can create a red spectrum in a couple different ways. One is to use an auto-regressive process. For example $x_{i+1} = \alpha x_i + s_n$, where α is a coefficient and s_n is a random number. But here let's take a more brute strength approach. If we create a matrix of Gaussian white noise:

```
a=randn(200,100);
```

we'll find a white spectrum:

```
% compute Fourier transform for each column in a
fa=fft(a);
% convert to spectrum by squaring absolute value and averaging over all
% columns in a
sa=mean(abs(fa(2:101,:)).^2,2);
% plot on loglog scale
loglog(1:100,sa); xlabel('frequency',''FontSize',14);
```

We can make our white data into red data by scaling each value by some power of frequency k:

```
% make a matrix of frequencies
k=[0:100 99:-1:1]'*ones(1,100);
fb=fa./k;
fb(1,:)=0;
sb=mean(abs(fb(2:101,:)).^2,2);
loglog(1:100,sb); xlabel('frequency,'FontSize',14);
```

Since we scaled the Fourier transform by k, the spectrum was scaled by k^2, which means that we have a k^{-2} spectrum. Now that we have the right spectrum, we can recover the data that we want using an inverse Fourier transform:

```
b=ifft(fb);
```

Detrend. If the data have a dominant trend, then you know a lot about the trend without bothering to compute spectra, so you should remove the trend first and focus on the remainder of the signal. For the data in Figure 1, detrending is guaranteed to produce straightforward white spectra, but for a red spectrum, we might encounter complications. For example:

```
c=detrend(b);
fc=fft(c);
sc=mean(abs(fc(2:101,:)).^2,2);
loglog(1:100,sc); xlabel('frequency,'FontSize',14);
```

(Note that in all of these examples, I've fudged the units a bit—there's a factor of 2 missing, plus a normalization. These are left as an exercise for the reader.)

As Figure 4 illustrates, detrending red data before computing the spectrum slightly flattens the spectrum, but it doesn't make the spectrum flat again. In this case, where the linear trend doesn't fully explain the red spectrum, you may still be concerned about other types of edge effects. You have several additional options.

Pre-whiten. One way to minimize the impact of the red spectrum is to "pre-whiten" the data. When we pre-whiten, we assume that we already know about the background slope of the spectrum, so our goal is to take that out, so that we can concentrate our analysis on the bumps an wiggles that are harder to detect in the spectrum.

One of the simplest ways to pre-whiten is to compute the time derivative of the data. If I have a time series s with Fourier transform $\hat{s}$, then what happens if I compute the time-derivative of the data, ds/dt. The Fourier transform of the time derivative is:

$$\mathcal{F}\left(\frac{ds}{dt}\right) = \int_{-\infty}^{\infty} \frac{ds}{dt} e^{-i\omega t} dt = -\int_{-\infty}^{\infty} s \frac{de^{-i\omega t}}{dt} dt = i\omega \int_{-\infty}^{\infty} s e^{-i\omega t} dt = i\omega \mathcal{F}(s) \tag{1}$$

```
% compute a first derivative of the data
db=diff(b);
% Fourier transform
fdb=fft(db); % compute the spectrum sdb=mean(abs(fdb(2:100,:)).^2,2);
% plot
loglog(1:100,sb,1:99,sdb,1:99,sdb./(1:99)'/^2*1000);
```

Figure **??** shows the impact of pre-whitening and then rereddening the spectrum by scaling by the frequency squared.

Window. Another common way to minimize edge effects is to multiply each data segment of length N by a window of length N. Typically we use a Hanning or Hamming window, both of which taper toward zero at the edges—this minimizes the discontinuity associated with effectively repeating the finite duration segments to create an infinite-duration record for the Fourier transform. (Since windowing minimizes the impact of points on the edges, we can reclaim some of the lost information by using segments that overlap by 50%.) Windowing cuts down on discontinuities, but it will have some impact on the next spectral slope, as illustrated in Figure **??**. Here we'll consider both the Hanning and Hamming windows, applied to raw (red) data or detrended data.

```
% Hanning window for red data
fd1=fft(b.*(hanning(200)*ones(1,100)))*200/sum(hanning(200));
sd1=mean(abs(fd1(2:101,:)).^2,2);
```

```
% Hanning window for detrended data
fd2=fft(c.*(hanning(200)*ones(1,100)))*200/sum(hanning(200));
sd2=mean(abs(fd2(2:101,:)).^2,2);

% Hamming window for red data
fe1=fft(b.*(hamming(200)*ones(1,100)))*200/sum(hamming(200));
se1=mean(abs(fe1(2:101,:)).^2,2);

% Hamming window for detrended data
fe2=fft(c.*(hamming(200)*ones(1,100)))*200/sum(hamming(200));
se2=mean(abs(fe2(2:101,:)).^2,2);

loglog(1:100,sb,1:100,sd1,1:100,sd2,1:100,se1,1:100,se2);
```

As Figure 6 shows, the windowed records have the same large-scale slope as the raw data, though windowing slightly shifts the amplitude. The Hanning and Hamming windows are effectively indistinguisable, and detrending slightly flattens the low-wavenumber spectral slope.

If these methods make so little difference, why fret about them? First, recall that we're playing with highly idealized data. You might encounter a data for which windowing or detrending would make a big difference. Second, if you care about the low-frequency or high-frequency ends of your spectra, it's clear that your filtering choices will make a big difference. And you want to focus your analysis efforts on robust results—if switching from a Hanning window to a Hamming window changes your interpretation entirely, that might be a warning that your signal is woefully insignificant.

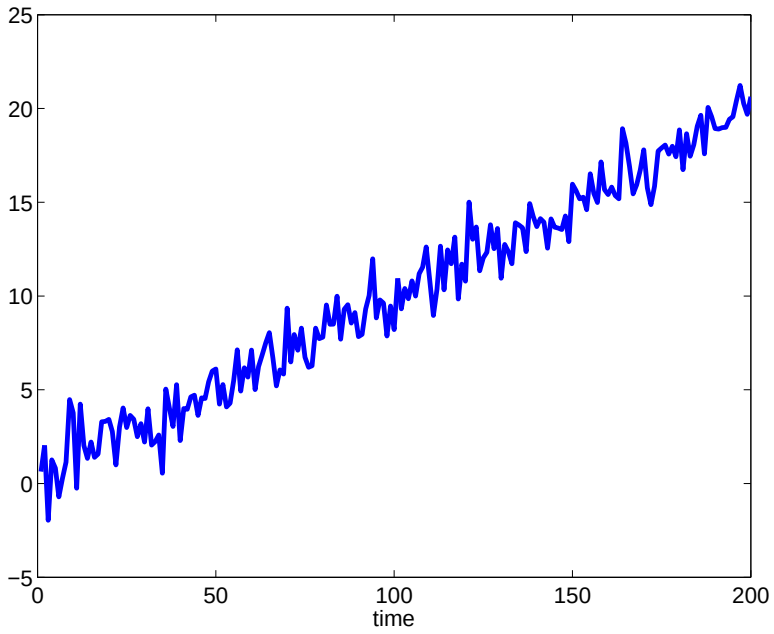

Figure 1: White noise with added trend, produced using

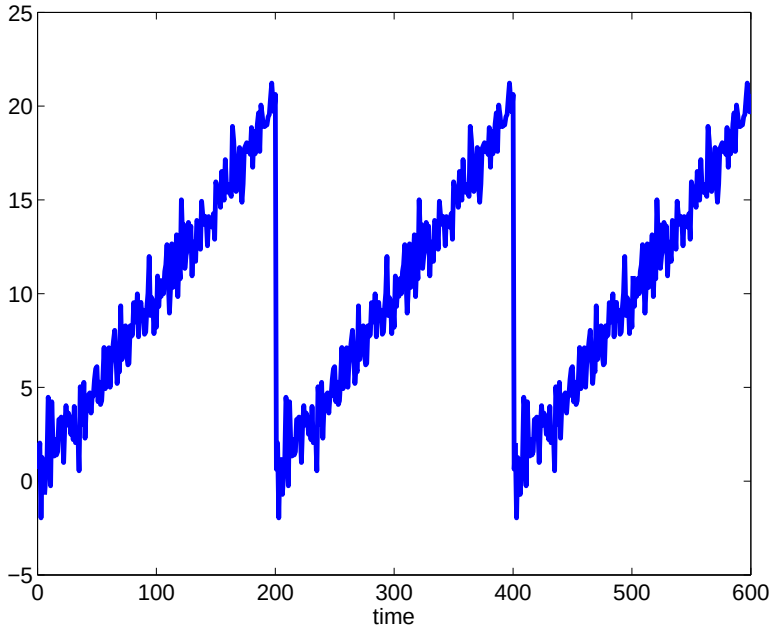

Figure 2: Data from Figure 1, repeated to represent FFT interpretation of record.

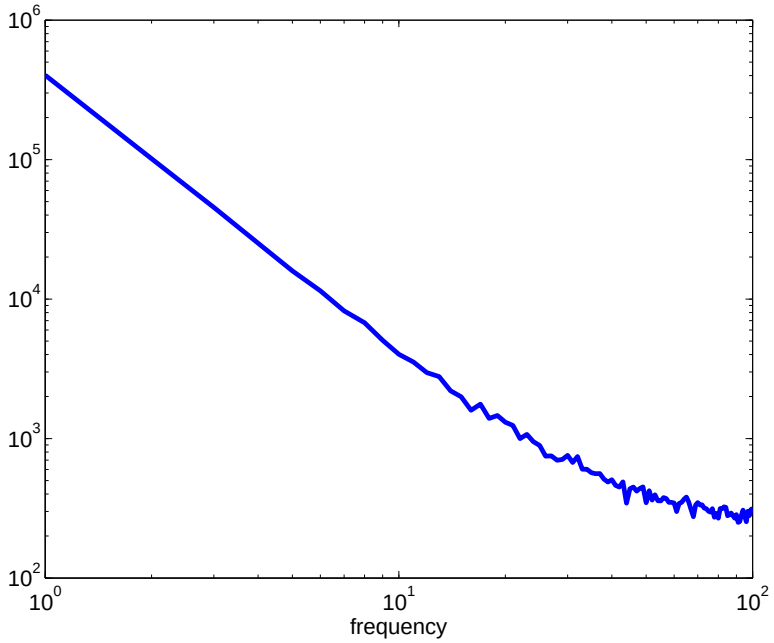

Figure 3: Spectrum computed from 100 realizations of Gaussian noise plus trend, such as the record in Figure 1.

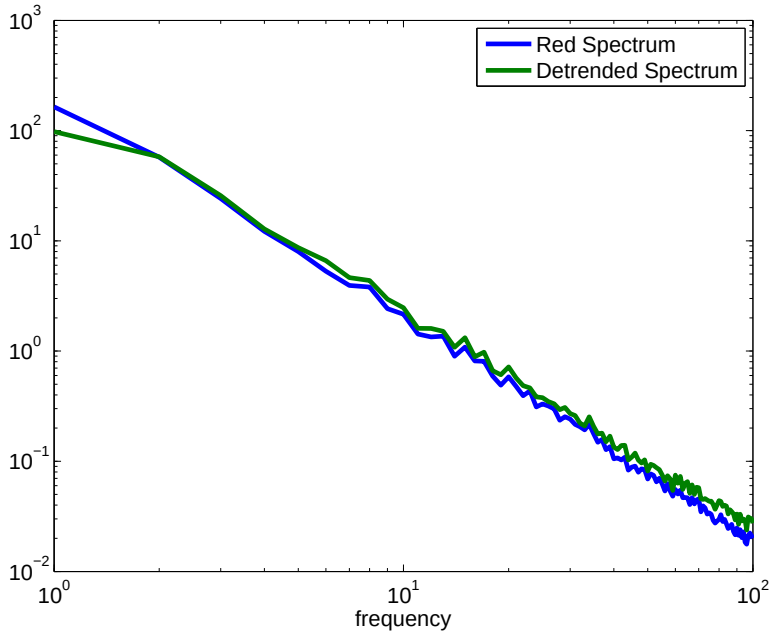

Figure 4: Spectrum computed from 100 realizations of red noise (blue), and with detrending before computing the spectrum (green).

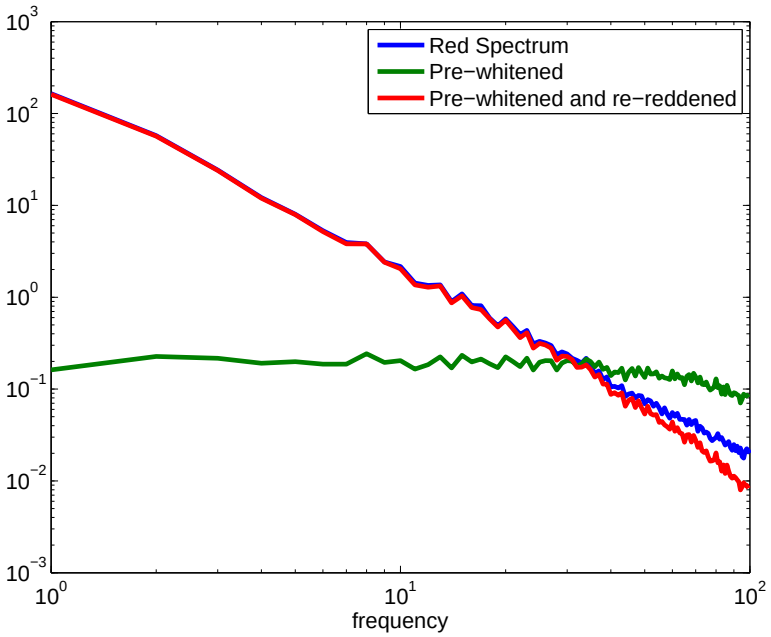

Figure 5: Spectrum computed from 100 realizations of red noise (blue), and with detrending before computing the spectrum (green).

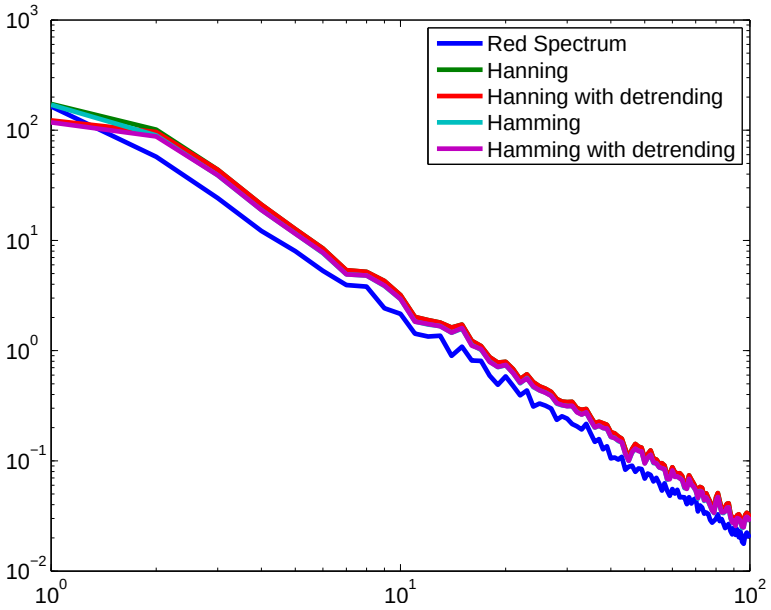

Figure 6: Spectra for original red time series and windowed records, with and without detrending.