Power Spectral Estimation With FFT (Numerical Recipes Section 13.4)

C. Kankelborg

Rev. February 10, 2020

1 The Periodogram and Windowing

Several methods have been developed for the estimation of power spectra from data (see *Numerical Recipes*, §§ 13.4,13.7-8). The simplest FFT estimate of the power spectrum is called the *periodogram* (*Numerical Recipes* eq. 13.4.5). The following example uses the periodogram technique, and illustrates the need for *windowing*. The accompanying figures (1, 2, 3) correspond to the three plots generated by the example.

```
% powerspec.m
% Periodogram Example

actual_f_bin = 1 + 13/2
   % This will be the exact (non-integer) bin for the frequency.
   % The 1+ is because Octave/Matlab arrays are unit offset.
   % The second term is a frequency in in cycles per domain.
N = 200; % Number of bins
x = (0:N-1)*2*pi*(actual_f_bin - 1)/(N-1);
y = cos(x);
y2 =  hanning(N)' .* y;

figure(1)
hold off
plot(1:N, y, 'r', 1:N, y2, 'k');
xlabel('x (spatial bin)')
ylabel('signal y(x)')
legend('raw data','windowed')
%axis([1,N])
```

```
print('powerspec1.pdf','-dpdfwrite')

yf = fft(y);
y2f = fft(y2);
ps = abs(yf).^2;
ps2 = abs(y2f).^2;

figure(2)
hold off
normalization = sum(y.^2)/sum(y2.^2)
plot(ps,'r')
hold on
plot(normalization*ps2,'k');
xlabel('k (spectral bin)')
ylabel('power spectrum')
legend('Raw power spectrum','PS of windowed data')
axis([1,N,0,max(ps)])
print('powerspec2.pdf','-dpdfwrite')

figure(3)
hold off
plot(ps(1:20),'r')
hold on
plot(normalization*ps2(1:20),'k');
hold on
xlabel('k (spectral bin)')
ylabel('power spectrum')
plot(actual_f_bin*[1,1], [0,1.1*max(ps2*normalization)],'b-');
centroid  = sum(  ps(1:20).*(1:20) )./sum( ps(1:20))
centroid2 = sum( ps2(1:20).*(1:20) )./sum(ps2(1:20))
plot(centroid*[1,1], [0,1.1*max(ps2*normalization)],'r--');
plot(centroid2*[1,1], [0,1.1*max(ps2*normalization)],'k--');
legend('Raw power spectrum','PS of windowed data','Exact signal frequency','Raw PS centroid','Wi
axis([1,20,0,max(ps)])
print('powerspec3.pdf','-dpdfwrite')
```

The example calculates the power spectrum using the FFT with and without *windowing* (in optics and image processing, the customary term is *apodization*). The Octave/MATLAB function `hanning(N)` produces an N-element array of the form $\frac{1}{2}\left[1-\cos\left(\frac{2\pi(n-1)}{N-1}\right)\right]$. As you can see in figure 1, this eliminates the wraparound discontinuity of the data. Since the Hanning window is a smooth function with a broad peak, its only side effect is that the FFT of the windowed data will be smoothed (convolved with the Fourier transform of the window, which is itself a narrow, peaked function). Without

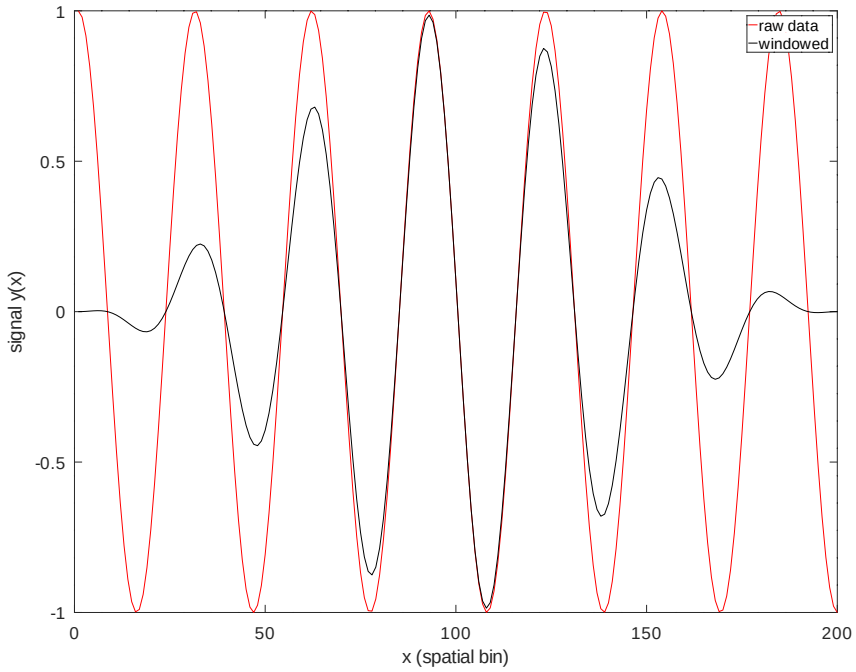

Figure 1: The signal and its hanning-windowed counterpart.

the windowing, there is a $1/\nu^2$ tail artifact in the power spectrum; this is eliminated in the power spectrum of the windowed data, at the expense of broadening the peak (figures 2 & 3).

2 Repeatability and Uncertainty

Often, we are looking for the power spectrum of some stochastic phenomenon, like the avalanche effect in a diode or the pattern of waves on the surface of the ocean. In cases like this, the periodogram is merely an estimate of the true power spectrum. Sampled data misses what came before the sampled interval, what came after, and what came in between the samples. Ideally, we would measure the signal over an infinite interval of time, at infinitesimal sampling period. But it turns out that once we have a high enough sampling frequency and a long enough duration to catch the frequencies of interest, getting more data only adds noise to the periodogram. *Numerical Recipes*

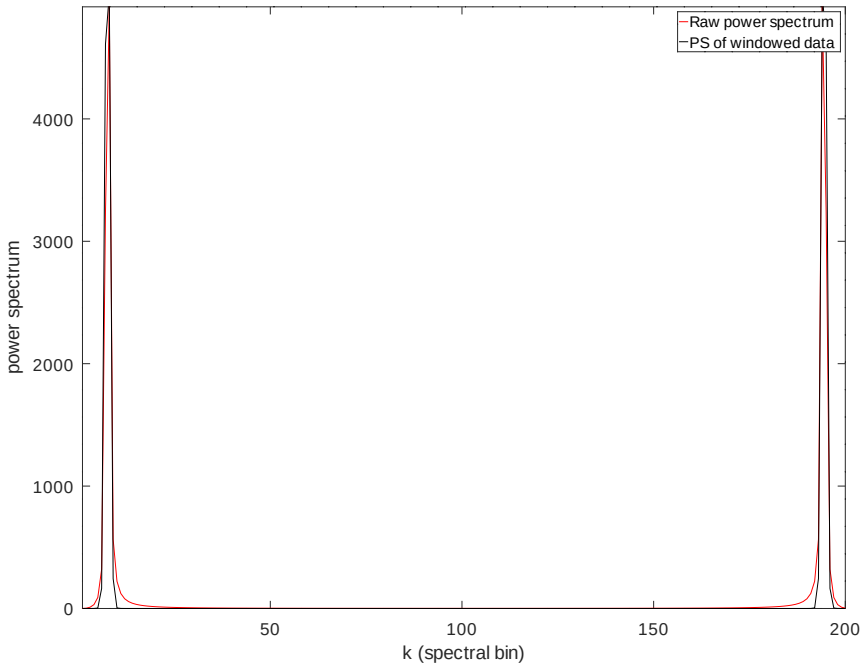

Figure 2: The power spectrum of the signal and its hanning-windowed counterpart.

asserts that the standard deviation of the periodogram at any frequency is equal to its mean (100% error!). This does not change as N increases (the periodogram reaches for higher spectral resolution rather than tightening up the error bars). One solution to this problem is to partition the data into many segments. The optimal information per data point is obtained when the second half of each segment overlaps the first half of the next segment.
Question: What assumptions underlie the 100% error estimate given by *Numerical Recipes*?

3 Example

Begin by reviewing the following codes.

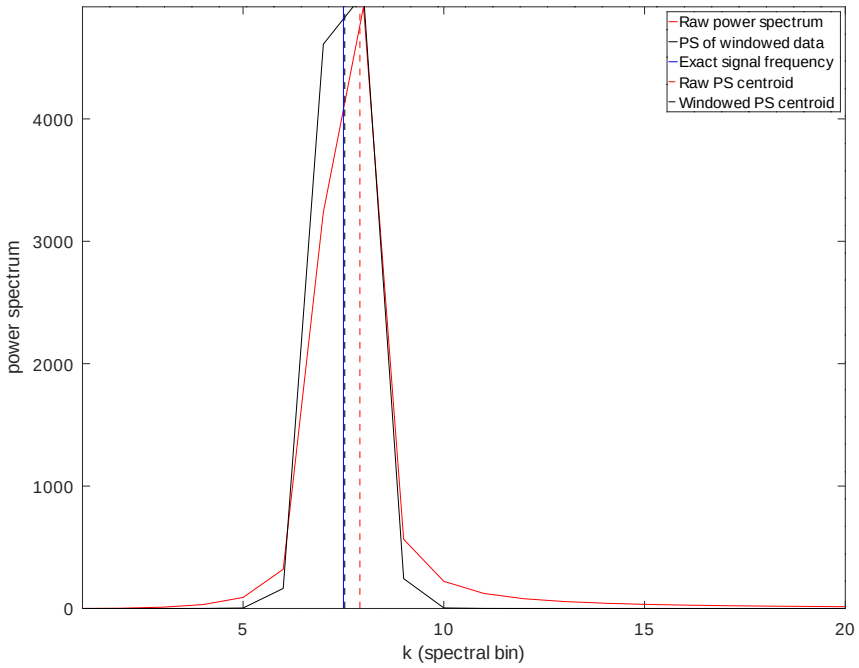

Figure 3: Same as the previous figure, but zoomed in. The periodogram calculated from the raw data has tails that fall off like $1/k$ due to the discontinuity across the domain boundary. Vertical lines compare estimates of the signal frequency. The Hanning window offers a marked improvement in frequency estimation and eliminates the $1/k$ tails.

3.1 Power spectral estimation code

```
% spectrum.m
function [powerspec,frequency] = spectrum(filename, N, dt)

% Power spectral estimation by partitioning and windowing. The program is
% able to analyze data sets larger than memory by processing a binary stream.
% Big-Endian ieee floating point numbers (32 bits) are assumed.
% Binary file given by filename is read in chunks of N floats,
% assumed big-Endian. Power spectrum is built 2N samples at a time.
% The sampling period is optionally given by dt.
%
% CCK 2017-Feb-09 ported to Matlab (now runs in both Matlab & Octave).
```

```
if (exist('dt') ~= 1) dt=1.0;
end

window = ( 1 - cos( 2*pi*(1:(2*N))/(2*N) ) ) / 2;  % Hann window

frequency = (0:N) / (2*N*dt);  % Convenient frequency axis
   % (cycles per unit time, with same time units as dt).
   % Note that only the DC, postive, and Nyquist frequencies are represented here;
   % negative frequencies are redundant.

coeffs2 = zeros(1,2*N);  % Create empty array for squared Fourier coefficients

datafile = fopen(filename,"r");
% (We now need a do-until loop, which exists in Octave and has finally
% been added to MATLAB, but unfortunately with a different syntax. The
% following code, while slightly awkward, should work in both languages.)
M=0;
while (1)
   [chunk, count] = fread(datafile, [1,N], "float", "ieee-be");
   if (count ~= N) break; end;
   if (exist('lastchunk') == 1)
      coeffs2 = coeffs2 + abs( fft(window.*[lastchunk,chunk]) ).^2;
         % matlab lacks +=.
   end
   lastchunk = chunk;
   M=M+1;
end
stdout = 1; % Octave defines this constant by default.
fprintf(stdout, "%u chunks of %u floats, remainder %u loaded from file: %s \n", M, N, count, fil

powerspec = coeffs2(1:N+1)/(2*M*N)^2;
   % DC thru Nyquist frequency.

fclose(datafile);
```

3.2 Example time series

```
% timeseries4.m
%
% Produce a time series in M chunks of size N, containing 2 sinusoidal
% signals and some noise. Save as a binary file of floats (datafile.bin).
% Takes about 40s to write 1GB (timeseries(1024,1024*256)), 2014-Jan.
%
% CCK 2017-Feb-09 ported to Matlab (now runs in both Matlab & Octave).
```

```
function timeseries(M,N, SNR)

% The optional argument SNR specifies the signal-to-noise ratio
% (default = 1.0).

if ( exist("SNR") ~= 1 ) SNR = 1.0; end

% Physical characteristics of signal
e = exp(1); % Matlab lacks this constant, which is standard in Octave.
M           % number of chunks
N           % chunk size
SNR         % signal-to-noise ratio
A = 1.0     % amplitude (V)
T = e       % period (s)
freq = 1/T      % frequency (Hz)
freq2 = e*freq  % second signal
dt = 0.25       % sampling time (s)
% Physical characteristics of noise
sigma = A / sqrt(SNR * 2)   % standard deviation (V)
mean =  0.0;                % mean (V)

fileID = fopen("datafile.bin","w")
phasenoise=0; % initialize
noise_old=0; % initialize
for i=0:(M-1)
   t = dt*( (0:(N-1)) + i*N);
   chunk = A/sqrt(2) * sin(2 * pi * freq * t); % signal 1
   % phasenoise = phasenoise(end) + cumsum( normrnd(0,freq2*dt,1,N) );
   phasenoise = phasenoise(end) + cumsum( freq2*dt*randn(1,N) );
      % phase changes in a random walk freq2*dt radians per time step. After n steps,
      % the expectation value of the phase change is <phi> = sqrt(n) * freq2*dt.
      % The coherence time, tau, is the time at which <phi> = pi/2,
      % which leads to tau = n*dt = dt * ( pi / (2*freq2*dt) )^2 = 9.9 sec.
   chunk = chunk + A/sqrt(2) * cos(2 * pi * freq2 * t + phasenoise);
   %noise = normrnd(mean, sigma, 1, N);   % noise
   noise = mean + sigma * randn(1, N);   % noise
   noise = noise + 0.9*[noise_old(end), noise(1:end-1)]; % redden the noise
   chunk = chunk + noise;
   noise_old = noise;
   fwrite(fileID, chunk, "float", "ieee-be");
end
fclose(fileID);
```

3.3 Script to run the example and produce figures

```
% timeseries4test.m
% Test of spectrum.m using timeseries4.m
%
% CCK 2017-Feb-09 ported to Matlab (now runs in both Matlab & Octave).

M=1000;  % Number of chunks
N=1024;  % Chunk size
dt = 0.25; % To agree with timeseries4.m
SNR=10;
timeseries4(M, N, SNR);
dt = 0.25;

fileID = fopen("datafile.bin","r");
signal = fread(fileID, M*N, "float", "ieee-be");
time = dt*(0:(M*N - 1));
figure(1)
hold off
plot(time(1:2*N), signal(1:2*N), "-b")
xlabel('time (s)')
ylabel('signal (V)')

figure(2)
hold off
[spec, frequency] = spectrum("datafile.bin", M*N/2, dt);
semilogy(frequency, spec,'r')
hold on
[spec, frequency] = spectrum("datafile.bin", N, dt);
plot(frequency, spec,'b')
xlabel('frequency (Hz)')
ylabel('power spectrum')
axis([0,frequency(end),min(spec),max(spec)*10])
```

3.4 In-class exercise

1. Run timeseries4test.m, and examine the plots.
2. Do you see the "100% error" in the ordinary periodogram (red)?
3. Does averaging many partitions help:
   (a) Estimation of the red noise spectrum?
   (b) Determination of the line shape for the peak broadened by phase noise?

(c) Frequency estimation for the 0.36788 Hz peak?

4. Discuss the implications for power spectral analysis of *resolved* and *unresolved* spectral features.

It generally helps to increase spectral resolution when looking at unresolved peaks. But with well-resolved features like the red noise and the broadened peak in the above example, better power spectral estimation is obtained by averaging many windowed partitions.