12.714 Practical Section 02 May 02, 2012

Part 1: Analytic transforms versus FFT algorithm

*(a) For a box car time domain signal with width 2*BW and amplitude BA, compare the analytic version of the Fourier transform with the numerical calculation from the Matlab fft function.*

Part1a of Matlab code generates these results. The analytic expression is given by

```
AnalBoxCar = 2*AB*BW*sin(2*pi*BW*f)./(2*pi*BW*f);
```

where AB is amplitude and 2*BW is width of boxcar and f is frequency in cycles/unit time.

An example matlab code is below.

```
N = 65; dt = 2*BW/N;
BoxCar = AB*ones(1,N);

NF = 1024; % number of points in fft
FFTBoxCar = fft(BoxCar,NF)/N;
% Now rotate the fft to shift the center of the box car
time to zero
toff = -floor(N/2)*dt;
f = [0:NF/2-1]/(dt*(NF)); % Freqeuncies in fft
FF = FFTBoxCar(1:NF/2).*exp(-2*pi*i*f*toff);
```

Here we use 1024 padded FFT (the boxcar is 65-samples wide). These values can be changed to see effect. The important step here is to rotate the fft because the boxcar is matlab runs from 1 to 64 compared to the analytic expression which is expecting –BW to BW of the box car.

Notes: There still seems to be a small problem in the rotation of the fft does not generate a pure real signal, which it should have. The consequence is a small difference, which grows with frequency. These difference most likely arise from the finite duration of the fft. The sampling is a comb of Dirac functions but the finite duration is a long box car whose transform will be convolved with the Dirac comb resulting in broadened and finite duration comb.

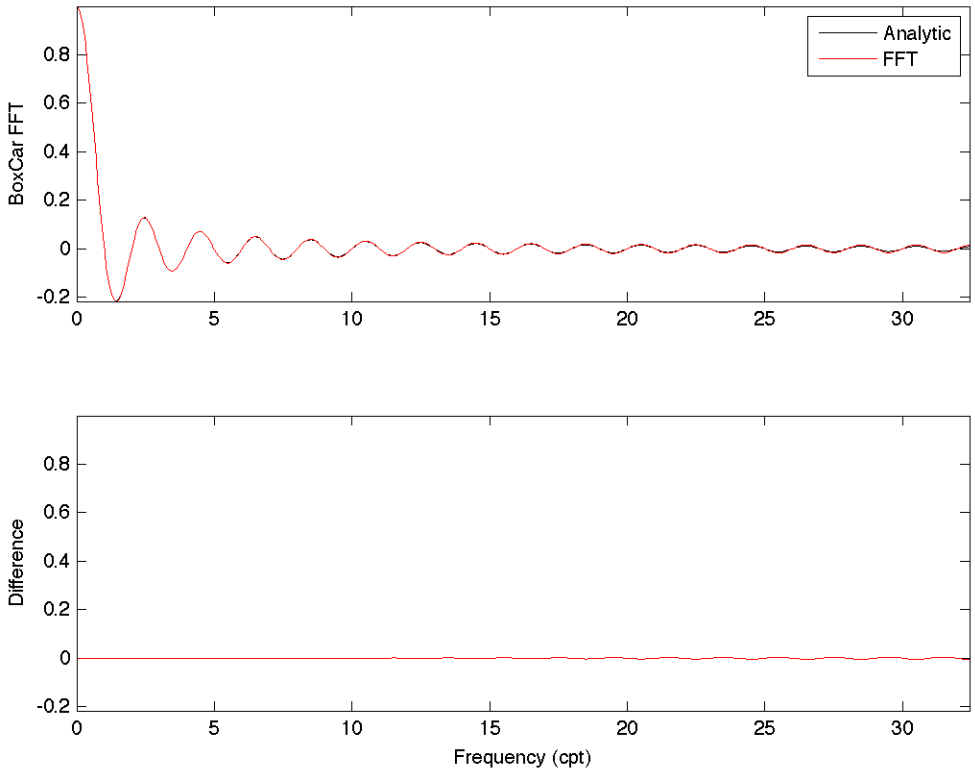

Fig 1a. Comparison of BoxCar analytic Fourier transform and Matlab FFT.

(b) Repeat the same analytic and numerical calculations for a triangle function.

Results here are generated in a similar process as before. The analytic expression is

```
AnalTriang = 4*AB*BW*(sin(2*pi*BW*f)./(2*pi*BW*f)).^2;
```

The matlab code in this case generates the triangle function by convolving two box car functions. Again a time offset is applied to make the triangle function centered on time 0.

The differences here are smaller than in Figure 1. (Zoom can be used in Matlab to see differences).

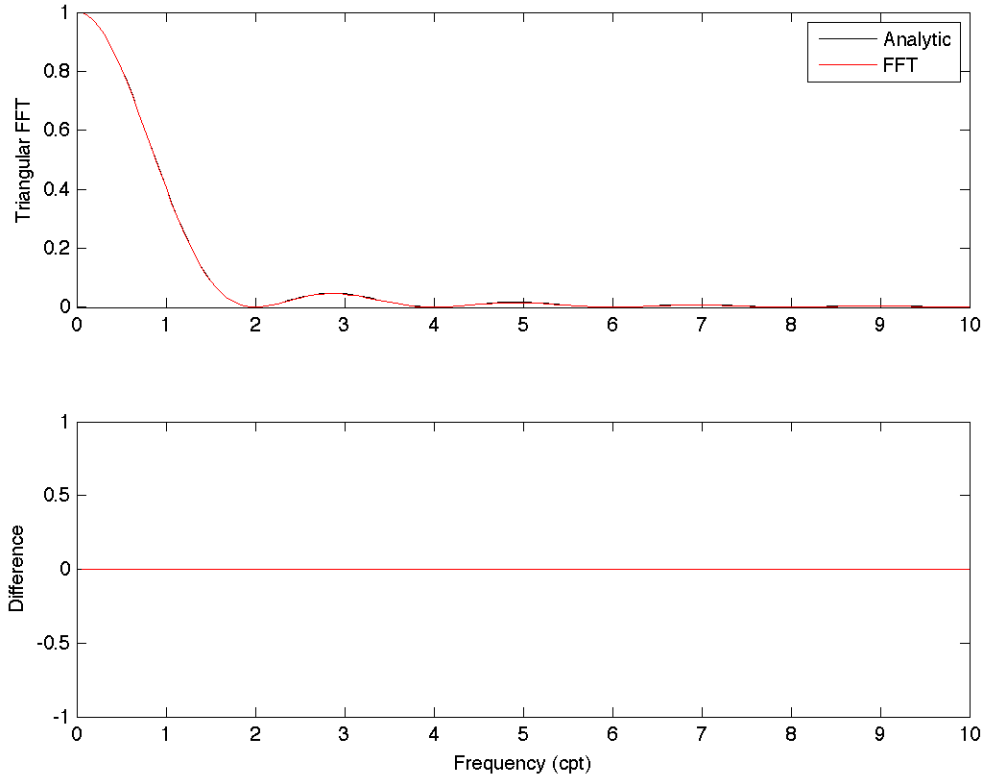

Fig 2: Analytic and FFT for triangle function.

Part 2: Repeating functions.

Generate a linear function (e.g., `gt1 = [-4:0.125:3.99];`*). Compute the power spectrum (conj(fft).*fftj) and plot in dB.*

Replicate the function so that there are two and four saw tooth functions. Compute the power spectrum of the replicated version and compare with the original

Solution to this problem is relatively straightforward. The increasing lengths of the time series means the frequency spacing is smaller for the longer durations and the amplitude changes because the pattern is being repeated. Results are shown in Figures 3 and 4.

Straightforward calculation is possible in this case. The mean value sets the power at zero frequency. In the way the code is written, 64 samples are used in the sawtooth, which generates exact zeros in the fft. Using 65 samples, generates small but not zero values and so the figures above have lines shown heading to these small values. Padding the fft with zeros (by using the second argument in the fft)

also generates interesting results by filling in the spectrum between the values shown above.

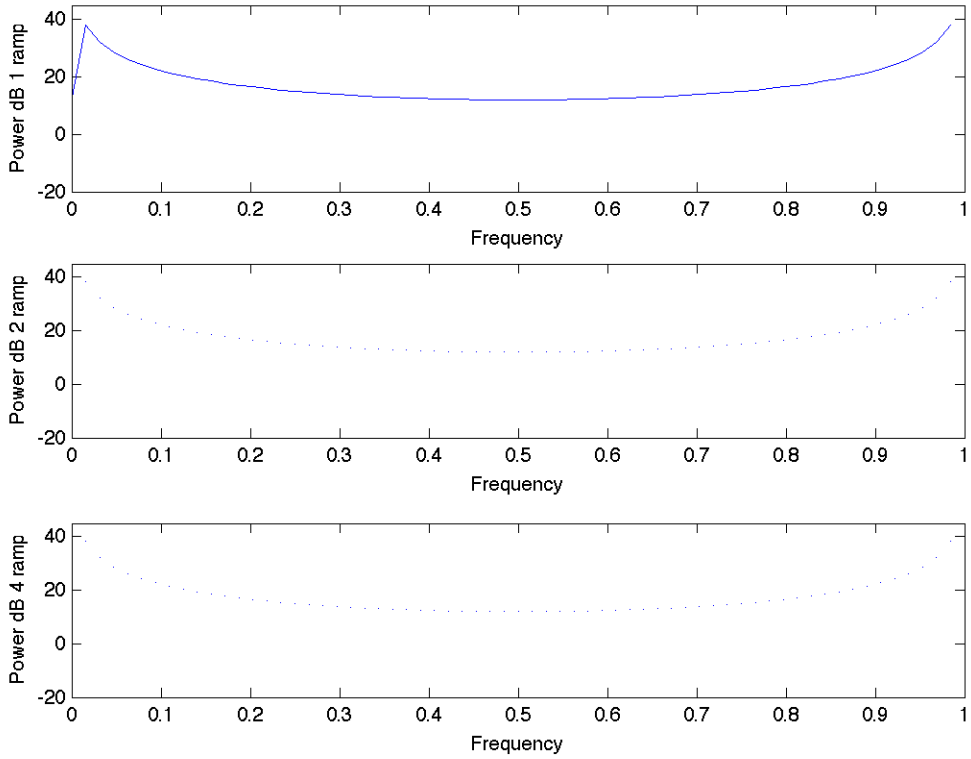

Figure 3: Comparison of power spectrum from 1, 2 and 4 saw tooth functions. The amplitudes have been scaled to account for the different numbers of terms. In the lower two parts, the line is dotted due to zeros (and this –infinity in the log) in the power spectrum.

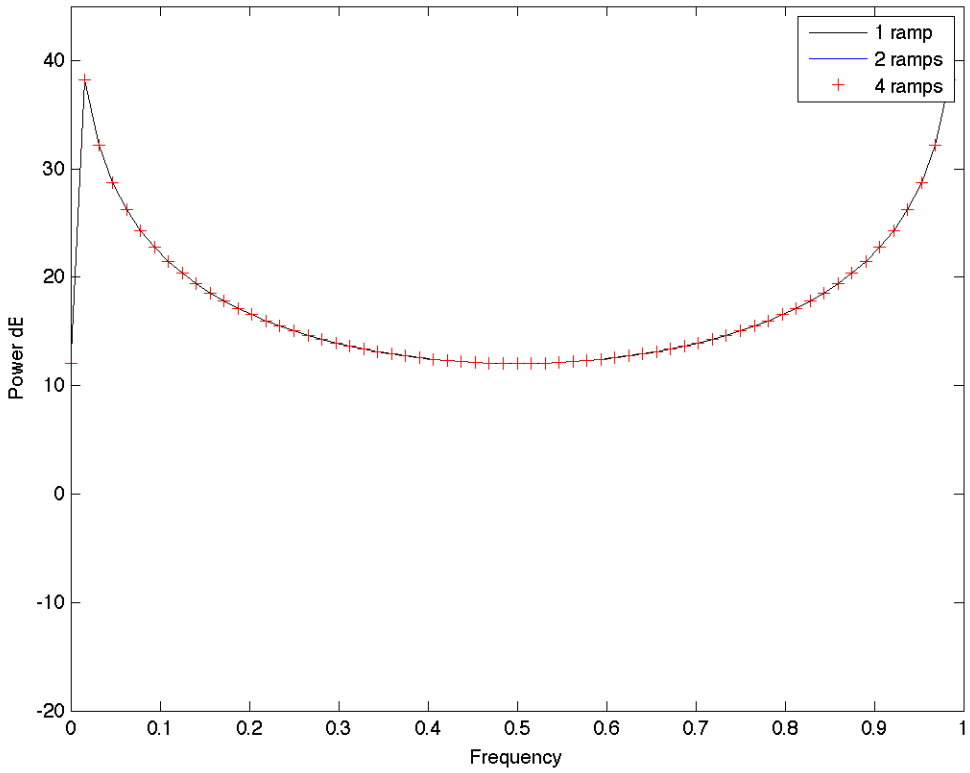

Figure 4: As above but with the values all shown on one plot

Part 3: Process noise example

(a) Generate a realization of a first order Gauss Markov process (an AR(1) model) and compute the auto-covariance sequence and the power spectral density. Compare the computed power spectral density with the theoretical expectation. Also generate a combined FOGM plus white noise model that has the same variance as the first model.

As a specific example: generate 10240 daily samples for a process with 100-day correlation time with a long-term variance of 2 mm^2. For the FOGM plus noise case assume the white noise has a standard deviation of 1 mm.

Generating time series is easy once the appropriate standard deviatons are determined. The code below shows the procedures used. Notice here that the random number generated is seeded so that the same series is generated with each run. Sigep1 and sigep2 below are the standard deviations of the white noise that will generate the desired long term standard deviation for the process.

```
randn('seed',120) ;  % Ensure we get same sequence each
time
tau = 100; % Decay time 100 days
dt = 1;   % Data spacing in days
beta = exp(-dt/tau);
```

```
sigtot  = sqrt(2.0); % This is going to be sigma total
sigwn   = 1.0 ; %  White noise sigma (uncorrelated)
sigfog1 = sqrt(sigtot^2-sigwn^2) ; %  Long terms standard
deviation for FOGM+WN
sigfog2 = sigtot ; % Long term sigma with just FOGM
sigep1  = sqrt(sigfog1^2*(1-beta^2)); %  Daily noise
driving FOGM process
sigep2  = sqrt(sigfog2^2*(1-beta^2)); %  Daily noise
driving FOGM process
```

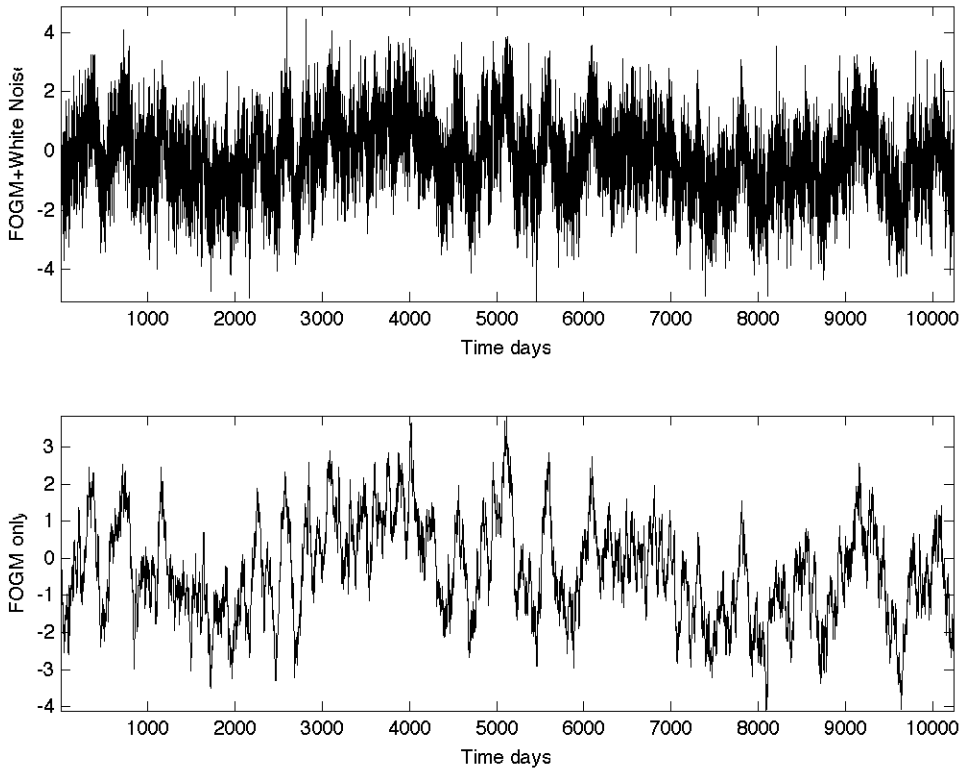

Figure 5: Realization of FOGM+WN and FOGM time series. The noise driving the FOGM is the same in both processes. The standard deviations are FOGM_WN 1.38; and FOGM only 1.33 mm (Expectation is 1.41 mm).

The autocovariance sequences are shown in Figure 6 and computed from the PSD functions (see original lecture 1 matlab code).

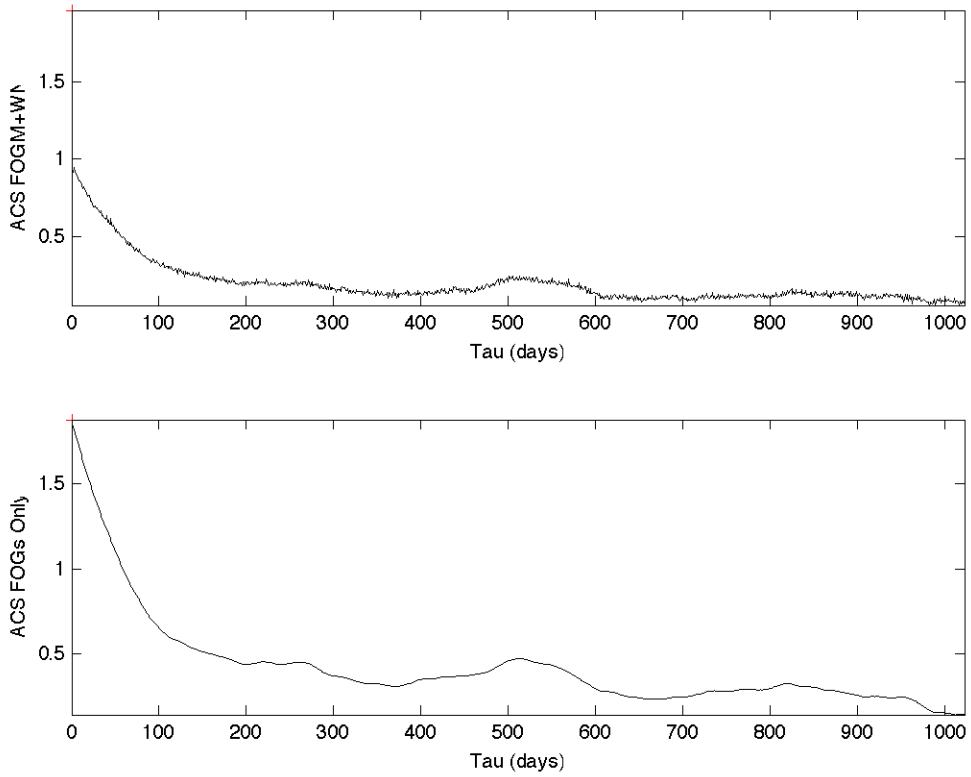

Figure 6: Autocovariance sequences for the FOGM+WH and FOGM series shown above.

The theoretical model for the PSD is given by

```
PSD1Th = (sigep2^2./((1-beta)^2 + 4*pi*fall.^2)+sigwn^2)/2;
```

where fall is the frequency sequence for the duration and sampling of the data.

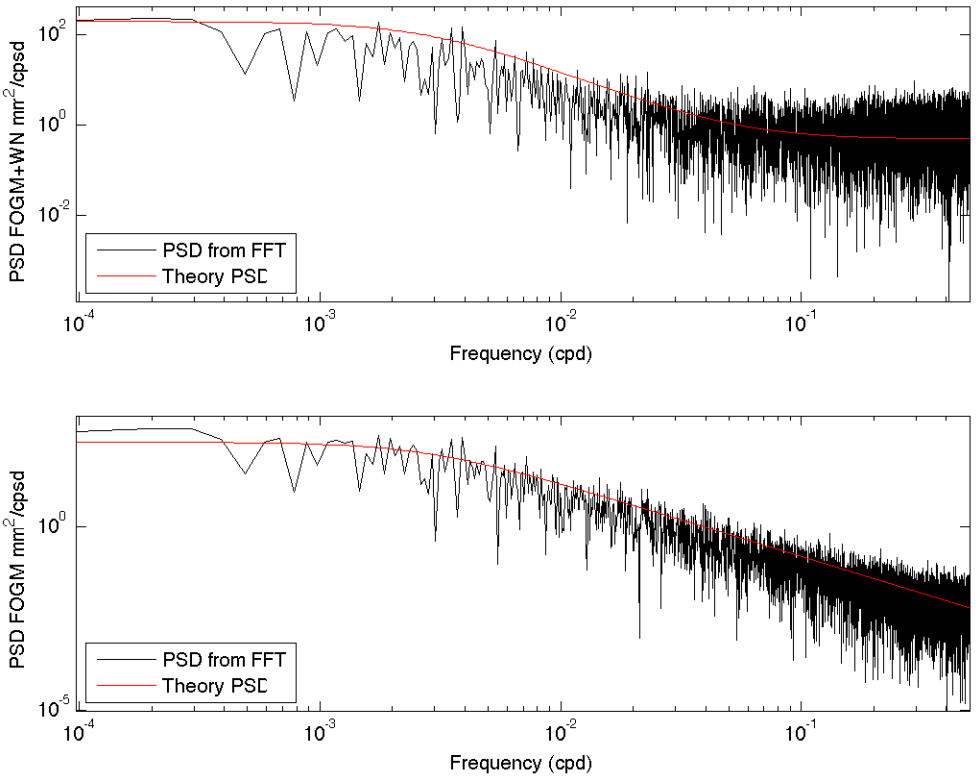

Figure 7: PSD for the FOGM+WN and pure FOGM process.

(b) Divide the data set into 10 segments, and compute the power spectra of each section. Average these power spectra and compare with the theoretical estimates.

The original 10240 samples are divided into 10 blocks each of 1024 samples. The PSD from each section are averaged and shown in figure 8. Later in the case we will examine better ways of reducing the noise and leakage in these PSD estimates.

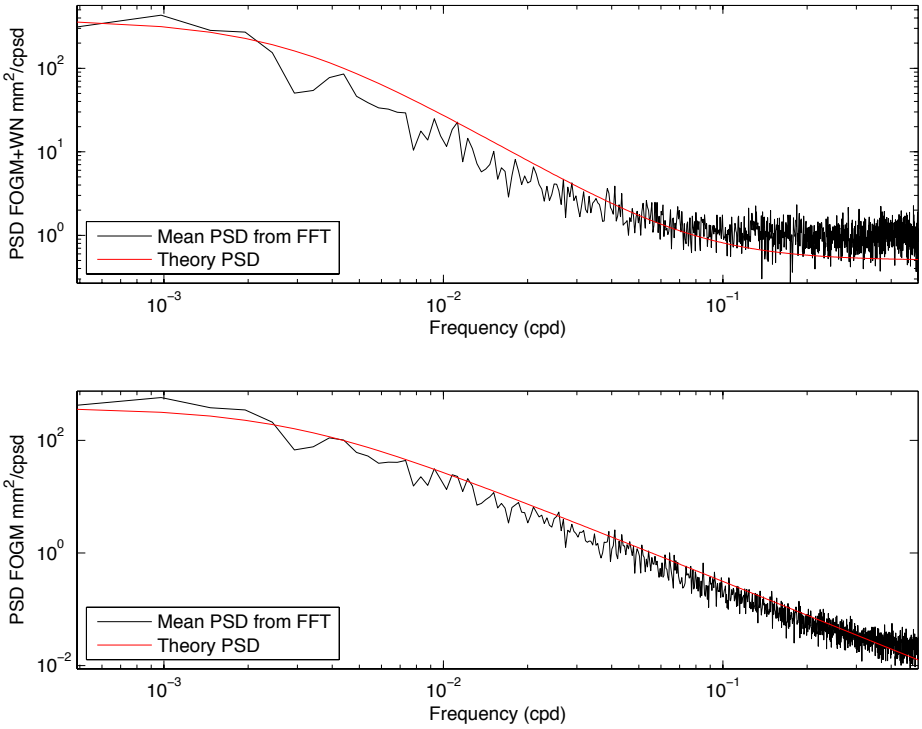

Figure 8: PSD for average of 10 realizations each with 1024 samples.