SIO221A Homework #7 (Eric Gallimore)

Table of Contents

Initialization

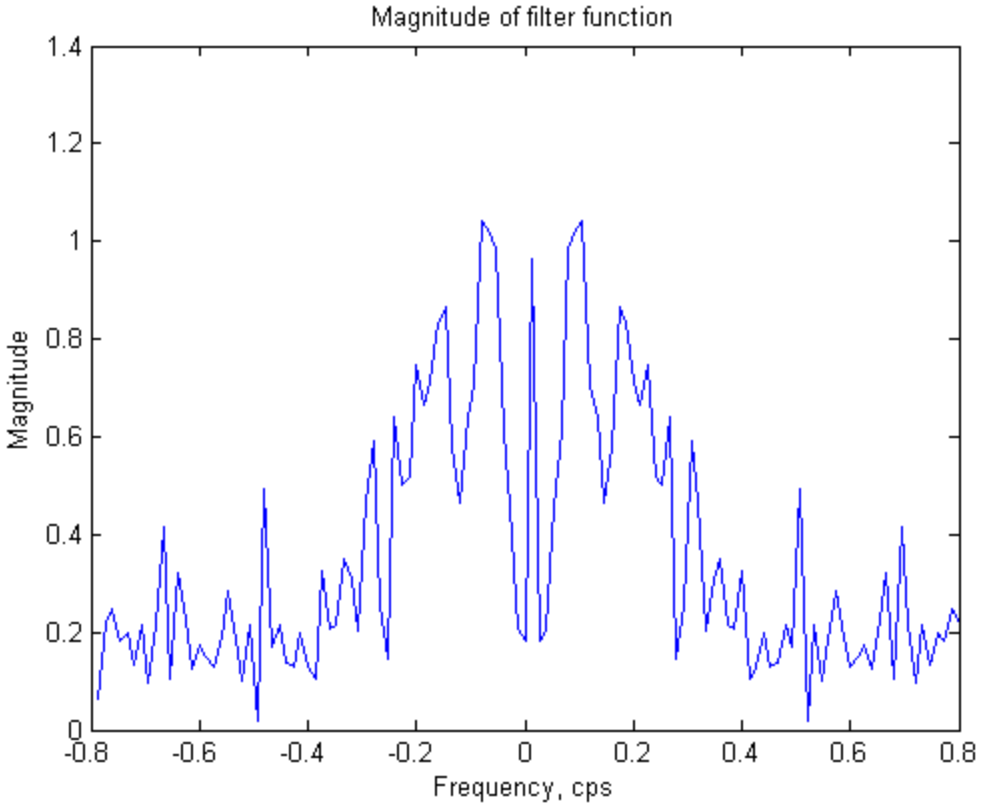
```
function hw7()

clear();
close all;
load('velb4degl.mat');

% Figure out how to partition this to get 50 DOF
dof = 50;
len_complete = size(velb4, 2);
points_per = floor(len_complete / dof);
```

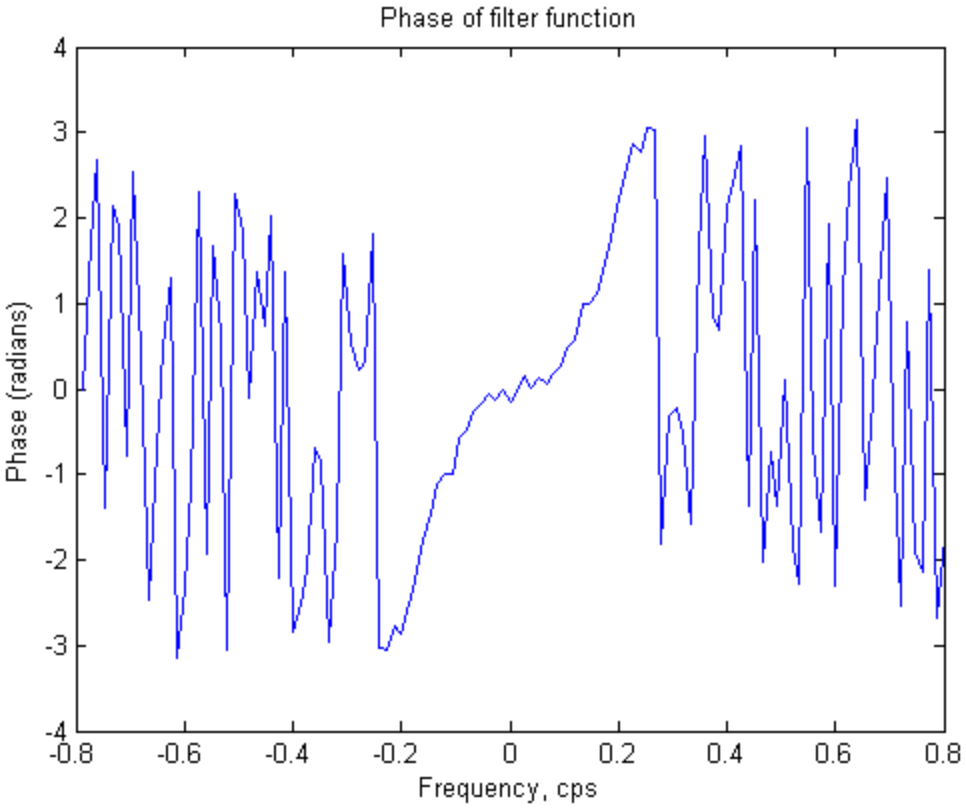
Step 1

Estimate H(sigma) between ranges 100 and 90 using spectra and cross spectral estimates calculated at 50 dof, as per Homework 6. Plot the Gain and Phase functions at positive and negative frequencies.

```
sample_interval = 0.625; % seconds

velocity_100 = reshape(velb4(100,:), [], 1);
velocity_90 = reshape(velb4(90,:), [], 1);
velocity_80 = reshape(velb4(80,:), [], 1);

[cross_spec_100_90, freq_bins] = crossspectrum(...
      velocity_90,velocity_100,sample_interval, dof);
[ehat_100_real, ehat_100, freq_bins_real, freq_bins, delta_f] = ...
    varspec_est_multidof(velocity_100, sample_interval, dof);

H_100_90 = cross_spec_100_90(1:length(freq_bins)) ./ ehat_100;



mag_h_100_90 = abs(H_100_90);
phase_h_100_90 = angle(H_100_90);

figure()
```

```
plot(freq_bins, fftshift(mag_h_100_90))
xlabel('Frequency, cps');
ylabel('Magnitude');
title('Magnitude of filter function');

figure()
plot(freq_bins, fftshift(phase_h_100_90));
xlabel('Frequency, cps');
ylabel('Phase (radians)');
title('Phase of filter function');
```



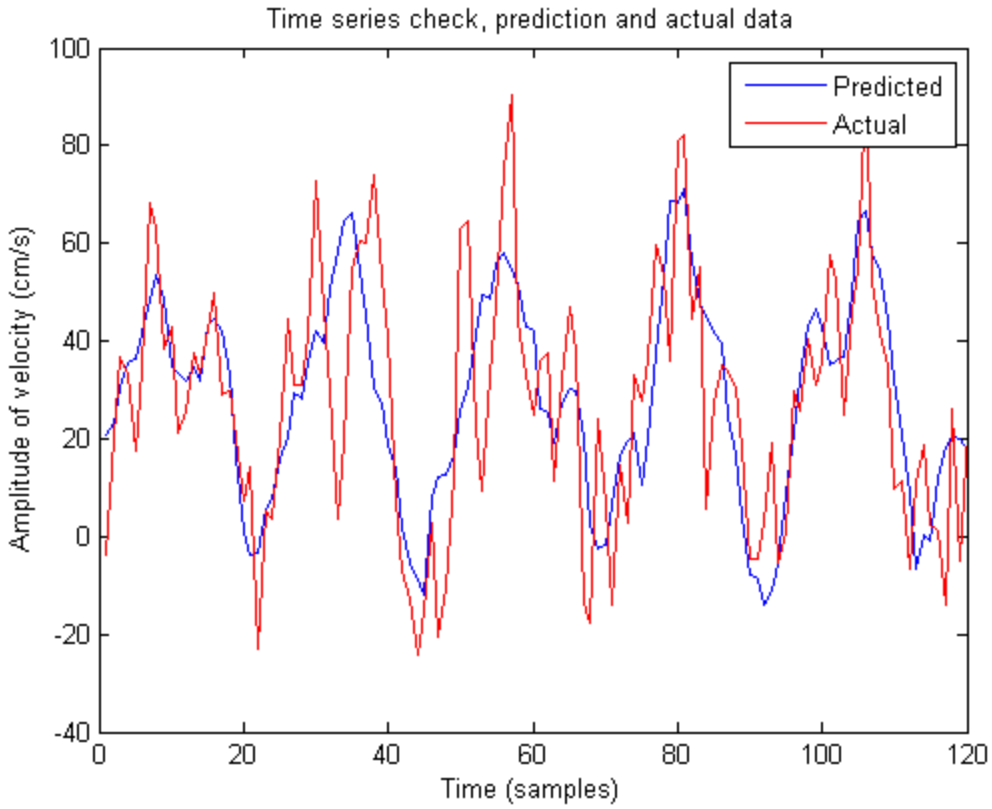


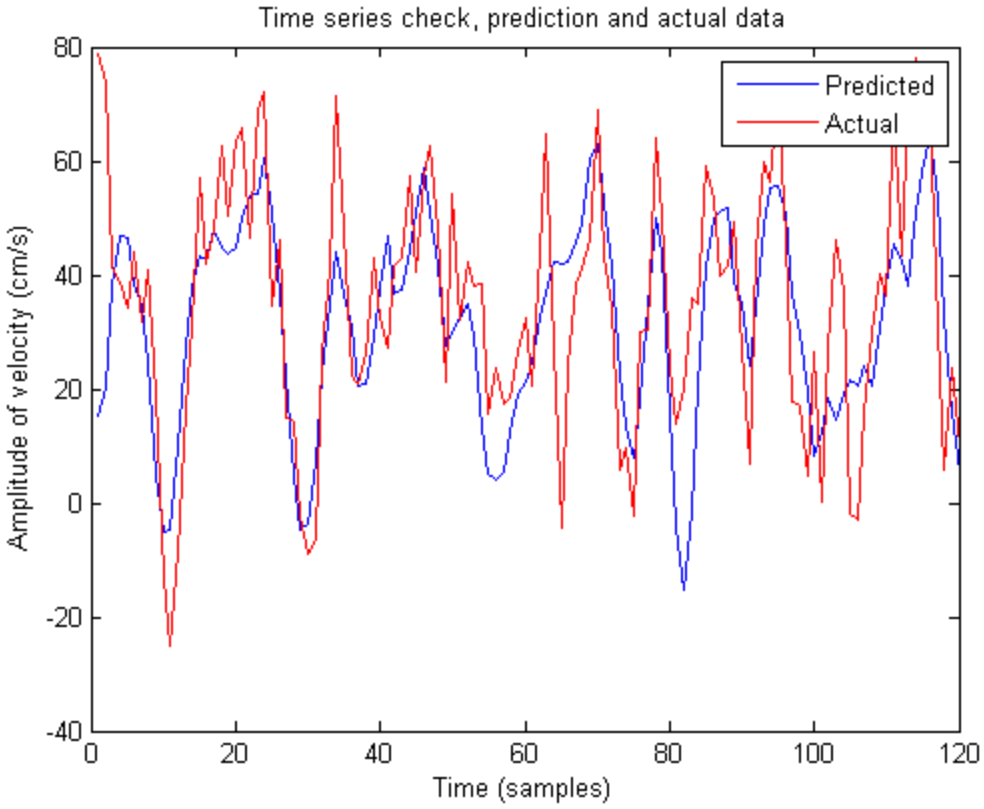
Step 2

You will probably have broken the original data series into 25 segments to calculate H(!). So go back through the 25 segments, Fourier transform the velocity data at range “90” (the forcing input), multiply by the 50 dof H(!), and then inverse-transform to get the predicted record as a function of time at range “80”. Hint: Take care to treat the negative frequencies of the input data and of H(!) correctly, so that your prediction is real. The inverse-transform output should have a very small imaginary part if this is done correctly.

```
% Testing:
%H_100_90 = ones(1,length(H_100_90));

% Get the 25 segments
number_of_sets = 25;
set_size = floor(length(velocity_90) / number_of_sets);
 for i=1:number_of_sets
        start_idx = (i-1) * set_size + 1;
        end_idx = start_idx + set_size - 1;

        data = velocity_90(start_idx:end_idx);
        a_90(:,i) = fft(data');
        a_predicted_80 = a_90(:,i)' .* H_100_90;

        predicted_80(:,i) = ifft(a_predicted_80');

        error(:,i) = predicted_80(:,i) - velocity_80(start_idx:end_idx);
```

```
        if ((i==5) || (i==20))
            figure();
            plot(1:length(predicted_80(:,i)), predicted_80(:,i))
            hold on;
            plot(velocity_80(start_idx:end_idx), 'r')
            hold off;
            legend('Predicted', 'Actual');
            xlabel('Time (samples)');
            ylabel('Amplitude of velocity (cm/s)');
            title('Time series check, prediction and actual data');
        end

end
```





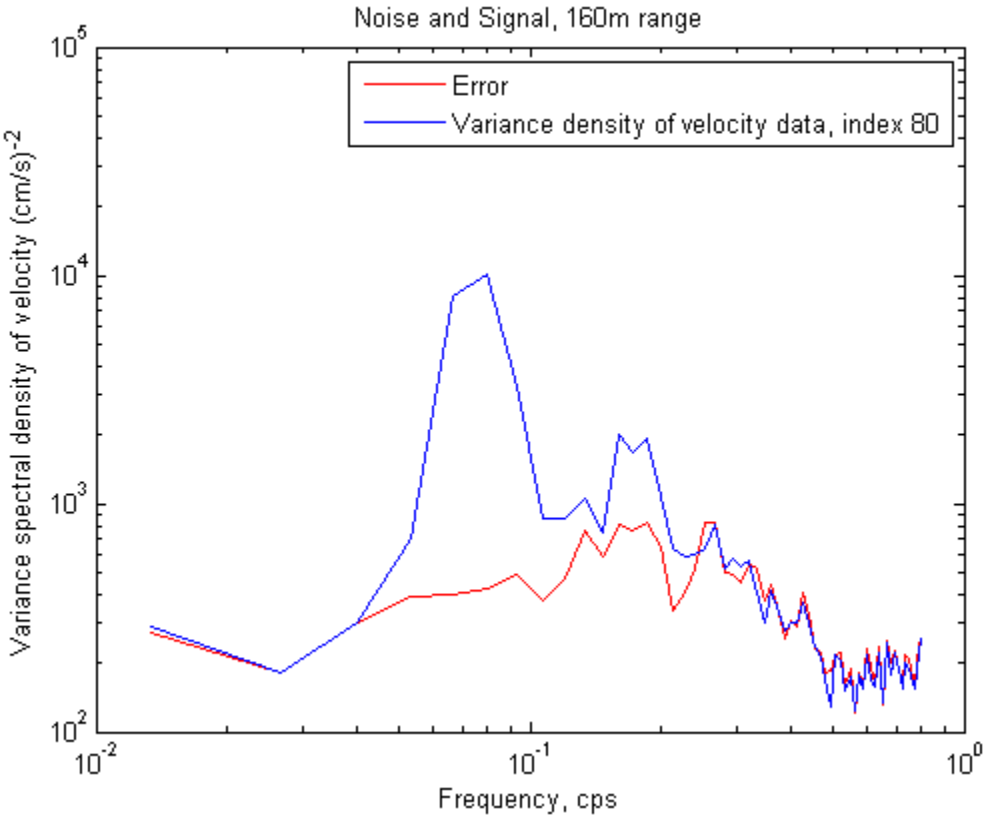
Step 3

```
Quantify the accuracy of the prediction by forming time series of
Error (t) = predicted(t) –velb4(80,t)
in each of the 25 segments.
Form power spectra of
Error(! ) and velb4(80, ! ) at 50 dof. Plot log-log on the same plot.
(This is the noise-to-signal ratio)
Also, for two of your sub-records, plot velb4(80, t) and predicted(t) on
the same plot. Which aspects of the wavefield are well-predicted which
are not?
```

```
error = reshape(error, [], 1,1);
[ehat_error_real, ehat_error, error_freq_bins_real, error_freq_bins, error_delta_f
    varspec_est_multidof(error, sample_interval, dof);

[ehat_80_real, ehat_80, freq_bins_real, freq_bins, delta_f] = ...
    varspec_est_multidof(velocity_80, sample_interval, dof);

figure();
loglog(freq_bins_real, ehat_error_real, 'r');
hold on;
loglog(freq_bins_real, ehat_80_real, 'b');
legend('Error', 'Variance density of velocity data, index 80');
xlabel('Frequency, cps');
ylabel('Variance spectral density of velocity (cm/s) ^-^2');
```

```
title('Noise and Signal, 160m range');

disp('The wavefield is well-predicted in areas where there are ');
disp('high-magnitude waves (swell, wind peaks).');
```

```
The wavefield is well-predicted in areas where there are

high-magnitude waves (swell, wind peaks).
```




```
end


function [cross_spec, freq_bins] = crossspectrum(data_a, data_b, sample_interval,
    % We get 2 DOF from each set of data (thanks to real/imag components.
    num_sets = ceil(dof / 2);

    len_complete = size(data_a, 1);
    points_per = floor(len_complete / num_sets);

    delta_f = (1/sample_interval) / points_per;  % same as fs / N
    freq_bins = (points_per/2 * delta_f + delta_f):delta_f:(points_per/2 * delta_f

    % Get the variance spectral densitity for each subset.
    for i=1:num_sets
        start_idx = (i-1) * points_per + 1;
        end_idx = start_idx + points_per - 1;

        a_a(:,i) = fft(data_a(start_idx:end_idx));
```

```
        a_b(:,i) = fft(data_b(start_idx:end_idx));

        crosses(i,:) = bsxfun(@times, conj(a_a(:, i)), a_b(:,i));
    end

    % normalize FFT by n^2
    % Get the average values (average of columns)
    cross_spec = mean(crosses) ./ delta_f ./ points_per^2;
    %cross_spec = reshape(cross_spec, [], 1, 1);

    % get rid of the first (mean) value.

end
```

Variance Spectral Density Functions

Take the given data and provide a variance density estimate The freq_bins array will be in reciprocal units to the units of sample_interval. ehat has units of (data_units^2) per (1/sample_interval_units)

```
function [ehat_real, ehat, freq_bins_real, freq_bins, delta_f] = ...
    varspec_est_real(data, sample_interval)

    % First, do the FFT to get an array of Fourier coefficients
    a = fft(data);

    % Figure out our frequency bins.
    N = length(data);
    delta_f = (1/sample_interval) / N;  % same as fs / N
    freq_bins_real = 0:delta_f:(N/2 * delta_f);
    freq_bins = (-N/2 * delta_f + delta_f):delta_f:(N/2 * delta_f);
    %freq_bins = 0:delta_f:N*delta_f - delta_f;

    % We are dealing with a real signal, so we only care about positive, real
    % freqency components.  To get the power, we want to "fold" both sides of
    % the spectrum together.  Since they are the same, we can just take one
    % side and multiply by 2.
    % To preserve variance, we have to normalize this result by dividing by
    % N^2, due to the Matlab implementation of fft.
    ehat_real = 2 * abs(a(1:length(freq_bins_real))).^2 ./ delta_f ./ ...
        (N^2);

    % Also, figure out the full spectrum
    ehat = abs(a(1:length(freq_bins))).^2 ./ delta_f ./ (N^2);

    % The first frequency bin will hold the mean, not a component of the
    % variance.  Since we are interested in the variance spectrum, drop
    % the first bin.
    %ehat = ehat(2:end);
    ehat_real = ehat_real(2:end);
    freq_bins_real = freq_bins_real(2:length(ehat_real)+1);

end
```

```
function [ehat_real, ehat, freq_bins_real, freq_bins, delta_f] = ...
    varspec_est_avg(data, sample_interval, set_size, number_of_sets)

    % Get the variance spectral densitity for each subset.
    for i=1:number_of_sets
        start_idx = (i-1) * set_size + 1;
        end_idx = start_idx + set_size - 1;

        [ehat_parts_real(i, :), ehat_parts(i, :), freq_bins_real, freq_bins, delta
            varspec_est_real(data(start_idx:end_idx), sample_interval);
    end

    % form the average
    % Get the average values (average of columns)
    ehat = mean(ehat_parts);
    ehat_real = mean(ehat_parts_real);

end

function [ehat_real, ehat, freq_bins_real, freq_bins, delta_f] = ...
    varspec_est_multidof(data, sample_interval, dof)

    % We get 2 DOF from each set of data (thanks to real/imag components.
    num_sets = ceil(dof / 2);

    len_complete = size(data, 1);
    points_per = floor(len_complete / num_sets);

    [ehat_real, ehat, freq_bins_real, freq_bins, delta_f] = ...
        varspec_est_avg(data, sample_interval, points_per, num_sets);
end
```

Published with MATLAB® 7.12