preliminary stuff: get the directory we're in

and add the validation subdirectory to the path

```
cpath = which('coherence_tutorial');
[rootDir, name, ext] = fileparts(cpath);
vpath = fullfile(rootDir, 'validation');
addpath(vpath);
ppath = fullfile(rootDir, 'preprocessing');
addpath(ppath);
dataDir = fullfile(rootDir, '../data'); %contains stim/response pairs
stimsDir = fullfile(dataDir, 'all_stims'); %contains the .wav files
```

The next three sections allow you to load and visualize single unit data from

The theunissen lab. Your goals are: 1. Get familiar with this data structure and 2. Load you own data in a similar structure. For the Theunissen data you can specify a directory for three brain regions and three example neurons in each.: 'mld' is the avian auditory midbrain 'ov' is the avian auditory thalamus 'l2a' is the avian auditory cortex each region has a 'good', 'avg', and 'bad' dataset, corresponding to the signal to noise ratio, quantified by information values.

```
cellDirName = 'mld_good';
cellDir = fullfile(dataDir, cellDirName);
```

now we're going to get the stimulus and response

files from the cell directory using a function that was written to deal with this directory structure. we'll will pull stim/response files for conspecific stimuli. You should write your own data load function for your data.

```
datasets = find_datasets(cellDir, stimsDir, 'conspecific');
cellStimDir = datasets{1}.dirname;
stimFiles = datasets{1}.srPairs.stimFiles; %paths to .wav files
respFiles = datasets{1}.srPairs.respFiles; %paths to spike* files
```

now we're going to preprocess the sound stimuli by taking the

short time fourier transform, and preprocess the raw spike times into PSTHs for each stim/response pair

```
preprocDir = fullfile(cellStimDir, 'preproc'); %cache the preprocessed data here
[s,mess,messid] = mkdir(preprocDir);
preprocOptions = struct; %we'll leave this empty and use default options

srData = preprocess_sound(stimFiles, respFiles, 'ft', struct, preprocDir);
pairCount = length(srData.datasets); %# of stim/response pairs
```

visualize the stimulus/response pairs by setting showSRPairs = 1

```
showSRPairs = 0;
if showSRPairs
  %go through each pair in the dataset
  for k = 1:pairCount
```

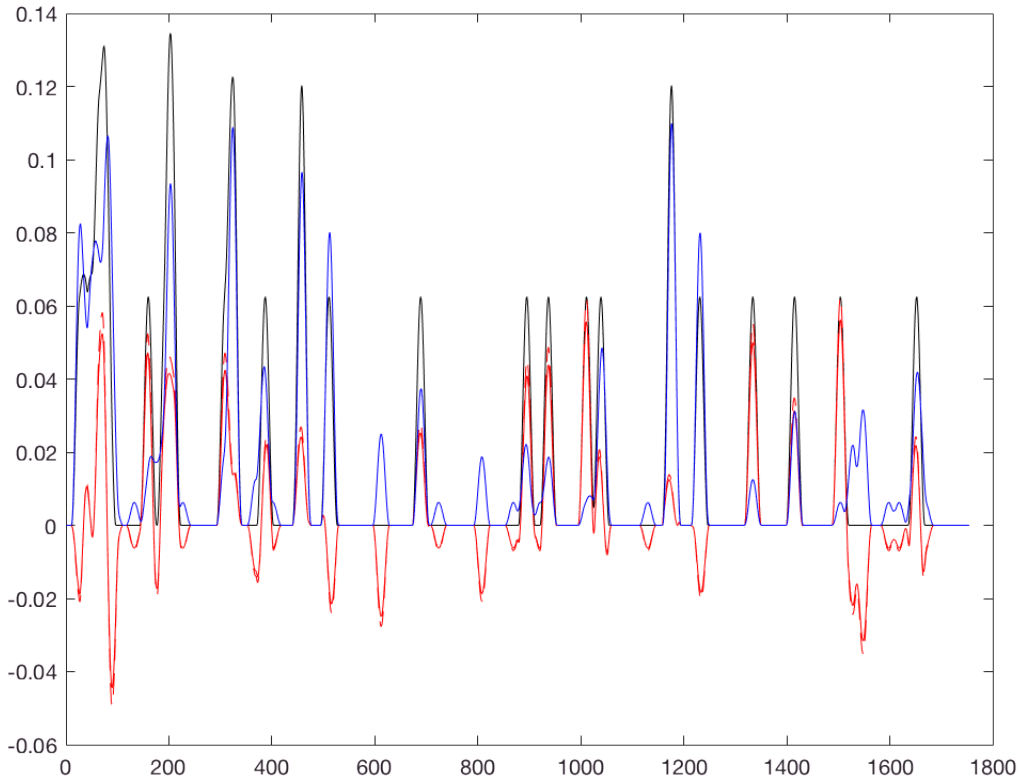
```
    ds = srData.datasets{k};
    plot_tf_resp(ds);
    end
end
```

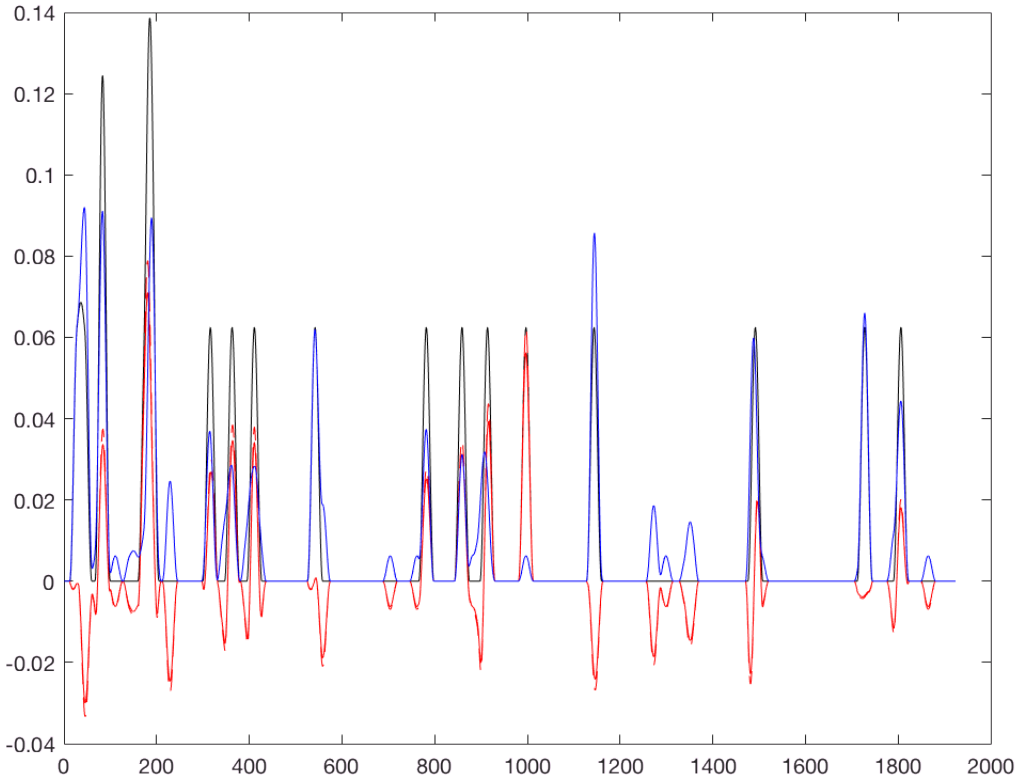
Assignment 1: Calculate the noise for each trial and display it

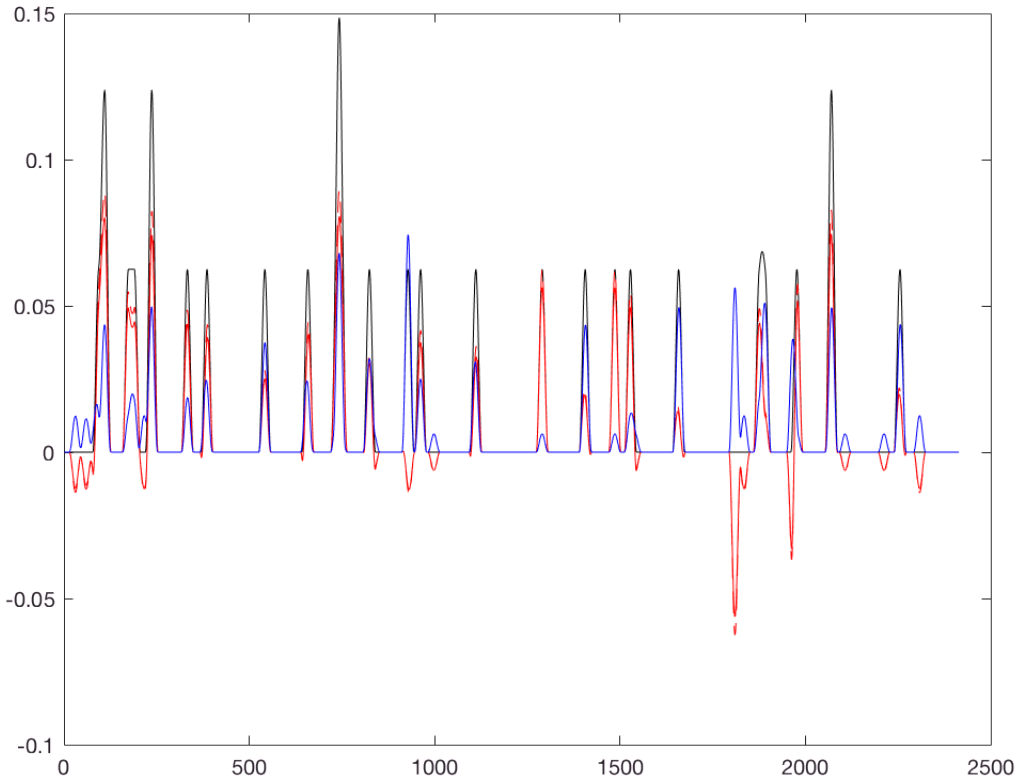
Calculate the noise using a signal generated from all trials as well as a signal that does not incldue the trial for wich you calculate the noise

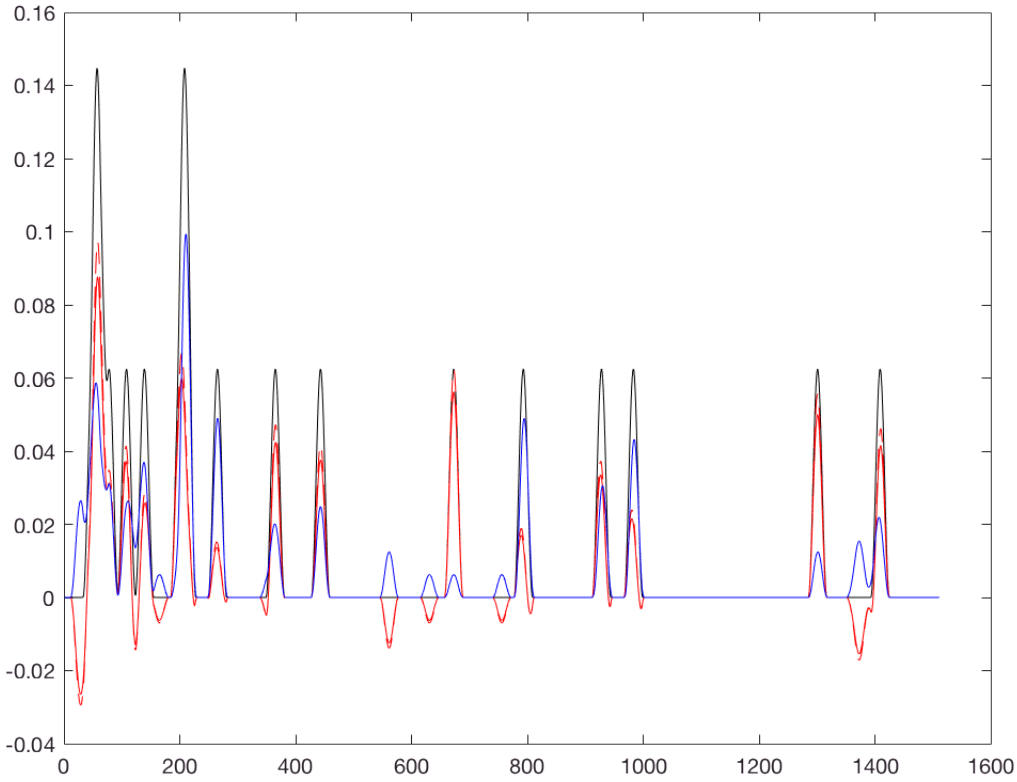
```
% These will be the noise and signal traces for all the stim-response pairs
noise_tot = [];
noise_d1_tot = [];
signal_tot = [];

wind1 = hanning(31)/sum(hanning(31));   % 31 ms smoothing for plotting only

for k=1:pairCount
    resp = srData.datasets{k}.resp;
    stim = srData.datasets{k}.stim;
    ntrial = length(resp.rawSpikeTimes);  % Number of trials for this stim-response
    stimdur = stim.stimLength*1000.0;     % Stimulus duration in ms.
    binsize = 1;                          % Sampling rate in for signal and noisems.
    ndur = round(stimdur/binsize)+1;        % Length of signal and noise for this stim-respons
    noise = zeros(ntrial, ndur);
    noise_d1 = zeros(ntrial, ndur);

% Transform spike arrival times into a time series at sampling rate binsize
    for itrial=1:ntrial

        stimes = resp.rawSpikeTimes{itrial};
        indx = ((stimes >= 0) & (stimes <= stimdur));

        stimes = stimes(indx);
        sindxs = round(stimes/binsize) + 1;
        % Prevent exceeding array dimensions (should not happen except
        % because of rounding off).
        for j = 1:length(sindxs)
            if sindxs(j) <= 0
                sindxs(j) = 1;
            end
            if sindxs(j) > ndur
                sindxs(j) = ndur;
            end
        end
        % Here is my time series - I called it noise but it is the
        % time-series for this trial
        noise(itrial, sindxs) = 1;
    end

    % Now calculate signal, noise and noise_d1 and plot the first trial
    % only (for clarity)

    figure;
    signal = mean(noise,1);   % The signal is estimated as the average of all trials
    signal_tot = [signal_tot signal];

    trials = 1:ntrial;
    for itrial=1:ntrial
```

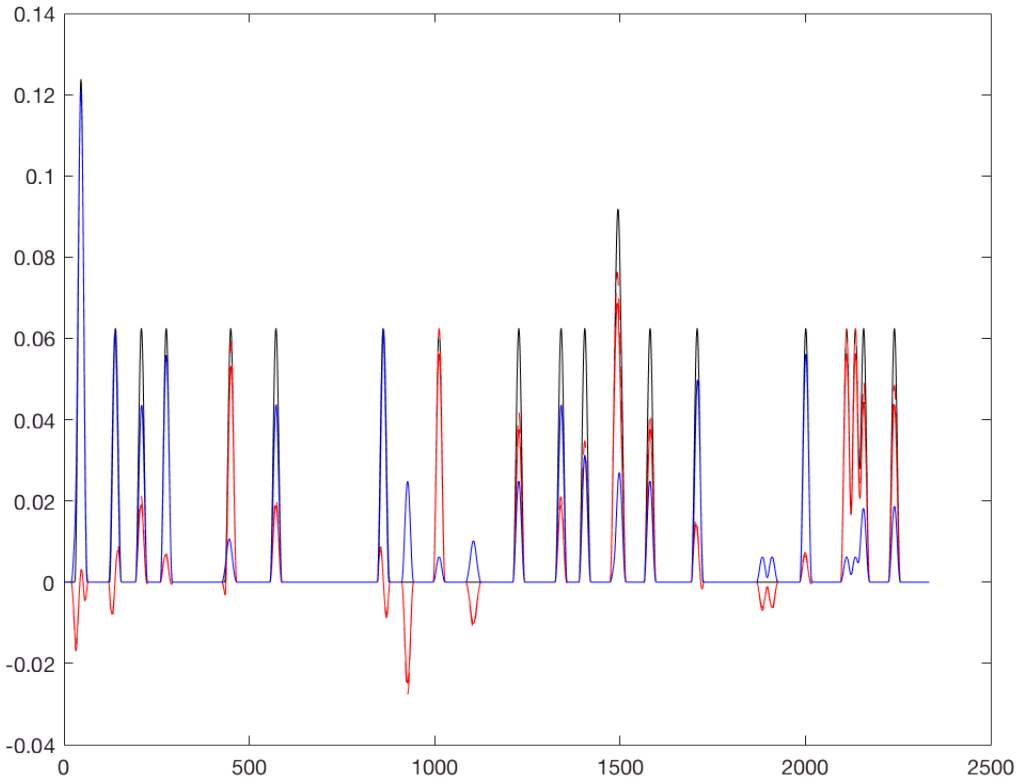
```
        % First plot the response of the first trial
        if (itrial == 1)
            plot(conv(noise(itrial,:), wind1), 'k');
            hold on;
        end

        % Calculate the two estimates of the noise
        noise_d1(itrial, :) = noise(itrial, :) - mean(noise(find(trials ~= itrial),:), 1);
        noise(itrial, :) = noise(itrial,:) - signal;
        noise_tot = [noise_tot noise(itrial, :)];
        noise_d1_tot = [noise_d1_tot noise_d1(itrial, :)];

        % Plot the noises
        if (itrial == 1)
            plot(conv(noise_d1(itrial, :), wind1), 'r--');
            plot(conv(noise(itrial,:), wind1), 'r');

            % Plot the signal
            plot(conv(signal, wind1), 'b');
            hold off;
        end
    end
end
```

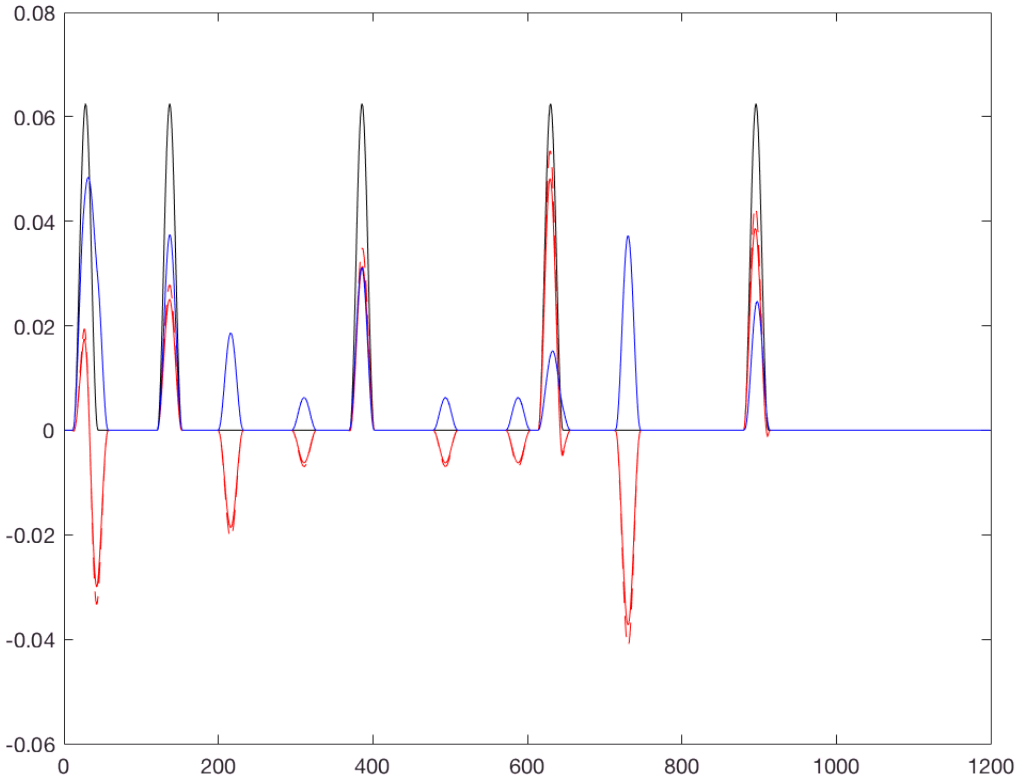


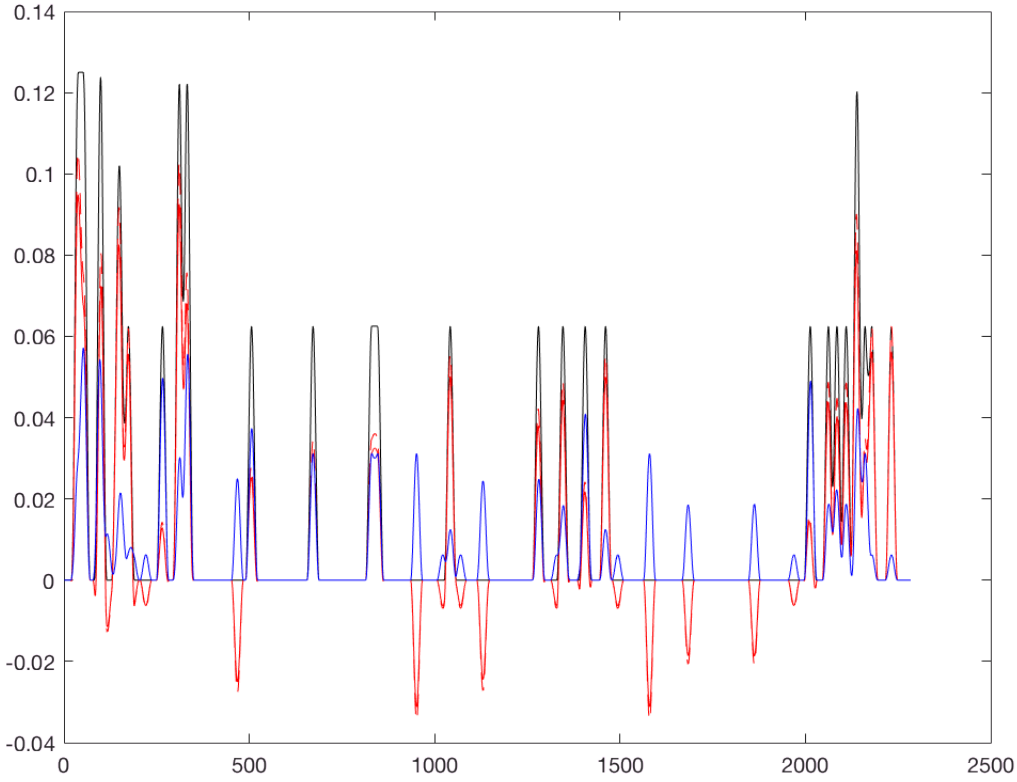

0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
0
200
400
600
800
1000
1200
1400
1600
1800
2000

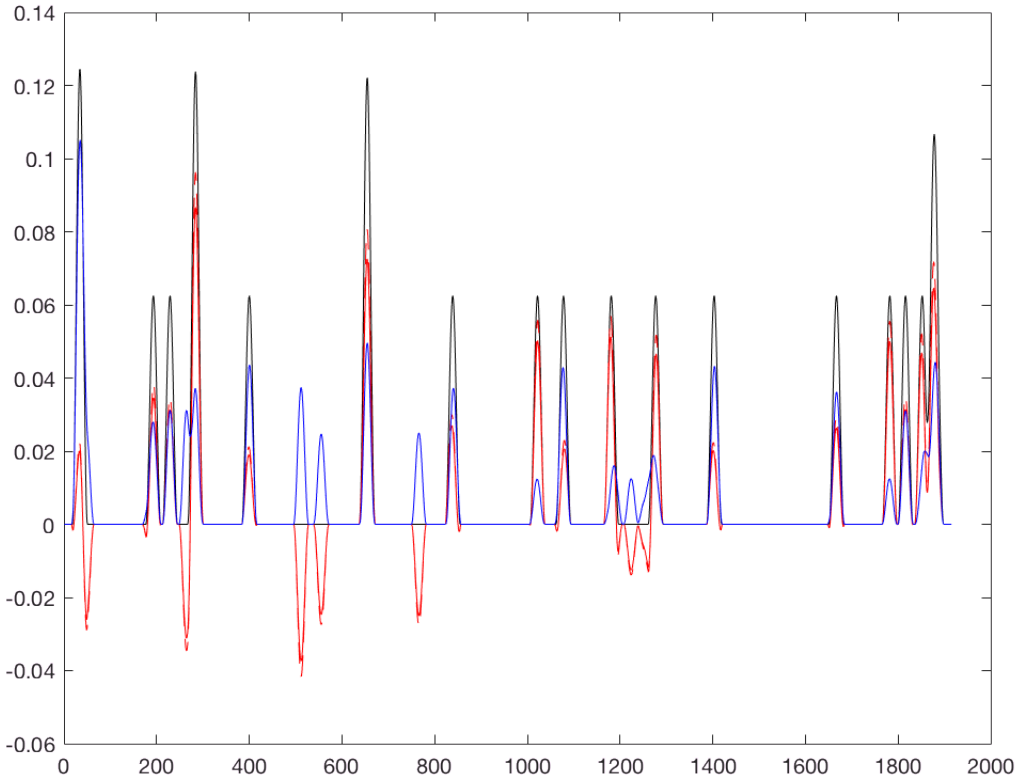

0.15
0.1
0.05
0
-0.05
-0.1
0
500
1000
1500
2000
2500

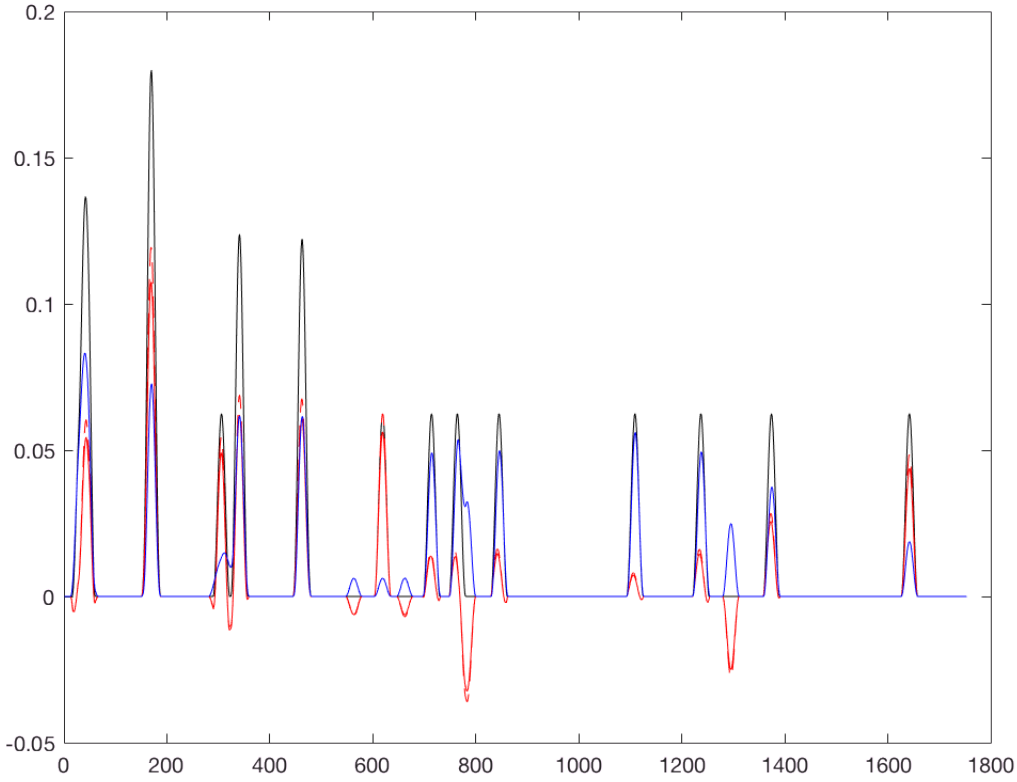

0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
0
200
400
600
800
1000
1200
1400
1600

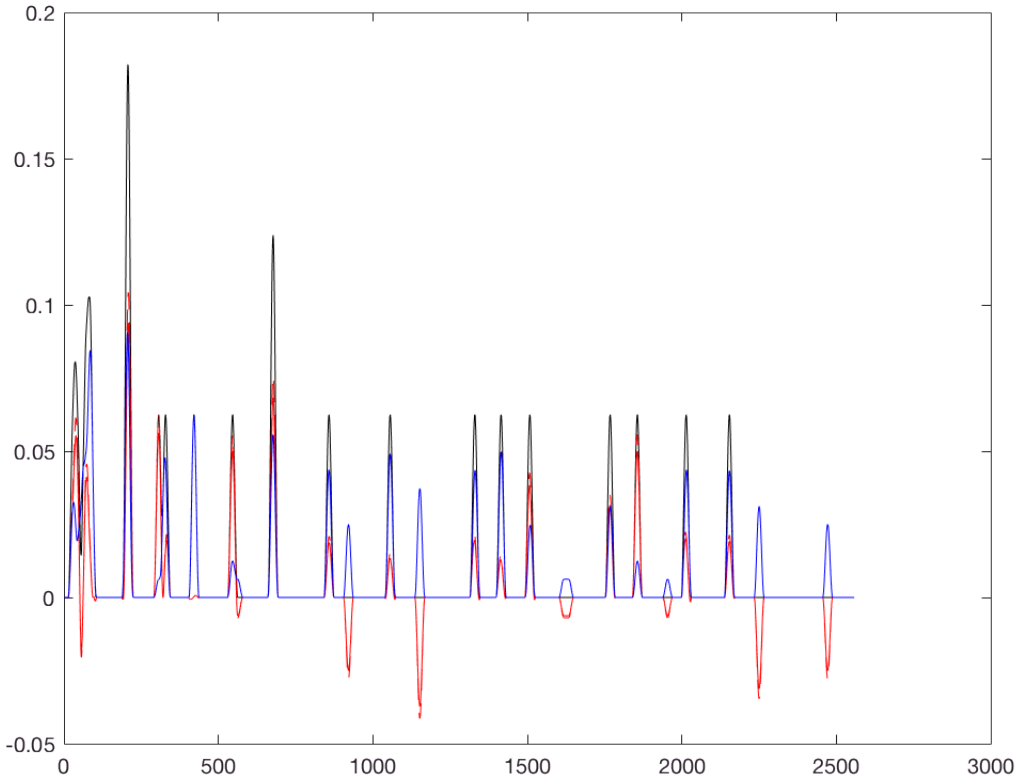

0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
0
500
1000
1500
2000
2500

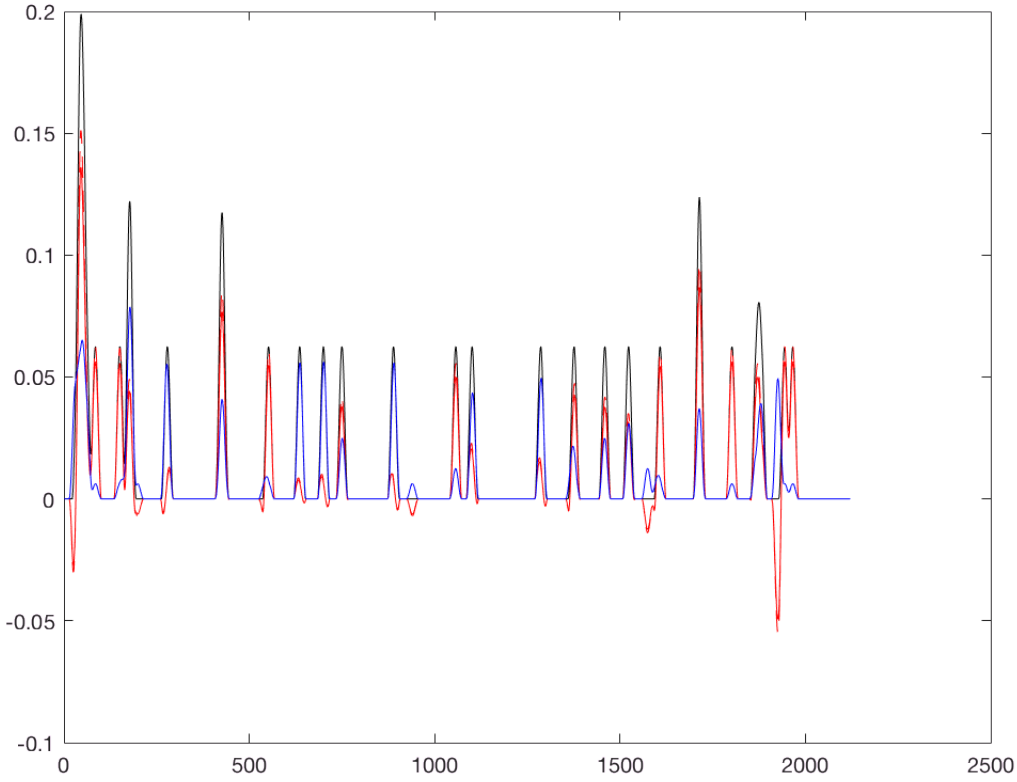

0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
0
200
400
600
800
1000
1200

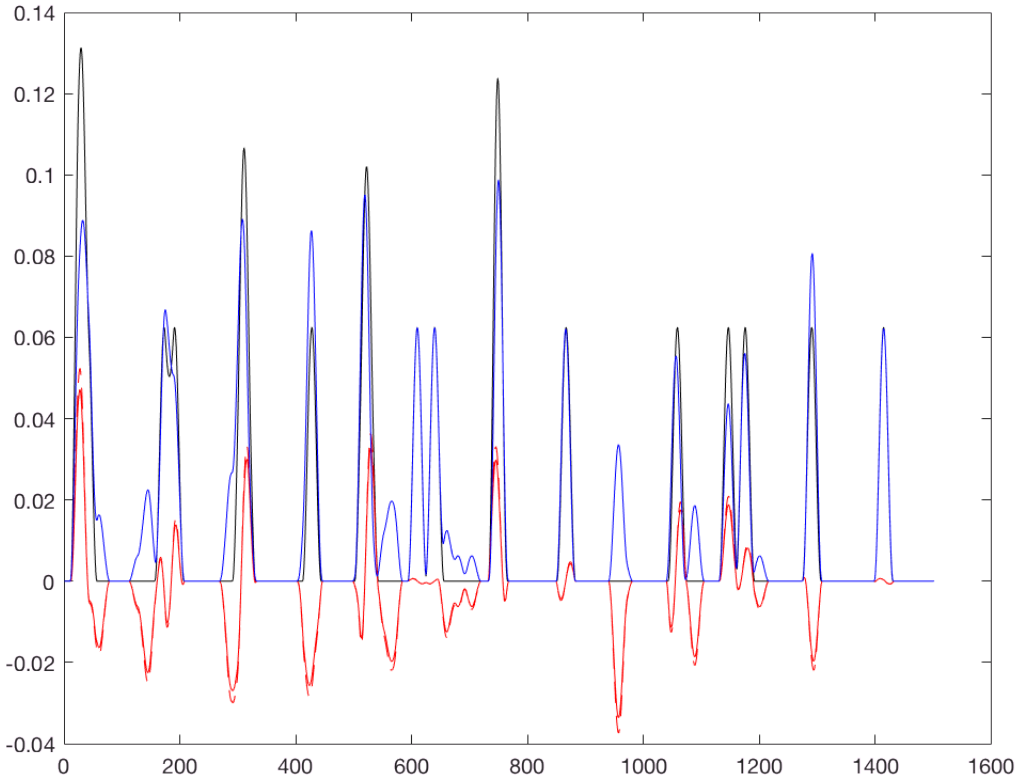

0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
0
500
1000
1500
2000
2500

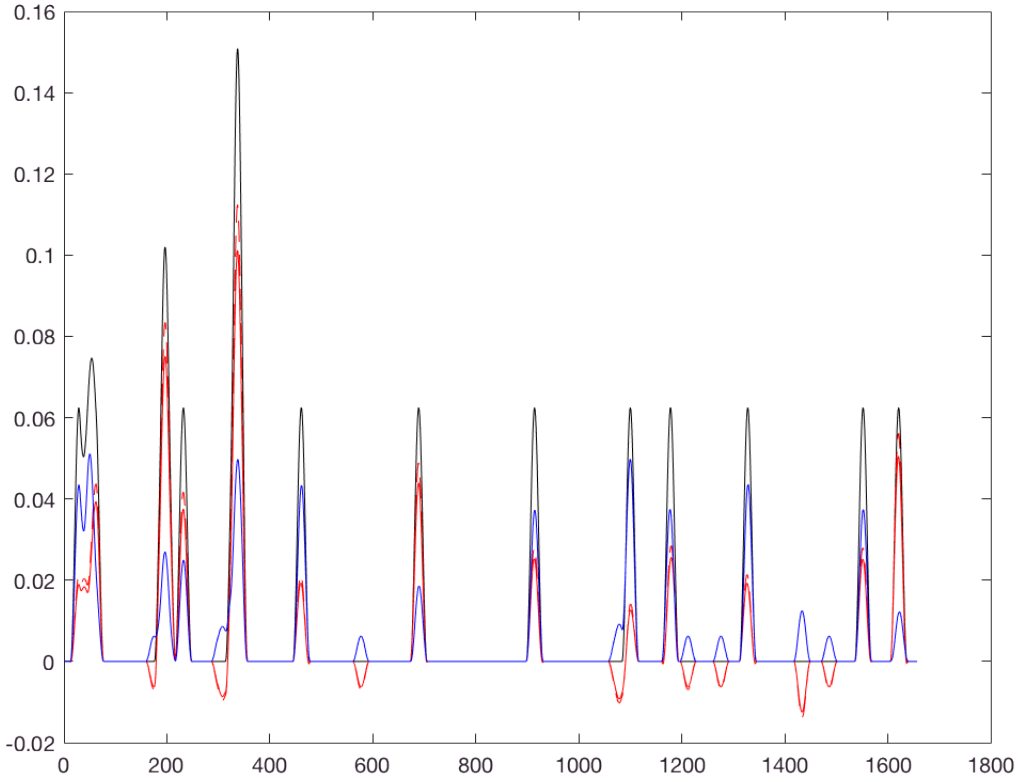

0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
0
200
400
600
800
1000
1200
1400
1600
1800
2000

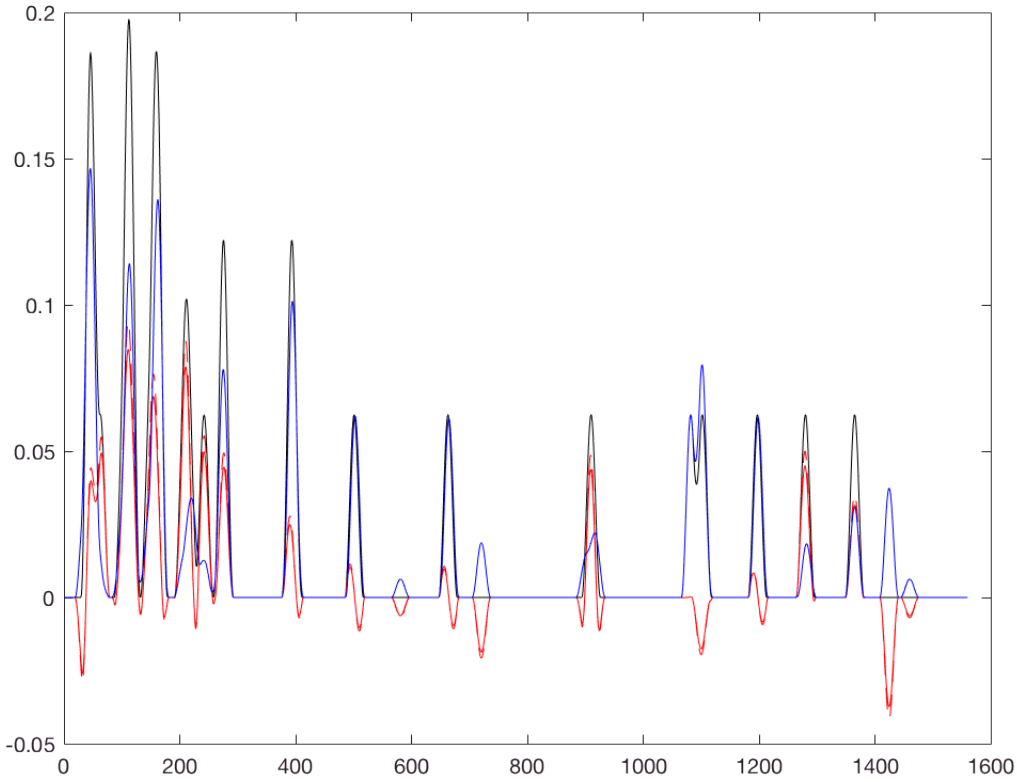

0.2
0.15
0.1
0.05
0
-0.05
0
200
400
600
800
1000
1200
1400
1600
1800

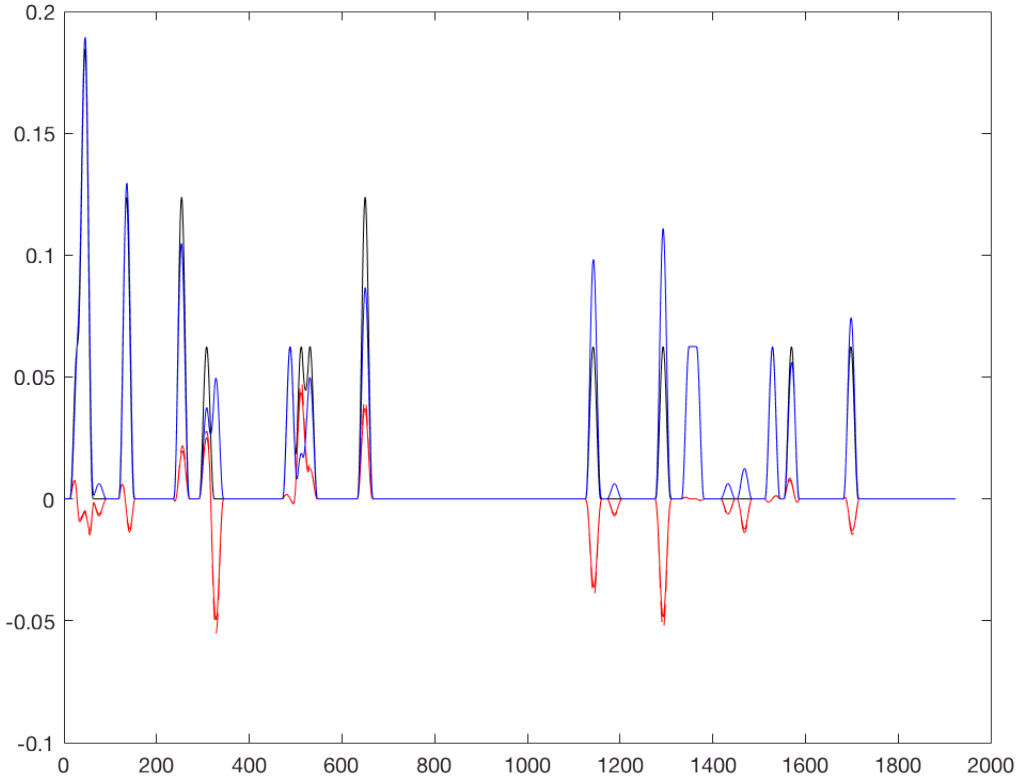

0.2
0.15
0.1
0.05
0
-0.05
0
500
1000
1500
2000
2500
3000

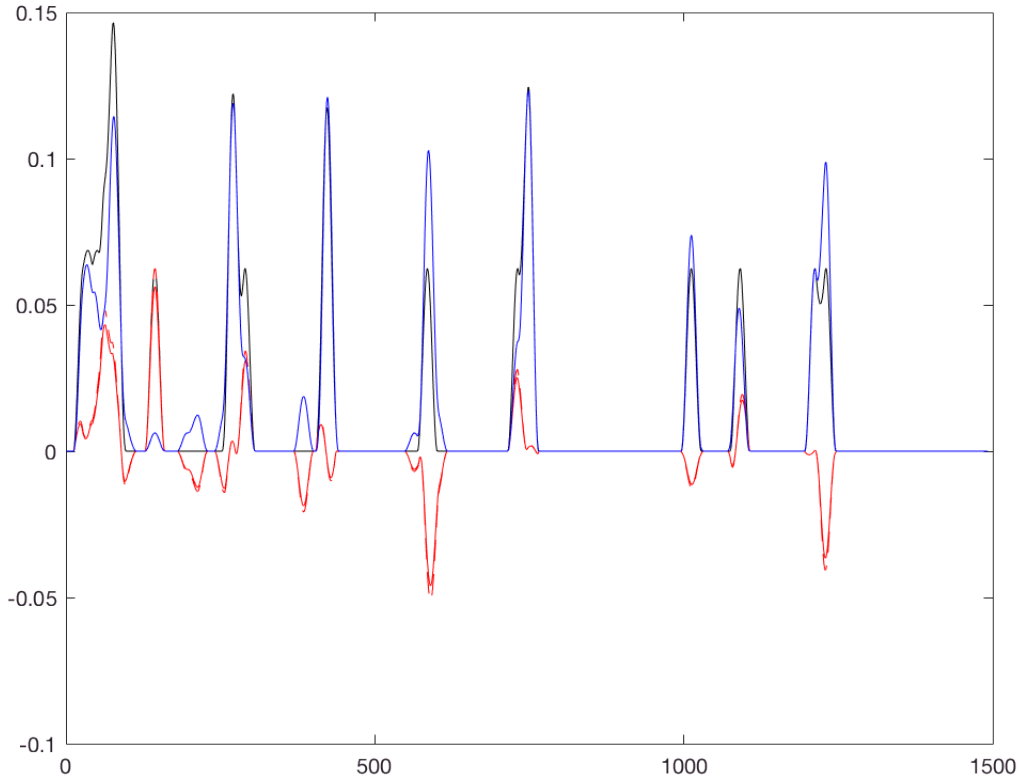

0.2
0.15
0.1
0.05
0
-0.05
-0.1
0
500
1000
1500
2000
2500

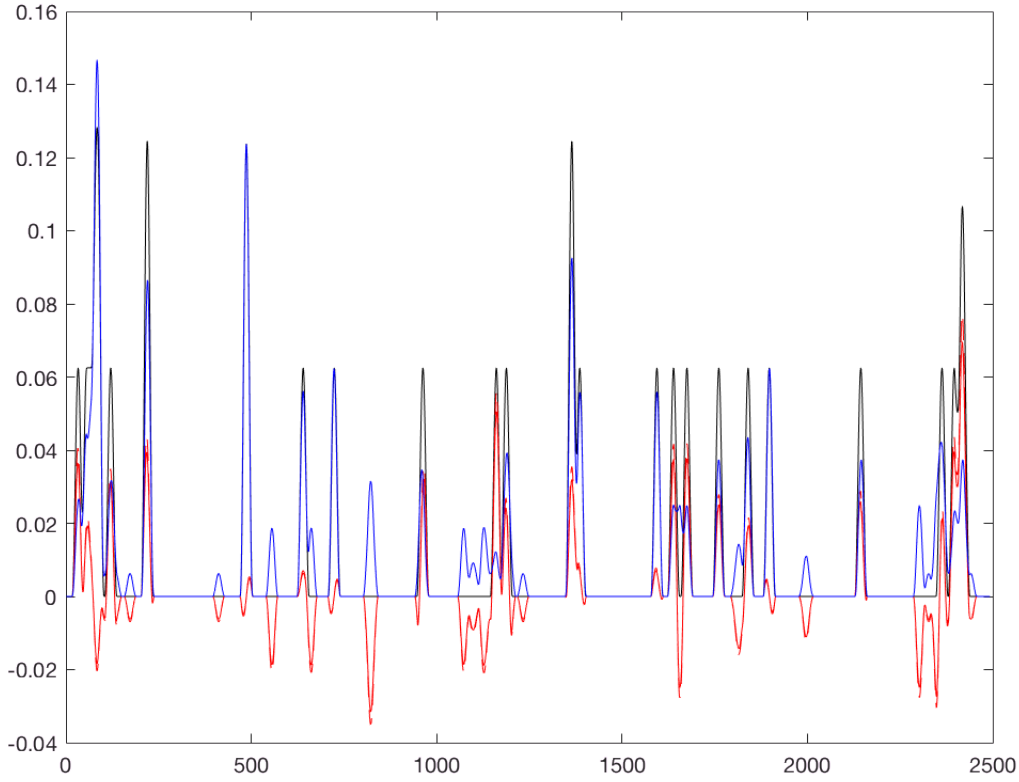

0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
0
200
400
600
800
1000
1200
1400
1600

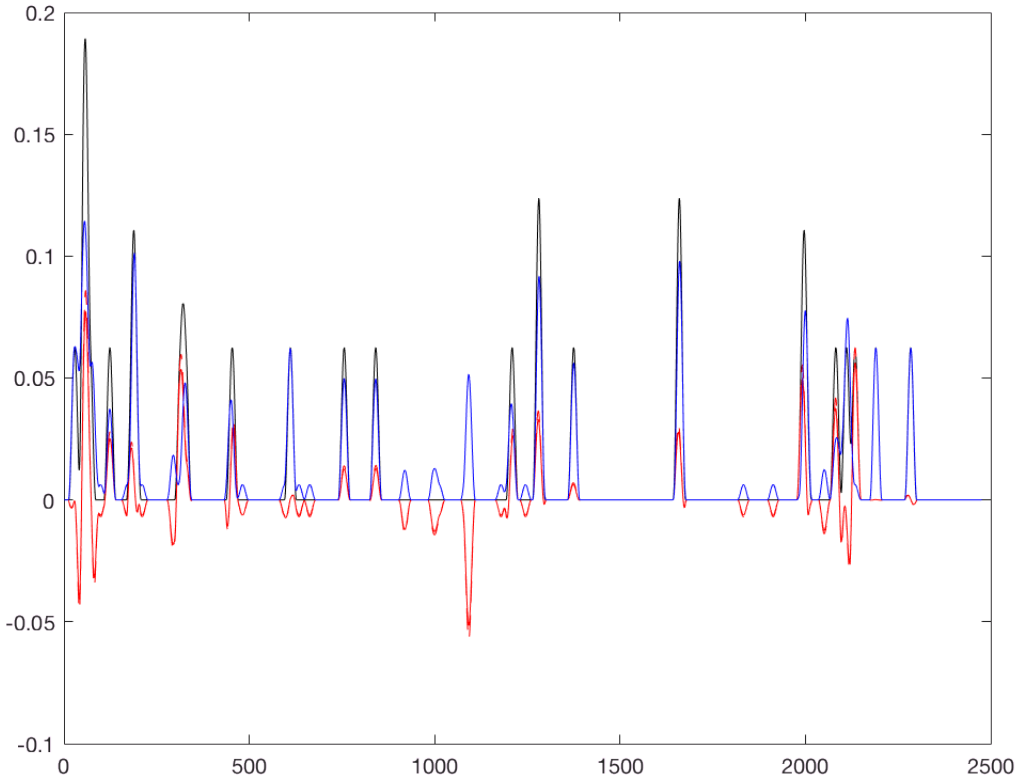

0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
0
200
400
600
800
1000
1200
1400
1600
1800

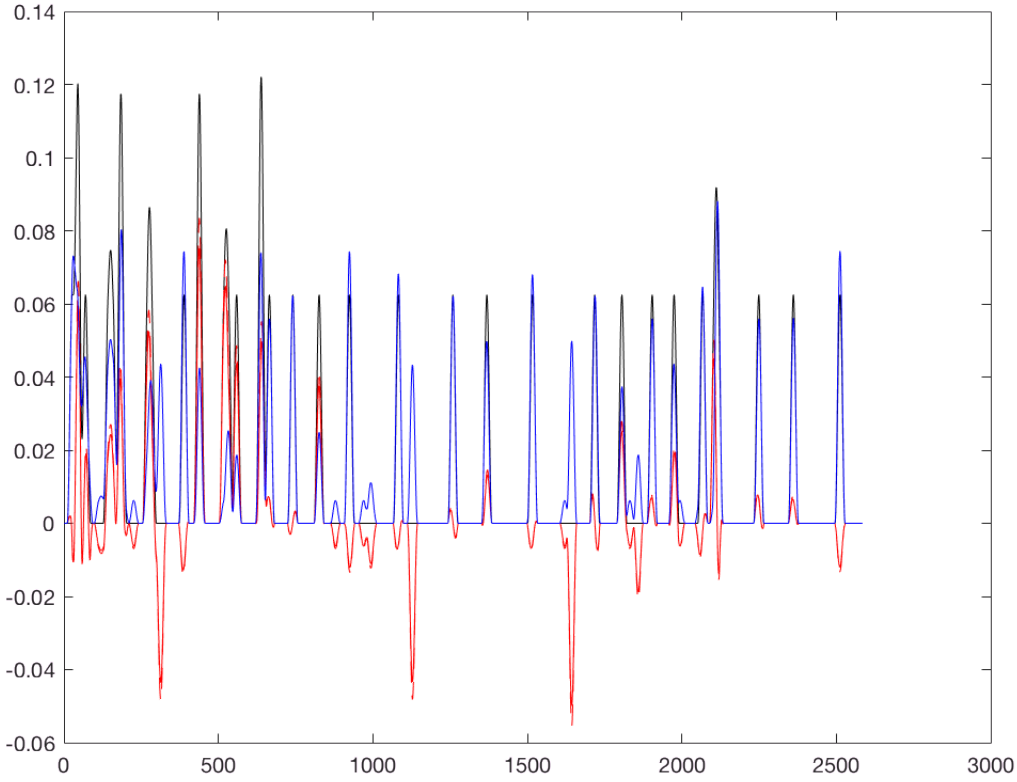

0.2
0.15
0.1
0.05
0
-0.05
0
200
400
600
800
1000
1200
1400
1600

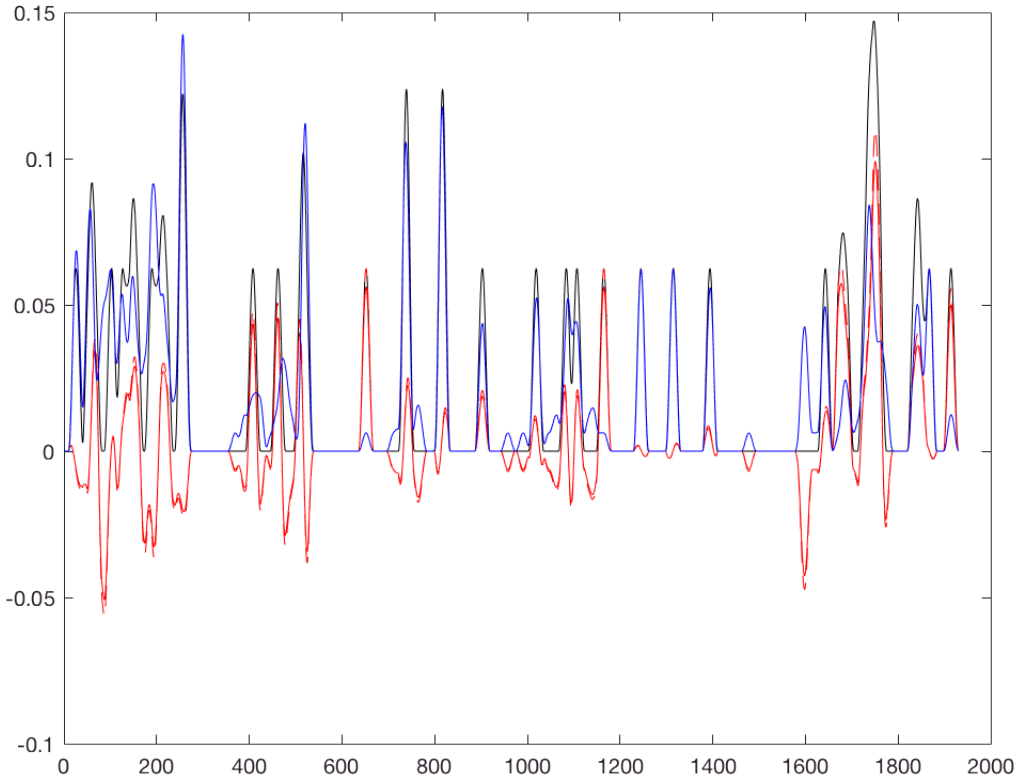

0.2
0.15
0.1
0.05
0
-0.05
-0.1
0
200
400
600
800
1000
1200
1400
1600
1800
2000


0.15
0.1
0.05
0
-0.05
-0.1
0
500
1000
1500


0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
0
500
1000
1500
2000
2500


0.2
0.15
0.1
0.05
0
-0.05
-0.1
0
500
1000
1500
2000
2500


0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
0
500
1000
1500
2000
2500
3000

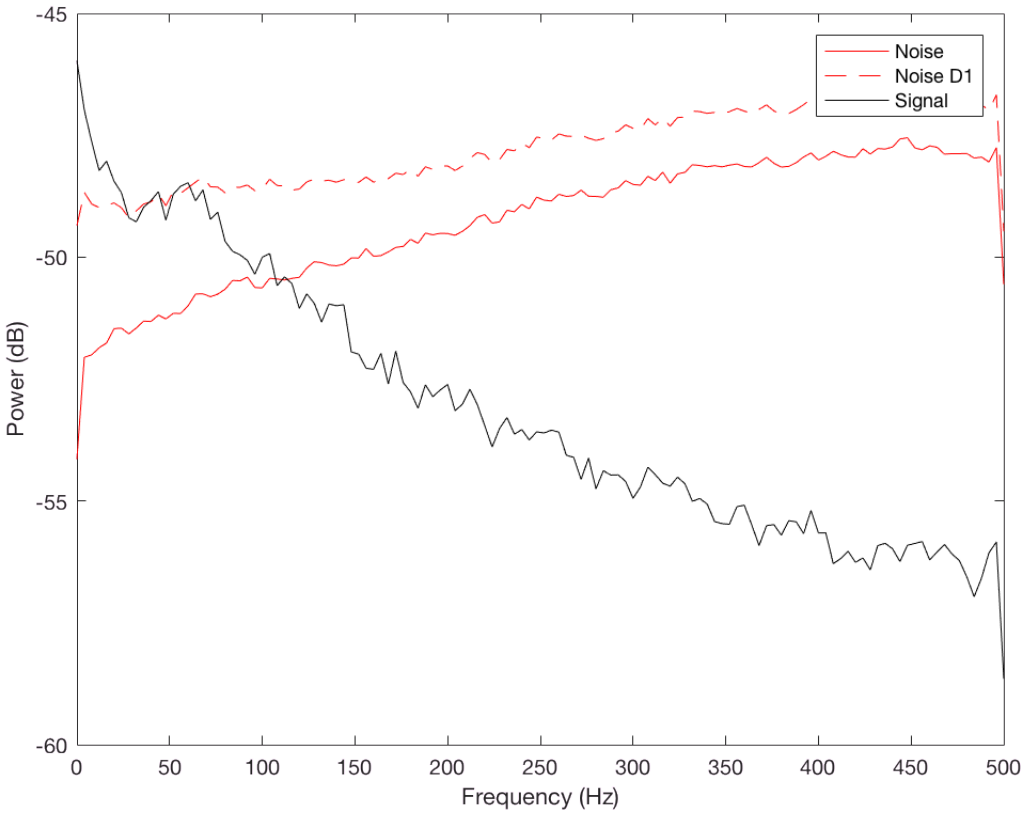


Assignment 2: Calculate the noise and signal psd obtained by averaging

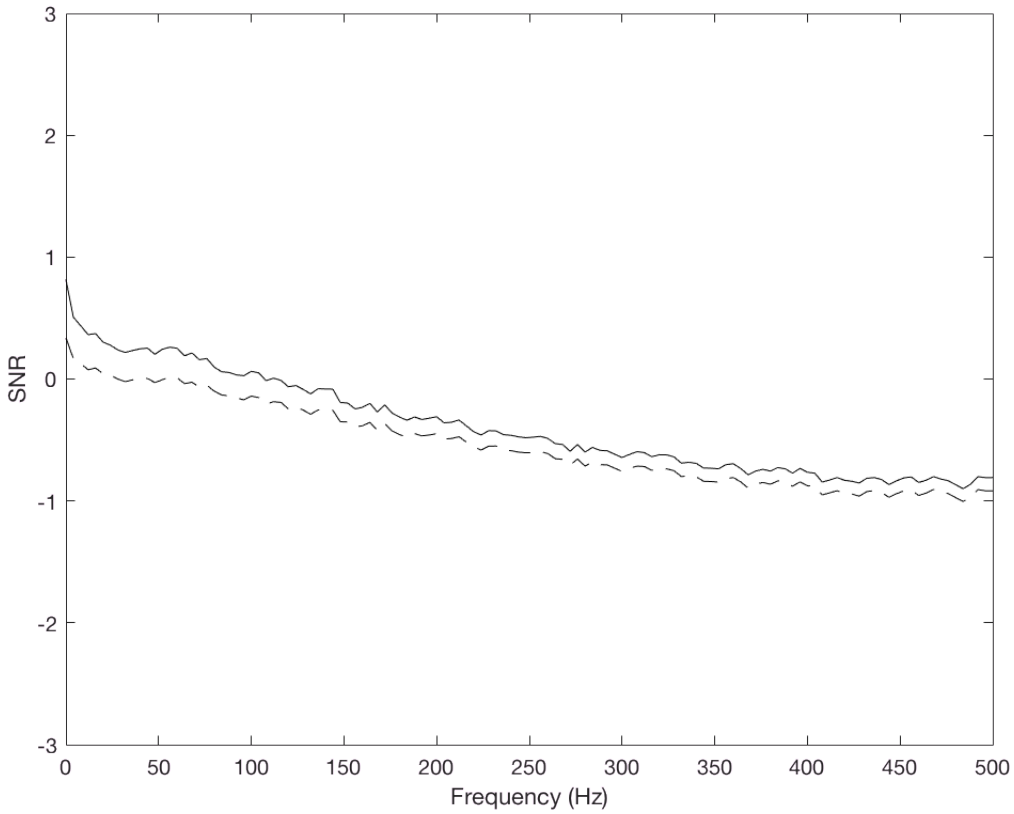
and by averaging after JNF. Is the noise white?

```
fs = 1000.0;
window = [];
nfft = 250;
 % There are many different algorithms for estimating a power spectral
 % density.  The simplest is the periodogram that divides the time series
 % into non-overlapping chunkds of size nfft (in points) and multiplies
 % that segment with the weights given by the window.  If window is null,
 % periodogram uses a rectangular window.
%[Pnoise,f] = periodogram(noise_tot,window,nfft,fs);
%[Pnoise_d1,f] = periodogram(noise_d1_tot,window,nfft,fs);
%[Psignal,f] = periodogram(signal_tot,window,nfft,fs);

% Another methods is to use overlapping chunks - this is called Welch's
% method.  Here window can be the number of points (usually equal to nfft)
% of a hamming window or a vector of weights. noverlap is the number of
% points in the overlapp. If noverlap is [], it is set to nfft/2.
% [Pxx,f] = pwelch(x,window,noverlap,nfft,fs).  You will see that using the
% hamming window gives a smoother

window = 250;
noverlap = 125;
nfft=250;
nw = 3;

%[Pnoise, f] = periodogram(noise_tot,[], window, fs);
```

```
[Pnoise,f] = pwelch(noise_tot, window,noverlap,nfft,fs);
[Pnoise_d1,f] = pwelch(noise_d1_tot, window,noverlap,nfft,fs);
[Psignal,f] = pwelch(signal_tot, window, noverlap, nfft,fs);

%[Pnoise,f] = pmtm(noise_tot, nw, nfft,fs);
%[Pnoise_d1,f] = pmtm(noise_d1_tot, nw, nfft,fs);
%[Psignal,f] = pmtm(signal_tot, nw, nfft,fs);

figure;
plot(f, 10*log10(Pnoise), 'r');
hold on;
plot(f, 10*log10(Pnoise_d1), 'r--');
plot(f, 10*log10(Psignal), 'k');
legend('Noise', 'Noise D1', 'Signal');
xlabel('Frequency (Hz)');
ylabel('Power (dB)');
hold off;
```



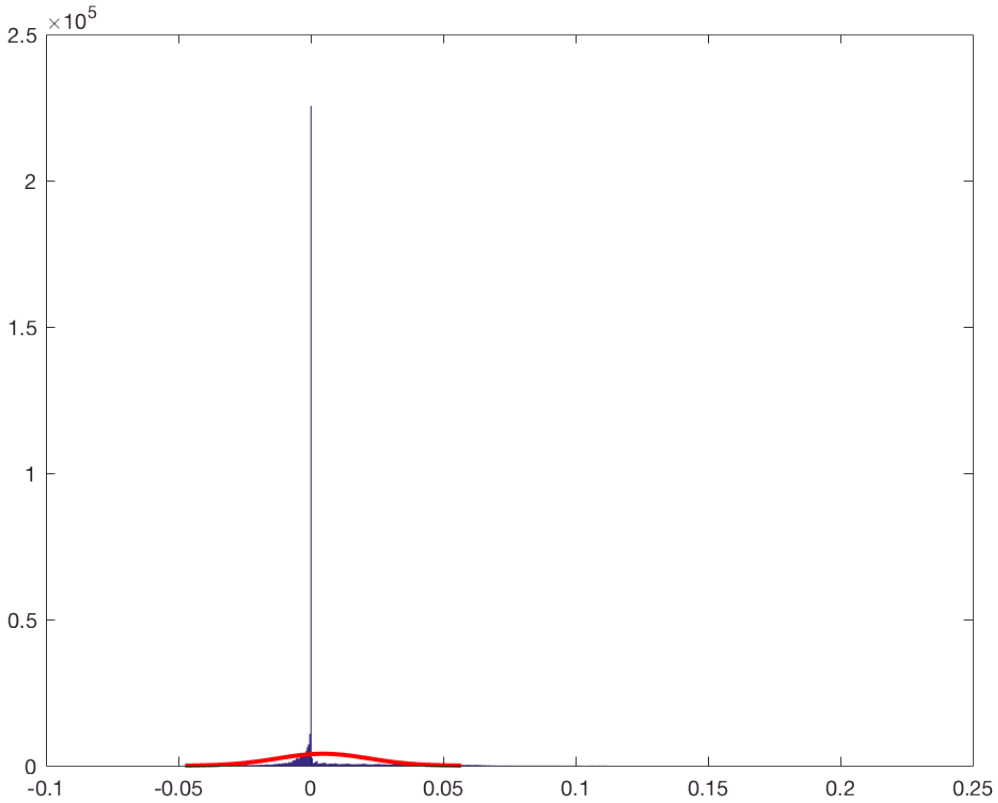

```
% Note that the noise is not white but increases with frequency.

% We can also display the signal to noise ratio.
figure;
plot(f, log10(Psignal./Pnoise), 'k');
hold on;
plot(f, log10(Psignal./Pnoise_d1), 'k--');
ylabel('SNR');
xlabel('Frequency (Hz)');
axis([0 500 -3 3]);
```



Assignment 3: Is the noise gaussian?

Matlab has a nice command called histfit that generates a histogram and displays a probability density fit. The default is a normal distribution.

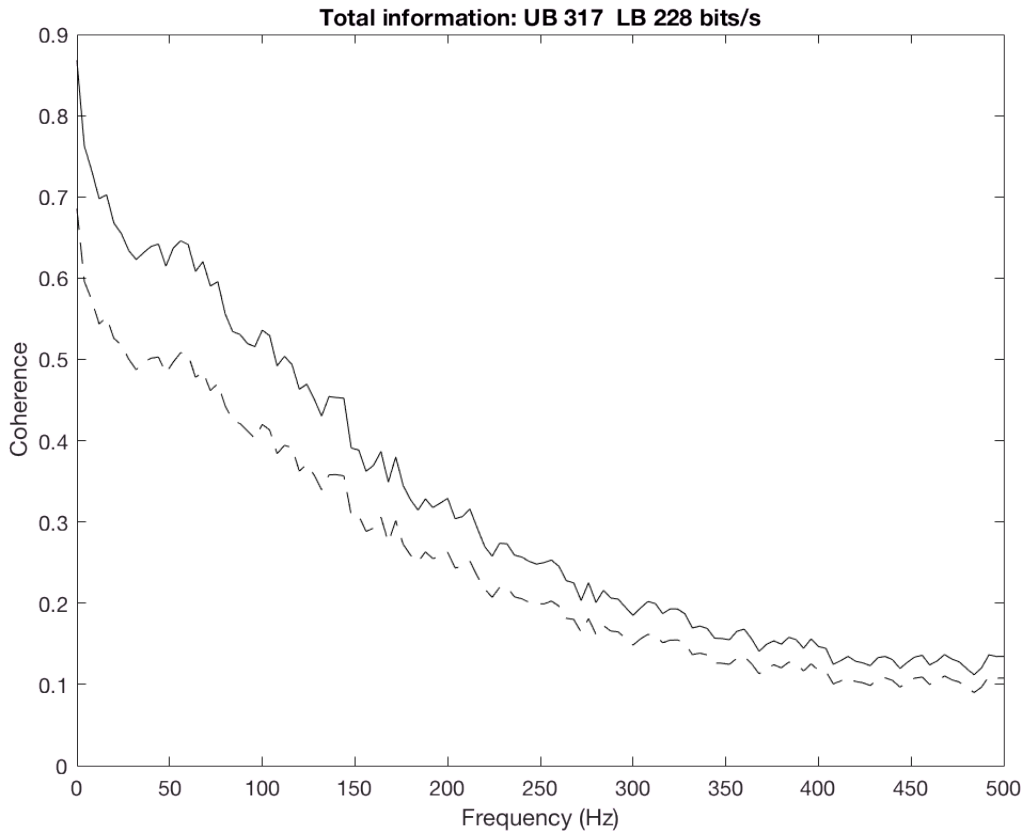
```
figure;
% histfit(noise_d1_tot);
% At 1ms resolution spiking noise is not very gaussian!  How about after
% smoothing?
histfit(conv(noise_d1_tot, wind1));
```




```
% One can see that neural noise has very high kurtosis.
```

Assignment 4: Calculate and display the coherence calculated from the signal to noise ratio.

```
% The coherence is given by
coh_snr = Psignal./(Pnoise + Psignal);
df = f(2) - f(1);
info_snr = sum(-log2(1-coh_snr))*df;
coh_snr_d1 = Psignal./(Pnoise_d1 + Psignal);
info_snr_d1 = sum(-log2(1-coh_snr_d1))*df;

figure;
plot(f, coh_snr, 'k');
hold on;
plot(f, coh_snr_d1, 'k--');
ylabel('Coherence');
xlabel('Frequency (Hz)');
title(sprintf('Total information: UB %.f  LB %.0f bits/s', info_snr, info_snr_d1));
```



We are now going to calculate the coherence and channel capacity using

Hsu, Borst, Theunissen methodology. In order to do that we need to take the raw spike times, split them into even and odd trials, and compute PSTHs for each half. there's already a function to do this, so we'll just call it.

```
[oddPsths, evenPsths] = compute_psth_halves(srData);
```

the next step is to concatenate the PSTHs across all stim/response

pairs into one big vector. we're going to do the same thing to the psth halves as well. we're also going to take note of the # of spike trials

```
psthConcat = [];
psthHalf1Concat = [];
psthHalf2Concat = [];
numStimPresentations = -1;
for k = 1:pairCount

  ds = srData.datasets{k};
  numStimPresentations = length(ds.resp.rawSpikeTimes);
  psth = ds.resp.psth;
  psthConcat = [psthConcat psth];

  psthHalf1Concat = [psthHalf1Concat oddPsths{k}];
  psthHalf2Concat = [psthHalf2Concat evenPsths{k}];
```

```
end
```

we're going to make a copy of the concatenated PSTH and corrupt

it with Gaussian noise, pretending it's a PSTH that comes from some model

```
noiseGain = 1e-1; %play with gain to increase or decrease PSTH corruption
noise = randn(size(psthConcat)) * noiseGain; %make some noise!
psthConcatNoisy = psthConcat + noise; %corrupt PSTH
psthConcatNoisy(psthConcatNoisy < 0) = 0; %rectify
psthConcatNoisy(psthConcatNoisy > 1) = 1; %rectify
```

finally, we're going to compute the upper bound of coherence, as

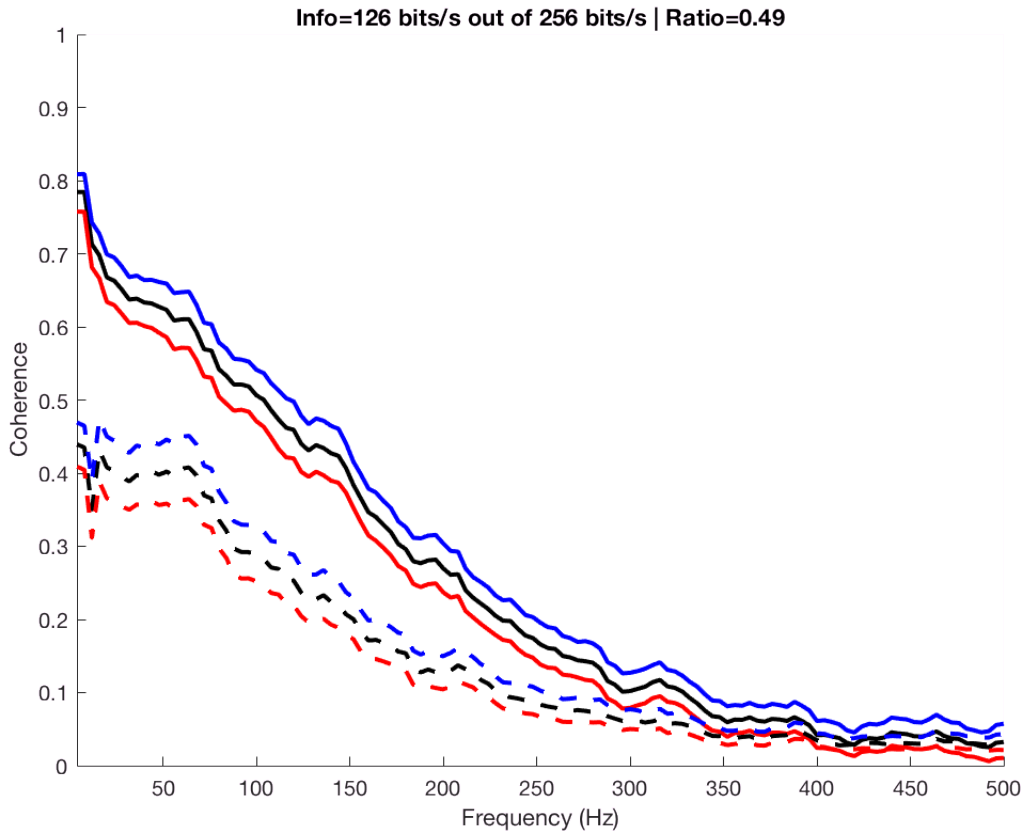
for the cell itself, as well as the coherence between the noise- corrupted PSTH and actual PSTH

```
infoFreqCutoff = -1; %max frequency in Hz to compute coherence for
infoWindowSize = 0.250; %window size in seconds to compute coherence FFT
[cBound, cModel] = compute_coherence_full(psthConcatNoisy, psthConcat, psthHalf1Concat,...
       psthHalf2Concat, srData.respSampleRate, numStimPresentations,...
       infoFreqCutoff, infoWindowSize);

performanceRatio = cModel.info / cBound.info; %how well did our noisy psth do?
```

now we'll make some plots of the coherence values, solid lines

are the upper bounds, dotted lines are noisy PSTHs

```
figure; hold on;
plot(cBound.f, cBound.c, 'k-', 'LineWidth', 2);
plot(cBound.f, cBound.cUpper, 'b-', 'LineWidth', 2);
plot(cBound.f, cBound.cLower, 'r-', 'LineWidth', 2);

plot(cModel.f, cModel.c, 'k--', 'LineWidth', 2);
plot(cModel.f, cModel.cUpper, 'b--', 'LineWidth', 2);
plot(cModel.f, cModel.cLower, 'r--', 'LineWidth', 2);
xlabel('Frequency (Hz)');
ylabel('Coherence');
theTitle = sprintf('Info=%0.0f bits/s out of %0.0f bits/s | Ratio=%0.2f', ...
     cModel.info, cBound.info, performanceRatio);
title(theTitle);
axis([min(cBound.f), max(cBound.f), 0, 1]);
```




```
%%%% Coherence with fMRI data
```

We are now going to calculate the coherence and channel capacity using

Hsu, Borst, Theunissen methodology for some fMRI voxel data.

```
% First we will load data from three voxels. The voxels are from visual
% cortex, recording during stimulation with natural movies. The 486 sec
% movie was repeated 10 times. One voxel has good signal to noise, another
% average, and the last bad signal to noise.
load ../data/vox_data.mat
```

As with the psths, average the even and odd trials, average all the

`trials, vor each voxel.`

```
vox_av_even = mean(vox_av(:,2:2:10),2);
vox_av_odd = mean(vox_av(:,1:2:10),2);
vox_av_mean = mean(vox_av,2);

vox_good_even = mean(vox_good(:,2:2:10),2);
vox_good_odd = mean(vox_good(:,1:2:10),2);
vox_good_mean = mean(vox_good,2);
```

```
vox_bad_even = mean(vox_bad(:,2:2:10),2);
vox_bad_odd = mean(vox_bad(:,1:2:10),2);
vox_bad_mean = mean(vox_bad,2);
```

Assignment. First plot the have response and one or two trials - to give you a sense of the data.

Then calculate the Noise, its spectral density (is it white) and its amplitude distribution (is it normal).

we're going to make a copy of the average response and corrupt

it with Gaussian noise, pretending it's a response that comes from some model

```
noiseGain = 1; %play with gain to increase or decrease PSTH corruption
noise = randn(size(vox_good_mean)) * noiseGain; %make some noise!
vox_good_meanNoisy = vox_good_mean + noise; %corrupt response
```

finally, we're going to compute the upper bound of coherence, as

for the voxel itself, as well as the coherence between the noise- corrupted response and actual response

```
infoFreqCutoff = -1; %max frequency in Hz to compute coherence for
infoWindowSize = 50; %window size in seconds to compute coherence FFT. The default in compute_
numStimPresentations = 10;
fmri_sampling_rate = 1;        % This BOLD signal has a 1 Hz sampling rate

[cBound, cModel] = compute_coherence_full(vox_good_meanNoisy, vox_good_mean, vox_good_even,...
       vox_good_odd, fmri_sampling_rate, numStimPresentations,...
       infoFreqCutoff, infoWindowSize);

performanceRatio = cModel.info / cBound.info; %how well did our noisy psth do?
```

now we'll make some plots of the coherence values, solid lines

are the upper bounds, dotted lines are noisy PSTHs

```
figure; hold on;
plot(cBound.f, cBound.c, 'k-', 'LineWidth', 2);
plot(cBound.f, cBound.cUpper, 'b-', 'LineWidth', 2);
plot(cBound.f, cBound.cLower, 'r-', 'LineWidth', 2);
```

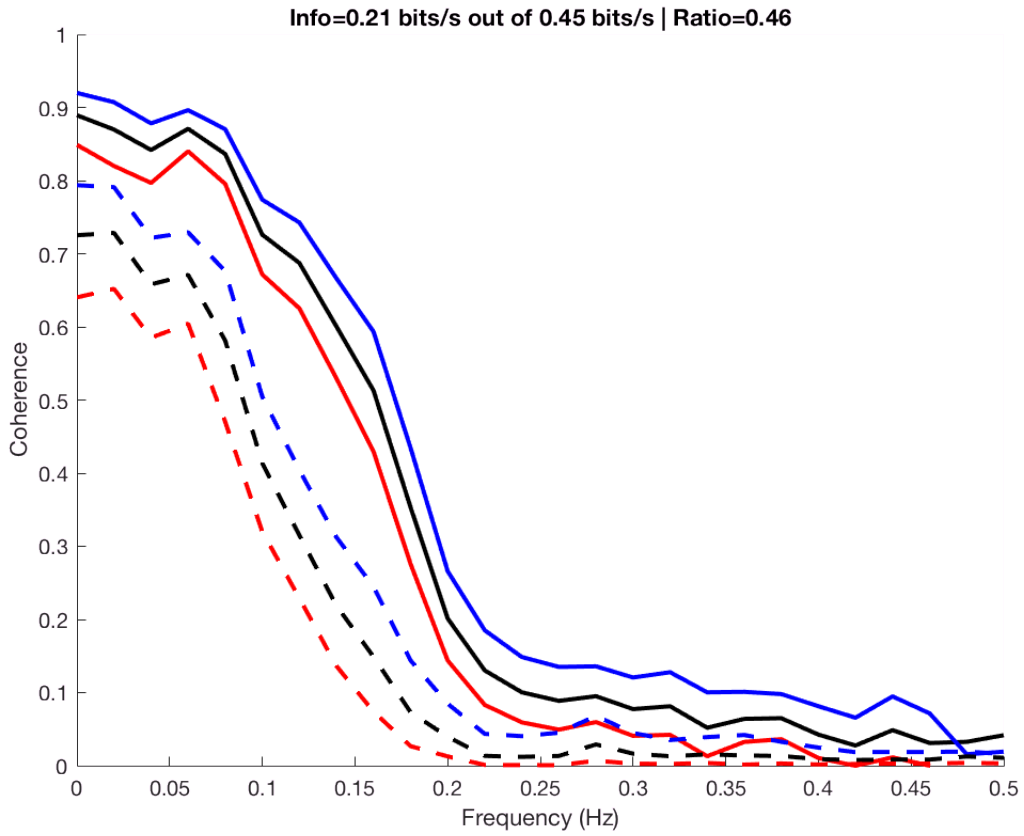
```
Warning: Imaginary parts of complex X and/or Y arguments ignored
```

```
plot(cModel.f, cModel.c, 'k--', 'LineWidth', 2);
plot(cModel.f, cModel.cUpper, 'b--', 'LineWidth', 2);
plot(cModel.f, cModel.cLower, 'r--', 'LineWidth', 2);
```

```
Warning: Imaginary parts of complex X and/or Y arguments ignored
```

```
xlabel('Frequency (Hz)');
```

```
ylabel('Coherence');
theTitle = sprintf('Info=%0.2f bits/s out of %0.2f bits/s | Ratio=%0.2f', ...
     cModel.info, cBound.info, performanceRatio);
title(theTitle);
axis([min(cBound.f), max(cBound.f), 0, 1]);
```



Now we're going to compute coherence with a response that does come from a model

of visual cortex created by the Gallant lab. The model predictions are the variables

that end in "_pred", i.e. vox_good_pred

```
infoFreqCutoff = -1; %max frequency in Hz to compute coherence for
infoWindowSize = 50; %window size in seconds to compute coherence FFT.  The default in compute
numStimPresentations = 10;
fmri_sampling_rate = 1;        % This BOLD signal has a 1 Hz sampling rate

[cBound, cModel] = compute_coherence_full(vox_good_pred, vox_good_mean, vox_good_even,...
       vox_good_odd, fmri_sampling_rate, numStimPresentations,...
       infoFreqCutoff, infoWindowSize);

performanceRatio = cModel.info / cBound.info; %how well did our noisy psth do?
```

now we'll make some plots of the coherence values, solid lines

are the upper bounds, dotted lines are noisy PSTHs

```
figure; hold on;
plot(cBound.f, cBound.c, 'k-', 'LineWidth', 2);
plot(cBound.f, cBound.cUpper, 'b-', 'LineWidth', 2);
plot(cBound.f, cBound.cLower, 'r-', 'LineWidth', 2);
```

```
Warning: Imaginary parts of complex X and/or Y arguments ignored
```

```
plot(cModel.f, cModel.c, 'k--', 'LineWidth', 2);
plot(cModel.f, cModel.cUpper, 'b--', 'LineWidth', 2);
plot(cModel.f, cModel.cLower, 'r--', 'LineWidth', 2);
```

```
Warning: Imaginary parts of complex X and/or Y arguments ignored
```

```
xlabel('Frequency (Hz)');
ylabel('Coherence');
theTitle = sprintf('Info=%0.2f bits/s out of %0.2f bits/s | Ratio=%0.2f', ...
     cModel.info, cBound.info, performanceRatio);
title(theTitle);
axis([min(cBound.f), max(cBound.f), 0, 1]);
```