Supplementary material: simulation code.

Copy and paste the following code to Matlab.

% This script simulates frequency-tagged EEG. The same data is analysed

% via spectrum (formula A), complex spectrum (formula B), and coherency (formula C and D)

clear all;

close all;

clc;

%% Generate my signal.

sampling\_rate = 1000; % In Hz.

trial\_length = 10; % In seconds

samples = sampling\_rate \* trial\_length; % How many samples are we working with?

% This is for white noise

carrier\_amplitude = 0.003; % Super weak signal. (was 0.003)

frequency = 13; % This for the carrier and is in Hz.

% Create the carrier

carrier = carrier\_amplitude \* sind(frequency\*(1:samples)\*(360/sampling\_rate)); %The divisor is to make time in seconds.

noise\_amplitude = 1; %(was 1)

birdie\_frequency\_jitter\_percent = 20; % This is frequency instability, in percent.

% This is for the birdie signal, which is an incoherent sinewave added to the noise.

birdie1\_frequency = 8; % This is in Hz.

birdie1\_amplitude = 3; % ...and this makes it a very weak signal. (Was 30)

% This birdie is disabled.

% This is an other birdie signal, that slows down from trial to trial.

birdie2\_frequency = 29; % This is in Hz.

birdie2\_amplitude = 0; % Stronger signal. Was 20.

birdie2\_frequency\_drift\_per\_second = -0.15; % This is in Hz per second.

birdie2\_frequency\_drift\_per\_trial = -0.0005; % This is in Hz per trial

%% Now organise things into trials.

% For a specific trial range, change this interval. De

trials\_start\_end = linspace(1, 2000, 50); % Trial numbers are linearly spaced.

%trials\_start\_end = logspace(0, 3.3, 50); % Log space for p-value performance plotting

% No matter what interval we set up, let's use the least number of them.

trials\_start\_end = unique(round(trials\_start\_end));

% We will use the entire trial in our FFT, so

fourier\_coeff\_we\_want = round(frequency / ((sampling\_rate/2) / (samples/2)) );

% Specify which band of the spectrum we want to plot.

plot\_band = [1, 30]; %This is in Hz.

% Sanity check

if(max(plot\_band) < frequency || min(plot\_band) > frequency)

 fprintf('Plot band is [%d, %d]Hz, whereas plot frequency is %dHz.\n', plot\_band(1), plot\_band(2), frequency);

 error('Signal frequency is outside plotting bands!')

end

if(max(plot\_band) < birdie1\_frequency || min(plot\_band) > birdie1\_frequency)

 fprintf('Plot band is [%d, %d]Hz, whereas plot frequency is %dHz.\n', plot\_band(1), plot\_band(2), birdie\_frequency);

 error('Birdie 1''s frequency is outside plotting bands!')

end

if(max(plot\_band) < birdie2\_frequency || min(plot\_band) > birdie2\_frequency)

 fprintf('Plot band is [%d, %d]Hz, whereas plot frequency is %dHz.\n', plot\_band(1), plot\_band(2), birdie\_frequency);

 error('Birdie 2''s frequency is outside plotting bands!')

end

fft\_plot\_boundaries = 1 + round(plot\_band / ((sampling\_rate/2) / (samples/2)) );

number\_of\_fourier\_coefficients = fft\_plot\_boundaries(2) - fft\_plot\_boundaries(1);

figure('Position', [0, 400, 880, 400]) %these numbers are obscure, I am assuming a 720p screen, this makes sure it fits.

fprintf('Assembling simulated experiments with the following number of trials:\n');

%% Loop-de-loop

for(i = 1:length(trials\_start\_end))

 %pause(0.5); % Slow things down so we can see it.

 %% First of all, assemble the data set.

 if(~mod(i-1, 20))

 fprintf('\n') %This is just for convenience, adds sporadic new lines to the console.

 end

 fprintf('%4d, ', trials\_start\_end(i));

 for(j = 1:trials\_start\_end(i))

 %% Generate the noise and the trial data

 white\_noise = noise\_amplitude \* (rand(1, samples) \*2) - 1; % Simple white noise.

 % We will also insert the birdie1 signal. This is going to be out of phase in each trial, but it is rather strong.

 birdie1\_frequecy\_unstable = birdie1\_frequency + birdie1\_frequency \* (rand()\*2-1) \* ((birdie\_frequency\_jitter\_percent/2) / 100); % This makes the frequency 'wobble' a bit. Like a real signal.

 birdie1 = birdie1\_amplitude \* sind(birdie1\_frequecy\_unstable\*(1:samples)\*(360/sampling\_rate) + rand(1)\*360); % The last bit adds a random phase.

 % Birdie 2 signal will show some small drift per trial.

 birdie2\_drifted\_frequencies = (birdie2\_frequency + (j-1)\*birdie2\_frequency\_drift\_per\_trial) + linspace(0, trial\_length, samples) \* birdie2\_frequency\_drift\_per\_second;

 birdie2\_drifted\_frequencies\_unstable = birdie2\_drifted\_frequencies + birdie2\_frequency \* (rand()\*2-1) \* ((birdie\_frequency\_jitter\_percent/2) / 100); % This adds an extra frequency instability

 birdie2 = birdie2\_amplitude \* sind( (birdie2\_drifted\_frequencies .\* (1:samples)\*(360/sampling\_rate) + rand(1)\*360) ); % The last bit adds a random phase.

 % Noise is everything we don't want: normal noise and interfering signals.

 noise = white\_noise + birdie1 + birdie2;

 %This is memory leaky, but it's just a small demo, so we are OK

 trial\_data(j, :) = carrier + noise; % This is the signal with white noise

 noise\_data(j, :) = noise; % You can try different types of noise here, but only white noise is implemented.

 %Also, while we are here, generate the Fourier-transform of this too!

 temp = fft(trial\_data(j, :)); %FFT, with negative frequencies left in the array.

 trial\_spectrum(j, :) = temp(2:(1+samples/2)); %We now got rid of the negative frequencies, and we get rid of the DC component.

 clear temp;

 %Re-use the temp variable for white noise, for the complex spectrum stats

 temp = fft(noise\_data(j, :)); %FFT, with negative frequencies left in the array.

 noise\_spectrum(j, :) = temp(2:(1+samples/2)); %We now got rid of the negative frequencies, and we get rid of the DC component

 clear temp;

 end

 %With our generated data, we need to look at the frequency component of interest in the FFT stuff.

 trial\_spectrum\_scalar\_means = mean(abs(trial\_spectrum), 1);

 trial\_spectrum\_vector\_means = abs(mean(trial\_spectrum, 1));

 noise\_spectrum\_scalar\_means = mean(abs(noise\_spectrum), 1);

 noise\_spectrum\_vector\_means = abs(mean(noise\_spectrum, 1));

 % Calculate coherency a'la Norcia and Tyler 1984

 norcia\_tyler\_trial\_coherency = abs( mean(trial\_spectrum, 1) ./ trial\_spectrum\_scalar\_means );

 norcia\_tyler\_noise\_coherency = abs( mean(noise\_spectrum, 1) ./ noise\_spectrum\_scalar\_means );

 % Calculate coherency: normalise the vectors

 normalised\_trial\_vectors = trial\_spectrum ./ abs(trial\_spectrum); %This is now a bunch of vectors with the length of 1.

 normalised\_noise\_vectors = noise\_spectrum ./ abs(noise\_spectrum); %This is now a bunch of vectors with the length of 1.

 % Calculate coherency: take the absolute value and mean of the normalised vectors

 zoltan\_trial\_coherency = abs(mean(normalised\_trial\_vectors, 1));

 zoltan\_noise\_coherency = abs(mean(normalised\_noise\_vectors, 1));

 %% STATS

 % We simply work out the probability based on the white noise distribution:

 % We take the sum of values that are above the one we are testing (i.e. the 'chosen value')

 % ...and divide by the length of the noise array.

 % For the purpose of performance comparison, we also save these P

 % values.

 % 1: Spectral peak versus peak noise. This is a Gaussian distribution, so we can use the t-test

 spectrum\_chosen\_value = trial\_spectrum\_scalar\_means(fourier\_coeff\_we\_want);

 spec\_p(i) = sum(noise\_spectrum\_scalar\_means >= spectrum\_chosen\_value) / length(noise\_spectrum\_scalar\_means);

 [~, spec\_p(i)] = ttest2(spectrum\_chosen\_value, noise\_spectrum\_scalar\_means, 'Tail', 'right');

 % 2: Complex spectrum versus pink noise. This doesn't really look like Gaussian,

 % but it's also not uniform distribution.

 complex\_spectrum\_chosen\_value = trial\_spectrum\_vector\_means(fourier\_coeff\_we\_want);

 [~, com\_p(i)] = ttest2(complex\_spectrum\_chosen\_value, noise\_spectrum\_vector\_means, 'Tail', 'right');

 % 3: Norcia and Tyler's coherency versus white noise distribution

 coherency\_nt\_chosen\_value = norcia\_tyler\_trial\_coherency(fourier\_coeff\_we\_want);

 cohnt\_p(i) = sum(norcia\_tyler\_noise\_coherency >= coherency\_nt\_chosen\_value) / length(norcia\_tyler\_noise\_coherency);

 % 4: Zoltan's coherency versus white noise distribution

 coherency\_chosen\_value = zoltan\_trial\_coherency(fourier\_coeff\_we\_want);

 cohy\_p(i) = sum(zoltan\_noise\_coherency >= coherency\_chosen\_value) / length(zoltan\_noise\_coherency);

 % The same but with one-tailed unpaired T-tests.

 % [~, spec\_p(i)] = ttest2(spectrum\_chosen\_value, noise\_spectrum\_scalar\_means, 'Tail', 'right');

 % [~, com\_p(i)] = ttest2(complex\_spectrum\_chosen\_value, noise\_spectrum\_vector\_means, 'Tail', 'right');

 % [~, cohnt\_p(i)] = ttest2(coherency\_nt\_chosen\_value, norcia\_tyler\_noise\_coherency, 'Tail', 'right');

 % [~, cohy\_p(i)] = ttest2(coherency\_chosen\_value, zoltan\_noise\_coherency, 'Tail', 'right');

 %% PLOT!

 % This is the spectrum

 subplot(1, 4, 1)

 plot(linspace(plot\_band(1), plot\_band(2), fft\_plot\_boundaries(2)-fft\_plot\_boundaries(1)+1), trial\_spectrum\_scalar\_means(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2)), 'r-', 'LineWidth', 2);

 xlim(plot\_band)

 ylim([min(trial\_spectrum\_scalar\_means(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2))), max(trial\_spectrum\_scalar\_means(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2))\*1.1)])

 %Add vertical line for thre freuqency marker.

 xlabel('Frequency [Hz]', 'FontSize', 14)

 ylabel('Fourier component magnitude', 'FontSize', 14)

 title(sprintf('Formula A (Spectrum):\n k=%d trials, f=%dHz', trials\_start\_end(i), frequency), 'FontSize', 12)

 set(gca, 'YScale', 'log')

 grid on;

 hold on;

 plot(linspace(plot\_band(1), plot\_band(2), fft\_plot\_boundaries(2)-fft\_plot\_boundaries(1)+1), noise\_spectrum\_scalar\_means(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2)), 'b-', 'LineWidth', 2);

 %Add text for P-value

 scatter(frequency, spectrum\_chosen\_value, 300, 'mx', 'LineWidth', 3); % Third argument is marker size

 text(frequency, max(ylim)\*0.5, sprintf('p=%.3f', spec\_p(i)), 'FontSize', 14, 'HorizontalAlignment', 'left');

 hold off;

 % This is complex spectrum

 subplot(1, 4, 2)

 plot(linspace(plot\_band(1), plot\_band(2), fft\_plot\_boundaries(2)-fft\_plot\_boundaries(1)+1), trial\_spectrum\_vector\_means(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2)), 'r-', 'LineWidth', 2);

 xlim(plot\_band)

 ylim([min(trial\_spectrum\_vector\_means(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2))), max(trial\_spectrum\_vector\_means(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2))\*1.5)])

 xlabel('Frequency [Hz]', 'FontSize', 14)

 ylabel('Fourier component magnitude', 'FontSize', 14)

 title(sprintf('Form. B (Complex spectrum):\n k=%d trials, f=%dHz', trials\_start\_end(i), frequency), 'FontSize', 12);

 set(gca, 'YScale', 'log')

 grid on;

 hold on;

 plot(linspace(plot\_band(1), plot\_band(2), fft\_plot\_boundaries(2)-fft\_plot\_boundaries(1)+1), noise\_spectrum\_vector\_means(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2)), 'b-', 'LineWidth', 2);

 %Add text for p-value

 scatter(frequency, complex\_spectrum\_chosen\_value, 300, 'mx', 'LineWidth', 3); % Third argument is marker size

 text(frequency, max(ylim)\*0.5, sprintf('p=%.3f', com\_p(i)), 'FontSize', 14, 'HorizontalAlignment', 'left');

 hold off;

 % This is Norcia and Tyler's coherency.

 subplot(1, 4, 3)

 plot(linspace(plot\_band(1), plot\_band(2), fft\_plot\_boundaries(2)-fft\_plot\_boundaries(1)+1), norcia\_tyler\_trial\_coherency(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2)), 'r-', 'LineWidth', 2);

 title(sprintf('Form. C (Coherency 1):\n k=%d trials, f=%dHz', trials\_start\_end(i), frequency), 'FontSize', 12)

 xlim(plot\_band)

 ylim([0 0.6])

 xlabel('Frequency [Hz]', 'FontSize', 14)

 ylabel('Coherency value', 'FontSize', 14)

 grid on;

 %Add text for p-value

 hold on;

 plot(linspace(plot\_band(1), plot\_band(2), fft\_plot\_boundaries(2)-fft\_plot\_boundaries(1)+1), norcia\_tyler\_noise\_coherency(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2)), 'b-', 'LineWidth', 2);

 scatter(frequency, coherency\_nt\_chosen\_value, 300, 'mx', 'LineWidth', 3); % Third argument is marker size

 text(frequency, max(ylim)\*0.8, sprintf('p=%.3f', cohnt\_p(i)), 'FontSize', 14, 'HorizontalAlignment', 'left');

 hold off;

 % This is my coherency

 subplot(1, 4, 4)

 plot(linspace(plot\_band(1), plot\_band(2), fft\_plot\_boundaries(2)-fft\_plot\_boundaries(1)+1), zoltan\_trial\_coherency(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2)), 'r-', 'LineWidth', 2);

 title(sprintf('Form. D (Coherency 2):\n k=%d trials, f=%dHz', trials\_start\_end(i), frequency), 'FontSize', 12)

 xlim(plot\_band)

 ylim([0 0.6])

 xlabel('Frequency [Hz]', 'FontSize', 14)

 ylabel('Coherency value', 'FontSize', 14)

 grid on;

 %Add text for p-value

 hold on;

 plot(linspace(plot\_band(1), plot\_band(2), fft\_plot\_boundaries(2)-fft\_plot\_boundaries(1)+1), zoltan\_noise\_coherency(fft\_plot\_boundaries(1):fft\_plot\_boundaries(2)), 'b-', 'LineWidth', 2);

 scatter(frequency, coherency\_chosen\_value, 300, 'mx', 'LineWidth', 3); % Third argument is marker size

 text(frequency, max(ylim)\*0.8, sprintf('p=%.3f', cohy\_p(i)), 'FontSize', 14, 'HorizontalAlignment', 'left');

 hold off;

 %Update the figure as plotting happens

 drawnow;

end

% Clean up some variables to save memory. I only have 8 GB at the moment.

clearvars noise\_data trial\_data noise\_spectrum trial\_spectrum normalised\_noise\_vectors normalised\_trial\_vectors

fprintf('\nDone! Now plotting:\n'); % add new line at the end.

%% Plot p-value performance

% For Matlab's curve fitting tool, you can use:

% trials\_start\_end as your X values

% spec\_p for the performance of spectrum (formula 1)

% com\_p for the performance of complex spectrum (formula 2)

% cohnt\_p for the performance of coherency 1 (formula 3)

% cohy\_p for the performance of cohy\_p (formula 4)

% This plot is noisy, it's commented out, but it shows how much better

% coherency is when compared to spectrum.

% For convenience, we are reshaping the vectors so Matlab's curve fits would work

trials\_start\_end = reshape(trials\_start\_end, length(trials\_start\_end), 1);

spec\_p = reshape(spec\_p, length(trials\_start\_end), 1); % Spectrum (Formula 1)

com\_p = reshape(com\_p, length(trials\_start\_end), 1); % Complex spectrum (Formula 2)

cohnt\_p = reshape(cohnt\_p, length(trials\_start\_end), 1); % NT coherency(Formula 3)

cohy\_p = reshape(cohy\_p, length(trials\_start\_end), 1); % ZD coherency(Formula 3)

formula\_to\_use = 'exp1'; % This should be complementary error function-based, but close enough.

figure;

line([trials\_start\_end(1), trials\_start\_end(end)], [0.05, 0.05], 'Linewidth', 4, 'Color', '#AAAAAA') % Significance line, gray.

hold on;

% Spectrum (Formula 1)

scatter(trials\_start\_end, spec\_p, 30, 'MarkerEdgeColor', '#0072BD') % Show data points

[spec\_fit\_model, spec\_fit\_quality, ~] = fit(trials\_start\_end, spec\_p, formula\_to\_use); % Create a fitting model, but this time it's bad.

spec\_line = spec\_fit\_model(trials\_start\_end); % Create the fitted line.

plot(trials\_start\_end, spec\_line, 'LineWidth', 3, 'Color', '#0072BD');

% Complex spectrum (Formula 2)

scatter(trials\_start\_end, com\_p, 30, '\*', 'MarkerEdgeColor', '#D95319'); % Show data points

[com\_fit\_model, com\_fit\_quality, ~] = fit(trials\_start\_end, com\_p, formula\_to\_use); % Create a fitting model and evaluate it

com\_line = com\_fit\_model(trials\_start\_end); % Create the fitted line.

plot(trials\_start\_end, com\_line, 'LineWidth', 3, 'Color', '#D95319');

fprintf('r^2 value of complex spectrum (formula 2)''s fitted curve is %0.3f\n', com\_fit\_quality.rsquare);

% NT coherency (Formula 3)

scatter(trials\_start\_end, cohnt\_p, 30, 'd', 'MarkerEdgeColor', '#EDB120'); % Show data points

 % Sometimes, if there are a lot of zeros in the data, the fitting

 % algorithm fails, In this case, we can exclude zeros.

%[cohnt\_fit\_model, cohnt\_fit\_quality, ~] = fit(trials\_start\_end, cohnt\_p, formula\_to\_use, 'exclude', cohnt\_p == 0); % Create a fitting model and evaluate it

[cohnt\_fit\_model, cohnt\_fit\_quality, ~] = fit(trials\_start\_end, cohnt\_p, formula\_to\_use); % Create a fitting model and evaluate it

cohnt\_line = cohnt\_fit\_model(trials\_start\_end); % Create the fitted line, all data

%cohnt\_line = movmean(cohnt\_p, 1);

plot(trials\_start\_end, cohnt\_line, 'LineWidth', 3, 'Color', '#EDB120');

fprintf('r^2 value of coherency 1 (formula 3)''s fitted curve is %0.3f\n', cohnt\_fit\_quality.rsquare);

% ZD Coherency (Formula 4)

scatter(trials\_start\_end, cohy\_p, 30, 'x', 'MarkerEdgeColor', '#77AC30'); % Show data points

 % Sometimes, if there are a lot of zeros in the data, the fitting

 % algorithm fails, In this case, we can exclude zeros.

%[cohy\_fit\_model, cohy\_fit\_quality, ~] = fit(trials\_start\_end, cohy\_p, formula\_to\_use, 'exclude', cohy\_p == 0); % Create a fitting model and evaluate it

[cohy\_fit\_model, cohy\_fit\_quality, ~] = fit(trials\_start\_end, cohy\_p, formula\_to\_use); % Create a fitting model and evaluate it

cohy\_line = cohy\_fit\_model(trials\_start\_end); % Create the fitted line,

%cohy\_line = movmean(cohy\_p, 5);

plot(trials\_start\_end, cohy\_line, 'LineWidth', 3, 'Color', '#77AC30');

fprintf('r^2 value of coherency 2 (formula 4)''s fitted curve is %0.3f\n', cohy\_fit\_quality.rsquare);

xlim([trials\_start\_end(1), trials\_start\_end(end)])

ylim([1/samples, 1]); % If we use log axes, this can't be zero.

set(gca, 'YScale', 'log');

%set(gca, 'XScale', 'log');

grid on;

set(gca, 'FontSize', 20);

legend('p=0.05 signifiance line', ...

 'Form. A (Spectrum): p-values', sprintf('Form. A''s line'), ...

 'Form. B (Complex spectrum): p-values', sprintf('Form. B''s fitted curve, r^{2} = %0.2f', com\_fit\_quality.rsquare), ...

 'Form. C (Coherency 1): p-values', sprintf('Form. C''s fitted curve, r^{2} = %0.2f', cohnt\_fit\_quality.rsquare), ...

 'Form. D (Coherency 2): p-values', sprintf('Form. D''s fitted curve, r^{2} = %0.2f', cohy\_fit\_quality.rsquare), ...

 'FontSize', 14, 'Location', 'south');

xlabel('Number of trials');

ylabel('p-value');

title('Performance of metrics');

%save('all\_trials\_done.mat');

fprintf('All done.\n')