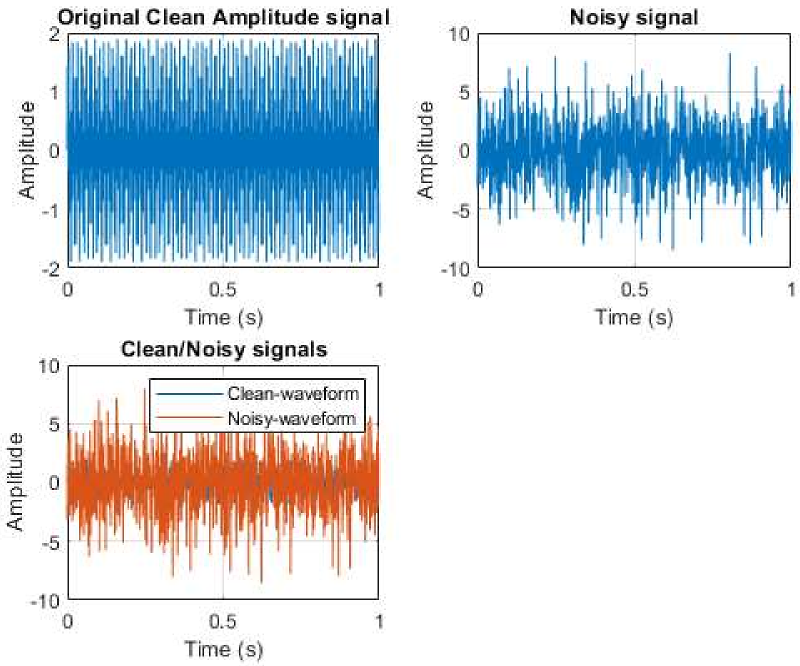
## Contents



## Adding 2 sinusoid clean signals & corrupt it with random noise

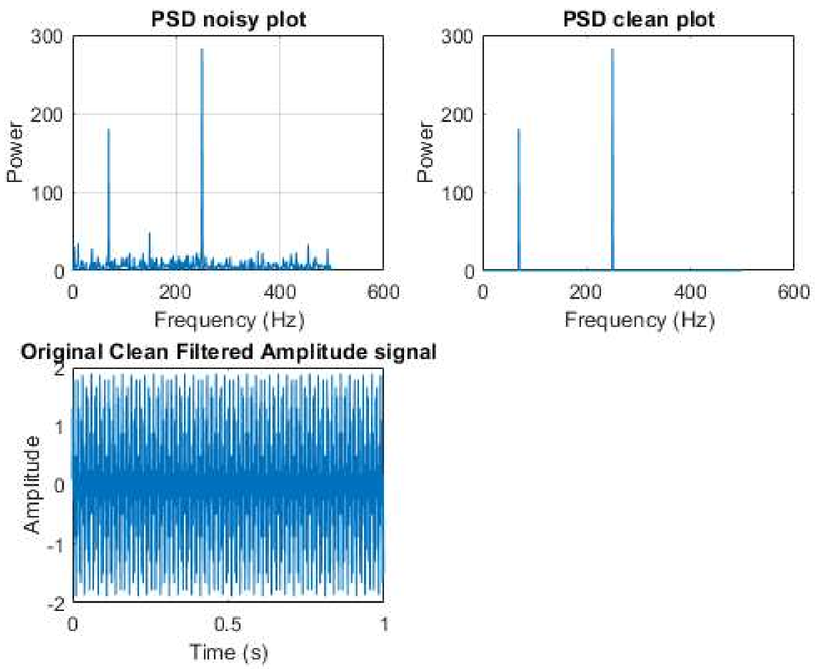
```
% Remove all global variables from the current workspace
clear;
% Clear all input & output from the command window display
clc;

% DFT (Discrete Fourier Transform)
% FFT (Fast Fourier Transform) - fast algorithm that computes DFT
% PSD (Power Spectrum Density)

% For a sampling frequency (fs) of 1000 and a length of a signal to analyze is 1sec, the signal is 1000 samples (or bins) long
fs = 1000;  % Sampling frequency = 1000Hz
T = 1/fs;   % Sampling interval or dt=0.001

f1 = 70;    % Sinusoidal Frequency = 70Hz
A1 = 0.8;   % Amplitude = 0.8

f2 = 250;   % Sinusoidal Frequency = 250Hz
A2 = 1.1;   % Amplitude = 1.1

t = 0 : T : 1 - T;          % Time vector of 1 sec

% Signal Input is a 2 tone signal containing a 70Hz sinusoid of amplitude 0.8 + a 250Hz sinusoid of amplitude 2.5
% It is assumed both sinusoidal signals have no phase
Sinewave1 = A1*sin(2*pi*f1*t);
Sinewave2 = A2*sin(2*pi*f2*t);

% Adding these 2 sinusoid signals to generate a CleanInput signal
CleanInput = Sinewave1 + Sinewave2;

% Corrupt the signal with random noise
Noise = 2.5*randn(size(t));

% Create a new signal NoisyInput, by injecting noise into the original signal CleanInput
NoisyInput = CleanInput + Noise;

%Length of t
N = length(t);              % Length of t, 1000

subplot(2,2,1)
plot(t,CleanInput); grid;
title('Original Clean Amplitude signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,2,2)
plot(t,NoisyInput); grid;
title('Noisy signal');
xlabel('Time (s)');
ylabel('Amplitude');

subplot(2,2,3)
plot(t,CleanInput,t,NoisyInput); grid;
title('Clean/Noisy signals');
legend('Clean-waveform','Noisy-waveform');
xlabel('Time (s)');
ylabel('Amplitude');
```




### Compute the FFT & PSD of the corrupted NoisyInput signal. Then Use the PSD to filter out noise

```
% There is N number of samples
% This generates a vector of fourier coefficients
% Each element of this vector is a complex number. It has a magnitude & a phase
Y = fft(NoisyInput, N);

% What is of interest is the magnitude of the complex numbers for each frequency
% Magnitude squared yields power

% PSD (Power Spectrum Density) :
% - Computes the power of each one of the fourier coefficient vector entries
% - Tells us how much power is in each frequency in the NoisyInput data

freq = 1/(T*N)*(0:N); % Create the x-axis of freqs from 0 -> N in Hz
L = 1:floor(N/2);     % Only plot the first half of freqs

PSD = abs(Y).^2/N;    % Power spectrum (how much power is in each frequency)
subplot(2,2,1)

% Plot the power spectral density vs those frequencies
% It shows us which frequency in Hz has the most power
% We can see clearly 2 large peaks and a bunch of noise
% This gives us some idea how to filter out our data
plot(freq(L),PSD(L)); grid;
title('PSD noisy plot');
xlabel('Frequency (Hz)');
ylabel('Power');

% Use the PSD to filter out noise
% Find all the indices of the frequencies where the power is greater than 100
%   1 where the power spectral is > 100
%   0 where the power spectral is < 100
indices = PSD > 100;

% Take the power spectum and multiply is with those indices
% Anything in the noise region gets multiplied by zero
% Only the values that had a power greater than 100 get multiplied by 1 so it only keeps the big peaks
PSDclean = PSD.*indices;

% Zero all entries with small fourier coefficients where the power spectral is < 100
Y = indices.*Y;

% Inverse fourier transform to get the Clean filtered signal in time
% We recover here the clean signal of the noisy data
ffilt = ifft(Y);

subplot(2,2,2);
plot(freq(L), PSDclean(L));
title('PSD clean plot');
```

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

subplot(2,2,3);
plot(t, ffilt);
title('Original Clean Filtered Amplitude signal');
xlabel('Time (s)');
ylabel('Amplitude');
```




*Published with MATLAB® R2020b*