DSP

Lab 2 Noise Analysis

A. Objectives:

- Use Matlab to simulate noise signals.
- Use the Histogram, Autocorrelation (AC), and Power Spectral Density (PSD) to analyze noise.
- Design an experiment to test a relationship between time window length in seconds for a PSD estimate and its frequency resolution in Hz.

B. Background

Noise is a process that cannot be predicted without error (using explicit functions/formulae). Noise is therefore characterized as a random variable (RV), which *is a function that maps an event into a real number*. The RV behavior can be described with a probability density function (PDF). Statistics are often used to describe noise amplitude fluctuations, such as means, variances, and root mean square (RMS) values. The most common noise model is an independent zero-mean Gaussian process. The power of a zero-mean noise signal is equivalent to the variance (RMS equivalent to the standard deviation). If noise is uncorrelated from sample to sample, it does not influence neighboring samples, and therefore cannot be estimated/predicted from neighboring samples. If, on the other hand, correlation exists between samples, then the correlation statistics can be used to predict a sample from its neighbors. So in addition to describing the signal amplitude fluctuations at a given time point, correlation between time samples is also characterized. This is done with the autocorrelation (AC) function in the time domain or the power spectral density (PSD) in the frequency domain. This lab focuses on simulating and analyzing noise signals with characteristics based on the PSD and PDF.

Probability Density Function:

Consider the event of making a measurement when no signal is present. Whatever is measured in this case is noise. Let $v[n]$ be a voltage value corresponding to a sample noise signal measurement whose amplitude distribution is described by the PDF $p_V(v)$. In the case of Gaussian distributed noise, the PDF is given by:

$$p_V(v) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(v-\mu)^2}{2\sigma^2}\right) \tag{1}$$

where μ is the mean and σ is the standard deviation (σ^2 is the variance). The mean or expected value of v is computed from the PDF by:

$$\mu = \mathbf{E}[v] = \int_{-\infty}^{\infty} v p_v(v) dv \tag{2}$$

where $\mathbf{E}[\cdot]$ is the expected value operator. If estimated from the data, the mean is computed from a simple average, often referred to as the sample mean:

$$\hat{\mu} = \frac{1}{N}\sum_{n=0}^{N-1} v[n] \tag{3}$$

Matlab's function *mean*() implements the formula in Eq. (3). The variance is computed from the PDF by:

$$\sigma^2 = \mathbf{E}\left[(v-\mu)^2\right] = \int_{-\infty}^{\infty} (v-\mu)^2 p_v(v) dv \tag{4}$$

Equation (4) can be expressed in several other useful forms by applying the linearity properties of the expected value operator:

$$\sigma^2 = \mathbf{E}\left[\left(v^2 - 2\mu v + \mu^2\right)\right] = \mathbf{E}\left[v^2\right] - 2\mu\mathbf{E}[v] + \mu^2 = \mathbf{E}\left[v^2\right] - \mu^2 \tag{5}$$

The variance can be computed from data (sample variance) with either the following formula:

$$\hat{\sigma}^2 = \frac{1}{N}\sum_{n=0}^{N-1}(v[n]-\hat{\mu})^2 = \frac{1}{N}\sum_{n=0}^{N-1}(v[n])^2 - \hat{\mu}^2 \tag{6}$$

Matlab's function *std*() can implement the above formula by setting the arguments accordingly however, but default it divides by N-1 rather than N. This is referred to as Bessel's correction and usually gives a better estimate of the population variance. Note that the above equation is equivalent to the formula for signal power, if the signal is zero-mean (if not zero mean, then power is equivalent to the 2nd moment, not the central 2nd moment, which is the variance).

The probably that a RV occurs within a range of values is computed from the integral of the PDF. The probability of $a \le v < b$ is given by:

$$\Pr[a \le v < b] = \int_a^b p_V(v) dv \tag{7}$$

The Gaussian distribution does not have a closed-form expression for its PDF integral (the cumulative distribution function (CDF)); however, tables and pre-computed functions exist from which these numbers can be obtained. The most famous is the *error function*, which Matlab implements with the function *erf*(x). This evaluated the CDF for the Gaussian RV with zero mean and a standard deviation equal to the reciprocal of $\sqrt{2}$. The PDF is integrated from -x to x, where x is the argument of the function. So for a zero-mean Gaussian noise process with $\sigma = 1/\sqrt{2}$ the probability that v is between $\pm b$ is computed from:

$$\Pr[-b \le v < b] = \mathrm{erf}(b) = \frac{2}{\sqrt{\pi}}\int_0^b \exp(-v^2) dv \tag{8}$$

Instead of integrating from –b, the lower limit is set to 0 and the integral is doubled to exploit the symmetry of the Gaussian PDF. For a general problem, the RV must be shifted to zero-mean and scaled down so its standard deviation becomes $\sqrt{2}$ to apply the error function. This scaling is applied to the original limits of integration and then strategically substituted into the error function of Eq. (8). For example, the probability that a Gaussian RV with mean 2 and standard deviation 3.5 is between *a* and *b* is given by:

$$\Pr[a \le v < b] = \frac{1}{2}\left(\operatorname{erf}\left(\frac{b-2}{3.5\sqrt{2}}\right) - \operatorname{erf}\left(\frac{a-2}{3.5\sqrt{2}}\right)\right), \tag{9}$$

Eq. (9) is equivalent to integrating the Gaussian PDF of mean 2 and standard deviation 3.5 over the interval from *a* to *b*.

The PDF can be estimated from sample data using the histogram and normalizing. Recall the histogram finds the number of sample values occurring between a sequence of contiguous intervals referred to as *bins*. If enough samples fall in each bin, then dividing by the total number of samples collected provides a statistically stable estimate of the probability that a sample value occurs in that bin, which is directly related to the PDF. So with enough sample data, the PDF can be estimated by computing the histogram with sufficiently small bin intervals and dividing each bin count by the total number of samples. The Matlab functions *hist*() and *histc*() are useful functions in performing PDF estimation. It should be noted, however, that bins containing only a few samples are unreliable estimates of the PDF. So less accuracy can be expected in the tails or low probability regions of the PDF, because for any given experiment to collect data for the histogram estimate, fewer samples will occur there than in other parts of the PDF.

Power Spectral Density and the Autocorrelation

If a process has inertia/memory, correlation will exist between samples, and the value of previous samples will influence the next sample. Most practical random processes are correlated to some degree. A noise processes that is uncorrelated, is referred to as ***white noise***. If correlation exists between samples, the noise process is referred to as ***colored noise***. White noise has a flat average spectrum (all frequencies have the same expected power). This is analogous to white light, which contains all frequencies in the optical spectrum. Colored noise has a frequency emphasis and typically results from filtering white noise with a band-limited filter.

The correlation between 2 signals is defined as the expected value between them, given by:

$$\sigma_{xy} = \mathbf{E}[xy] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(x-\mu_x)(y-\mu_y)\,p_{XY}(x,y)\,dx\,dy, \tag{10}$$

where p_{xy} is the joint PDF between RVs *x* and *y*. The autocorrelation (AC) function describes the correlation between a signal and its time lagged version. If the signal statistics do not change over time (i.e. a Gaussian process with a time-invariant mean and variance), it is referred to as a (wide-sense) *stationary process*. The AC function of a stationary process is a function of the time interval between the samples, independent of any particular time. The AC of a stationary signal is given by:

$$R(k) = \mathbf{E}\big[v[n]v[n-k]\big] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} v[n]v[n-k]\,p_{v[n]v[n-k]}(v[n],v[n-k])\,dv[n]\,dv[n-k] \tag{11}$$

where *k* is the time/sample distance between signals and is referred to as the *lag* value. The sample *biased* AC function is computed from:

$$\hat{R}[k] = \frac{1}{N}\sum_{n=k}^{N-1} v[n]v[n-k] \quad \text{for} \quad 0 \le k < N \tag{12}$$

where N is the length of available samples for the computation and the lag value k denotes the time offset between the products of the samples. Note the summation in Eq. (12) effectively averages all sample pair products separated by *k* units. For white noise, each sample is correlated completely with itself (occurs for *k*=0) and uncorrelated with its neighbors (occurs for *k*≠0). Therefore, the AC for a zero-mean white noise is effectively a discrete impulse function with the zero-lag amplitude equal to the variance (power) of the RV:

$$R[k] = \begin{cases} \sigma^2 & \text{for} \quad k = 0 \\ 0 & \text{for} \quad k \neq 0 \end{cases} \tag{13}$$

The Matlab function for computing an autocorrelation function is *xcorr*(). The unbiased version divides by *N*-*k* for each iteration, rather than *N*, to compensate for fewer points in the summation resulting from the increasing lag between the 2 vectors. The biased version is usually preferred because of the decreasing reliability in AC values computed from fewer samples. The biased version gives these values less weight.

The Fourier transform of the AC is the power spectral density function (PSD) and can be thought of as the average spectrum. Since the AC is symmetric, the phase of the PSD is zero. So it is completely described by its spectral magnitudes and given by the discrete Fourier transform (DFT) of the AC:

$$\hat{S}[m] = \sum_{k=-\frac{K}{2}}^{\frac{K}{2}-1} R[k]\exp\left(-j2\pi\frac{km}{K}\right) \tag{14}$$

where *K* is the number of lag values in the symmetric AC function (*R*[*k*]=*R*[-*k*]). One of the most popular ways to directly estimate the PSD is to:

1. break a long sequence of data into shorter overlapping segments
2. compute the DFT magnitude for each segment
3. average the magnitudes from the different segments together.

This results in an average spectral magnitude for each frequency. This approach is sometimes referred to as the time-hopping window method or Welch's method. This PSD estimation process can be expressed as:

$$\hat{S}[m] = \frac{1}{G}\sum_{g=0}^{G-1}\left|\hat{V}_g[m]\right|^2 \tag{15}$$

where V_g is the DFT extracted from the g^{th} windowed data segment $v[n]$, given by:

$$\hat{V}_g[m] = \frac{1}{N}\sum_{n=0}^{N-1} v[gL+n]w[n]\exp\left(-j2\pi\frac{nm}{N_{FFT}}\right) \qquad 0 \le m < N_{FFT} \tag{16}$$

where L is the number of samples between the start of each window, denoting the increment or hop distance between segments, $w[n]$ is windowing function that tapers the data down at each end of the extracted segment to limit the impact of abrupt changes in the signal as a results for segmenting it out of a continuous stream, and N_{FFT} is the number of DFT points over which DFT frequency samples are evaluated (usually achieved by padding with zeros so that $N_{FFT} = 2N$ or the next power of 2 greater than $2N$). If $L < N$, then N-L is the number of overlapping samples between adjacent windows. It is most common to have a 50% overlap between windows, especially with a tapering window. The Matlab function to estimate a PSD this way is *pwelch*().

In general when doing spectral estimation of a PSD, the following rules can be helpful when selecting parameters for the *pwelch*() function:

- Making N larger increases the length of the data segment (bigger time window) thereby reducing time resolution, but increasing frequency resolution. Decreased time resolution implies if two events happen within a time window, like a frequency shift, the spectrum will represent all frequencies simultaneously. You cannot determine the order of occurrence of the spectral. Increased frequency resolution implies finer detail in the spectrum. A noise-only process spectrum will typically have more jagged/random spikes, which can be smoothed out by reducing the window size and increasing the number of averages per DFT magnitude. However, the reduced frequency resolution will limit the ability to resolve 2 closely spaced deterministic (non-noise) frequencies.
- Making N smaller increases time resolution but reduces frequency resolution (spectrum will look more smooth, but possibly loosing fine detail of actual signal properties).
- Increasing the N_{FFT} parameter (number of DFT points) increases the grid points on which the spectrum is evaluated through interpolation. It does not enhance physical resolution, but provides a finer gird on the frequency axis to observe the spectrum.
- The finite window of data can be tapered to reduce the interference from strong frequencies in one part of the spectrum on weaker frequencies in another (spectral leakage). A tapered window enhances the time resolution by emphasizing one part of the time segment over the end points, but it reduces the frequency resolution. Matlab has functions to generate common tapering windows such as *boxcar*(), *hamming*(), *hann*(), *blackman*(), and *kaiser*(), in addition to the function *window*(), which provide access to many more tapering windows.

Spectral estimation parameters are chosen to limit artifacts and enhance critical details. However, this requires knowledge of the computational issues and trade-offs to find the best parameters for computing the spectrum. This lab introduces the basic concepts of spectral estimation and provides opportunities to apply tools for enhancing the reliability of estimated signal characterizations that are useful developing signal processing solutions (*e.g.* to build filters, classifiers, estimators, detectors …).

C. Pre-Laboratory Assignment

1. Compute the power value in dB for each signal listed below:
   a) A 60 Hz sine wave with amplitude of 1 volt.
   b) A DC voltage of 4 volts
   c) A 60 Hz sine wave with amplitude of 1 volt and a DC offset of 4 volts.
   d) A Gaussian random noise process with zero mean and standard deviation of 2.5 volt
   e) A Gaussian random noise process with a mean of 2 volts and standard deviation of 2.5 volts.

2. Given a sinusoidal **signal** with amplitude 2, and additive zero-mean Gaussian **noise** process. Find the value of the standard deviation (RMS) of the noise process, such that the SNR is 6 dB. Recall the SNR definition:

$$\mathrm{SNR}_{dB} = 10\log_{10}\left(\frac{\sigma_s^2}{\sigma_N^2}\right) = 20\log_{10}\left(\frac{\sigma_s}{\sigma_N}\right)$$

   where σ_S is the RMS value of the signal and σ_N is the RMS value of the noise process.

3. Find the probabilities p_1, p_2 and p_3 that the absolute value of a zero-mean Gaussian noise process with standard deviation of 2 volts exceeds each threshold value of 0.4, 2, and 5 volts, respectively.

4. Use Matlab code below to generate a signal consisting of the sum of 2 sine waves with unit amplitude for 2 seconds at a frequency of 50 Hz and 51 Hz with a sampling rate of 500 Hz. Assign the vector of points to variable *g* in Matlab's workspace.

```
>> fs = 500;
>> t = [0:round(fs*2)-1]/fs;
>> g1 = sin(2*pi*50*t);
>> g2 = sin(2*pi*51*t);
>> g = g1+g2;
```

   Take the DFT with various levels of zero padding (N_{FFT} parameter = 1000, 2048, and 4096 points using the commands:

```
>> s1 = fft(g,1000);
>> s2 = fft(g,2048);
>> s3 = fft(g,4096);
```

   Create corresponding frequency axes using:

```
>> f1 = fs*[0:length(s1)-1]/length(s1);
>> f2 = fs*[0:length(s2)-1]/length(s2);
>> f3 = fs*[0:length(s3)-1]/length(s3);
```

Then plot them all on the same graph with a linear scale using

```
figure(1)
plot(f1,abs(s1/length(g)),'k-')
hold on
plot(f2,abs(s2/length(g)),'g--')
plot(f3,abs(s3/length(g)),'b:')
```

Zoom in on a region frequency region of where the sine wave is located and present the plot (copy and paste graph in pre-lab assignment). Comment on the impact of padding with zeros.

5. Repeat Problem 4 using only a 0.25 second duration (instead of 2 seconds) for the sinusoidal signal. Comment on the impact of the actual window length.

6. Study the example mfile script *psdex.m* posted on the class website. There is nothing hand in on this part. Read through the comments and the code, run it and change parameters (such as the window length, zero padding, window shape, SNR …) to get a better idea of how to use the *pwelch*() function in Matlab to estimate a PSD.

D. Laboratory Assignment

Probability Density Function Estimation

No Write-Up is required in this section. Requested figures and explanations must be presented to the TA/Instructor for full demo credit **(2 points)**. I recommend pasting the requested figures in a presentation software package with brief notes to help with your explanations of what is going on.

The histogram of a set of random measurements can be used as an estimate of their PDF by summing the occurrences in each bin and normalizing by the sum of all occurrences. This way the area under the entire histogram curve is unity (just like the PDF). In Matlab this can be done for a vector, *h*, of random samples with the following commands:

```
%  Take histogram of h with 31 uniform bins from min to max values of h
>> [n, bc] = hist(h,31); % vector n is number of occurrence in each bin
                         % and vector bc contains the bin centers
>> p = n/sum(n);  %  normalize occurrences to create estimate of PDF
>> bar(bc, p)  % use bar plot to show estimate probability of
               % occurrence in each bin
```

1. Generate 250 Gaussian random values with zero mean and standard deviation 2.5. Plot the estimated PDF using 11, 41, and 91 bins. Present the plots to the TA (don't forget to label the axes). **Explain how this illustrates the impact of the number of bins on the PDF estimate.**

2. Repeat Problem 1 with 25000 Gaussian random values**. Explain how this illustrates the impact of the number of samples on the PDF estimate**

3. Generate 50000 Gaussian random values with zero mean and standard deviation 3. Then use the function *sigclip*() (download from class web site) to clip values to [-0.4, 0.4], [-2, 2], and [-5, 5]. Estimate the PDFs for each case before and after clipping (you must decide on the best/good number of bins for the estimate). Present the plots to the TA/Instructor. **Explain how this illustrates the impact of clipping on the PDF of the RV. Use the results from the pre-laboratory Problem 3 to help in your discussion (should be able to explain/predict the heights of the end bins).**

Signal and Noise Generation/Characterization

No Write-Up is required in this section. Requested figures and explanations must be presented to the TA/Instructor for full demo credit **(2 points)**. I recommend pasting the requested figures in a presentation software package with brief notes to help with your explanations of what is going on.

4. Generate a 60 Hz sine wave with amplitude 1 for 4 seconds at a sampling rate of 600 Hz (i.e. f_s = 600 Hz). Assign it to variable *sig* in Matlab's workspace. Also generate a corresponding zero-mean white Gaussian noise signal such that the signal to noise ratio is 6 dB, **if** they were added together. Assign the noise vector to variable *nos* in Matlab's workspace. For each of the vectors, compute and plot the following (don't forget to label axes):

   a. The AC with 50 lags maximum (**example:** [sigac, lags] = xcorr(sig, 50); and [nosac, lags] = xcorr(nos, 50);)

   b. The PSD with window size 64 samples, hamming tapering window, overlap of 32 samples, and 128 FFT evaluation points (**examples:** [sigpsd, f] = pwelch(sig, hamming(64), 32, 128, fs); [nospsd, f] = pwelch(nos, hamming(64), 32, 128, fs);)

   Present the plots to the TA. **Point out the important features of the plots and explain how they are consistent or inconsistent with your expectations for these particular signal and noise signals.**

5. Add the 2 signal vectors (*nos* and *sig*) of Exercise 4 together and plot the time waveforms over a 0.1 second interval. Compute the PSD and AC for the signal-plus-noise vector as in Exercise 4 and create plots of the PSD and AC. Present these plots to the TA/Instructor. **Point out the features of the PSD and AC that are primarily influenced by the signal and those that are primarily influenced by the noise.**

6. Filter the signal-plus-noise vector created in Exercise 5 using a low-pass filter with cut-off frequency at 100 Hz. If your signal-plus-noise vector is called “spn”, then this filtering can be implemented with the following command:

   ```
   >> fspn = filter([.02 .23 .51 .23 .02], 1, spn);
   ```

   where the vector in the first 2 arguments contains filter coefficients for the low-pass filter (see *help filter* in Matlab for a better understanding of this function and it arguments). Zoom in on a section of the waveform such that 4 to 8 cycles are contained in the plot window. Compute the PSD and AC for the filtered signal-plus-noise vector as in Exercises 4 and 5. Present the plots to the TA/Instructor. **Describe the impact of low-pass filtering on the data from time waveform, PSD, and AC plots with the noise and signal.**

PSD Estimation

This section requires a write-up for an experiment designed to test a relationship between the time window size and frequency resolution for PSD estimation. Follow the format in the syllabus for reporting (5 points).

The impact of time window length on the frequency resolution of the PSD estimate is studied. There is an approximate relationship between frequency resolution and the time window used to compute the DFT. This is given as:

$$\Delta_f \approx \frac{1}{T} \tag{17}$$

where T is the time window length for the DFT segment in seconds ($T = N / f_s$) and Δ_f is the frequency resolution in Hertz, The frequency resolution denotes the smallest frequency interval in which 2 frequencies can be resolved (appear as 2 distinct peaks). Your task is to design, perform, and report on an experiment to determine if the above relationship is valid (i.e. you can hypothesize the relationship in Eq. (17)). Things you will have to decide on in your experiment are:

- ranges of frequencies and separations to test for
- Number of trials and variations between them
- rules for determining when 2 frequencies appear distinct in the PSD.
- a criterion for either accepting or rejecting the hypothesis.

Below is an exercise you can play with to provide some intuition on building your own experiment. Other experimental parameters may or may not be critical to the relationship under test, such as noise level, tapering windows, sampling frequency, and number of FFT points. However given the time limitations for this course, I suggest setting a noise level, tapering window, sampling rate, and number of FFT points to something reasonable and keeping those constant. You should write the code for your experiment such that it would be trivial to change these values. You can discuss the potential impact of these in your analysis, but do not have to run an experiment to support your hunches.

7. **Warm-up exercise:** Create 2 sinusoids of unit amplitude separated by 5 Hz (i.e. 50 and 55 Hz) for 10 seconds and a sampling rate of 500 Hz. Create a corresponding zero-mean Gaussian noise signal with a standard deviation such that the SNR relative to the sine waves is 30 dB. Add all 3 vectors together and estimate the PSD for various values of DFT window lengths N to find the smallest window length at which the 2 frequencies can be resolved (you will have to use your judgment as to when the peaks for the signal cannot reliably be distinguished from the noise peaks). **The rule for determining this is part of your experimental design.** Denote this window length as N_R along with its equivalent length in seconds T_R. For the PSD estimation use a boxcar window and with a 50% overlap, and N_{FFT} points equal to 6 times the value of N.

   example: [pd, f] = pwelch(sigvec, boxcar(N), round(N/2), 6*N, fs);