This Lecture

- Fourier series
- Fourier transform
Instrumentation System


Digital Acquisition System

- Computers are nearly always in the middle of any instrumentation system to provide a complete interface with analog sensors and output devices.
Signal Processing

- The big question now is: “What information do we want to extract from the signal(s) that we have acquired?”
  - Amplitude
  - Frequency
  - Timing
  - Phase
  - Etc.
- Let’s focus on Frequency

A Sine Wave

\[ 5 \sin (2\pi t) \]
Amplitude = 5
Frequency = 4 Hz
**A Sampled Sine Wave Signal**

5\*\text{sin}(2\pi 4t)  
Amplitude = 5  
Frequency = 4 Hz  
Sampling rate = 256 samples/second  
Sampling duration = 1 second

**An Undersampled Signal**

\text{sin}(2\pi 8t), \text{SR} = 8.5 \text{ Hz}
The Nyquist Frequency

- The Nyquist frequency is equal to one-half of the sampling frequency.
- The Nyquist frequency is the highest frequency that can be measured in a signal.
- In other words – you **MUST** sample at least twice as fast as the fastest frequency in the signal you wish to capture.
- And, how do you determine what that maximum frequency is? Or more generally, how do you determine all of the frequencies comprising the signal you are capturing?

Fourier Series

- **Fourier Series** (English pronunciation: /ˈfɔːriə/ (T)) is a way to represent a wave-like function as the sum of simple sine waves. More formally, it decomposes any **periodic function** or periodic signal into the sum of a (possibly infinite) set of sine and cosine functions.

\[
F(t) = B_0 + \sum_{n=1}^{\infty} A_n \sin(n\omega t) + \sum_{n=1}^{\infty} B_n \cos(n\omega t)
\]

- \( B_0 = \frac{1}{T} \int_{T} F(t) \, dt \)
- \( A_n = \frac{2}{T} \int_{T} F(t) \sin(n\omega t) \, dt \)
- \( B_n = \frac{2}{T} \int_{T} F(t) \cos(n\omega t) \, dt \)

Where \( k = \frac{2\pi}{T} \)
### Alternative Mathematical Representations

\[ y(t) = A_0 + \sum_{n=1}^{\infty} \left( A_n \cos nt \right) + B_n \sin nt \]

may be written as

\[ y(t) = A_0 + \sum_{n=1}^{\infty} C_n \cos (nt - \phi_n) \]

or

\[ y(t) = A_0 + \sum_{n=1}^{\infty} C_n \sin (nt + \phi_n^*) \]

where

\[ C_n = \sqrt{A_n^2 + B_n^2} \]

\[ \tan \phi_n = \frac{B_n}{A_n} \text{ and } \tan \phi_n^* = \frac{A_n}{B_n} \]

### Fourier Series

- Introduction
  - [https://www.youtube.com/watch?v=UKHBWzoOKsY](https://www.youtube.com/watch?v=UKHBWzoOKsY)
- Frequency filtering
  - [https://www.youtube.com/watch?v=JndvN1ngSi4](https://www.youtube.com/watch?v=JndvN1ngSi4)
- Matlab demo
Fourier Series Symmetry

- If \( f(t) = f(-t) \), then \( f(t) \) is said to be an **even** function and all the \( b_n \) terms = 0. In other words, \( f(t) \) is mirrored about the y-axis, like the cosine function.

- If \( f(t) = -f(-t) \), then \( f(t) \) is said to be an **odd** function and all \( a_n \) terms (including the dc value) = 0. The sine function is an example.

Fourier Series Example

- Find the Fourier Series for the following waveform. Set \( V_m = 1 \) and \( T = 2\pi \). See the Matlab file `fourier_series_square.m` on the course web page for implementation.

\[
v(t) = \sum_{n=1}^{N} \frac{4}{n\pi} \sin(nt)
\]
The Fourier Transform

- A transform takes one function (or signal) and turns it into another function (or signal).
- Continuous Fourier and Inverse Fourier Transforms:

\[
H(f) = \int_{-\infty}^{\infty} h(t)e^{2\pi jft} dt
\]
\[
h(t) = \int_{-\infty}^{\infty} H(f)e^{-2\pi jft} df
\]

- Note that the transforms contain complex numbers.

The Discrete Fourier Transform

- Because we have computers in the mix and deal with discrete samples of information, mathematicians developed the **Discrete Fourier Transform (DFT)**:

\[
H_n = \sum_{k=0}^{N-1} h_k e^{2\pi jnk/N}
\]
\[
h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n e^{-2\pi jnk/N}
\]

- The DFT is also complex in nature.
Fast Fourier Transform

- You have probably heard about the Fast Fourier Transform (FFT). The FFT is an efficient algorithm for performing a Discrete Fourier Transform.
- The FFT algorithm was first published by Cooley & Tukey in 1965.
- In 1969, the 2048 point analysis of a seismic data trace took over 13 hours. Using the FFT, the same task on the same machine took 2.4 seconds.
- The FFT is, by far, the most common frequency analysis algorithm used in programs such as Matlab.

Example - Sample Waveform with $\tau = 5s$
Example - Sample Waveform with $\tau = 2s$

Example - Sample Waveform with $\tau = 1s$
Composite Waveform

Fourier Transform Information
A Little Video Help

- **Graphical Depiction of Fourier Series (watch first 10 minutes)**
- **Introduction to Fourier Transform**

Notes on the Fourier Transform

- The FFT takes, as inputs, a vector of discrete points representing magnitudes in the time domain.
- The FFT returns a set of complex numbers containing magnitude and phase information in the frequency domain.
  - Note that the data is duplicated from the midpoint on, actually mirrored about the x-axis since it returns values for frequencies between – infinity and +infinity.
- Sampling rate issues
  - Sampling too slow.
  - Sampling too fast and aliasing.
    - Possibility of not acquiring an entire period of the lowest frequencies.
- FFT illustration via Matlab.
% FFT Example
% Revision 3.0 Curt Nelson 1/30/2019
% Creates functions of time and explores various fft implementations
clear all;
clear plot;

% Calculate the number of time samples (1000 in this case).
start_time = 0;
end_time = .01; % 10 milli-seconds
delta_time = 1e-5; % 10 micro-seconds
number_time_samples = (end_time - start_time)/delta_time; % 1000 points

% Create time vector xor x-axis (1000 data points)
time_vec = start_time:delta_time:end_time;

% The sampling rate is 1/delta_time or 100,000 samples/second
sampling_rate = 1/delta_time;

% Next create the time domain function with a frequency of 200Hz, resulting
% in a period of 1/frequency or 5 milli-seconds.
freq = 200;
period = 1/freq;
time_function = sin(2*pi*freq*time_vec);

% Since we are sampling from 0 to 10ms, we should see 2 cycles
plot(time_vec,time_function);
title('Time Function with 1000 Data Points');
ylabel('volts');
xlabel('time - seconds');
grid on;
pause;
\[ \text{time\_function} = \sin(2\pi \cdot 200 \cdot x1t) \]

Matlab Code

```
% Now do an FFT on this time domain function
% fft\_results contain a complex number pair for each sample
fft\_results = fft(time\_function);

% Create the x axis for frequencies starting at the DC value (0 Hz)
dc\_value = 0;

% We only need to plot the first half of the frequencies because the fft returns
% the same data folded over on itself at maxfreq/2
% Frequency spacing is the sampling rate / by the number of samples and is
% the frequency resolution on the x axis.
freq\_spacing = sampling\_rate/number\_time\_samples;

% Maximum frequency for the fft is (sampling rate/2) – freq\_spacing
freq\_max = (sampling\_rate/2) - freq\_spacing;
```
FFT Matlab Code

% Next, create the x-axis points (0 - 49,900 in increments of 100Hz)
freq_plot_xaxis = dc_value:freq_spacing:freq_max;

% This results in (number of time samples/2) or 500 frequencies
number_freq_samples = number_time_samples/2;

% The magnitude of the fft must be computed from the complex fft_results
magnitude = abs(fft_results);

% Normalize magnitude by dividing by the number of frequency samples
nor_magnitude = magnitude/number_freq_samples;

% Plot the first 30 frequencies using red circles
plot(freq_plot_xaxis(1:30),nor_magnitude(1:30),’ro’);
pause;

FFT of sin(2*pi*200*t)
FFT Matlab Code

% Next, Let's create a signal of two frequencies with different magnitudes
new_sig = sin(2*pi*100*time_vec) + 2*sin(2*pi*200*time_vec);
plot(time_vec,new_sig);
pause;

FFT of sin(2*pi*100*time_vec) + 2*sin(2*pi*200*time_vec)

% Perform the fft and plot
fft_results = fft(new_sig);
magnitude = abs(fft_results);
nor_magnitude = magnitude/number_freq_samples;
plot(freq_plot_xaxis(1:30),nor_magnitude(1:30),’ro’);
pause;
FFT Matlab Code

% Finally, lets add eight more frequencies with various magnitudes and plot
new_sig = new_sig + .5*sin(2*pi*300*time_vec);
new_sig = new_sig + 2*sin(2*pi*400*time_vec);
new_sig = new_sig + 3*sin(2*pi*500*time_vec);
new_sig = new_sig + sin(2*pi*600*time_vec);
new_sig = new_sig + 2*sin(2*pi*700*time_vec);
new_sig = new_sig + .5*sin(2*pi*800*time_vec);
new_sig = new_sig + sin(2*pi*900*time_vec);
new_sig = new_sig + 3*sin(2*pi*1000*time_vec);

plot(time_vec,new_sig); % Plot the function
pause;

f(t) Composed of 10 Frequencies
% Perform the fft and plot
fft_results = fft(new_sig);
magnitude = abs(fft_results);
nor_magnitude = magnitude/number_freq_samples;
plot(freq_plot_xaxis(1:30),nor_magnitude(1:30),'ro');
pause;