CAL POLY

EE 367 Lab 1

Sampled Signals and Spectra

Report by:

Ethan Vosburg (evosburg@calpoly.edu)
Isaac Lake (ilake@calpoly.edu)
Ryan Ceballos (rceballo@calpoly.edu)

January 30, 2024

Table of Contents

1 Interpreting Spectra with Signals Generated in Python

1.1 Part 1 Design Specifications:

For part 1 we will be making a wave file with Python. The wave file will be constructed according to these specifications.

Attribute	Specification
Duration	2 Seconds
Frequency	$2000Hz$
Sample Rate	$16kHz$
Channel Count	Mono
Amplitude	75%
Phase	$45°$

Table 1: Wave Specifications

1. How many samples are generated in total?

   There would be 32,000 Samples
2. How many bytes are expected for the WAV file?

   With the addition of the 44 bytes for in the header, the resulting calculation is $32,000 + 44$ totaling $32,044$ bytes.
3. What is your chosen amplitude?

   The chosen amplitude is calculated by multiplying available bits for numeric values 2^{15} by the desired amplitude 75% resulting in $24,576$ samples.
4. What is the digital frequency in rads/sample?

   The digital frequency can be calculated by the following:

$$F = \frac{f_0}{f_s}$$

$$F = \frac{2kHz}{16kHz}$$

$$F = \frac{1}{8} = 0.125$$

5. MatLab Plot

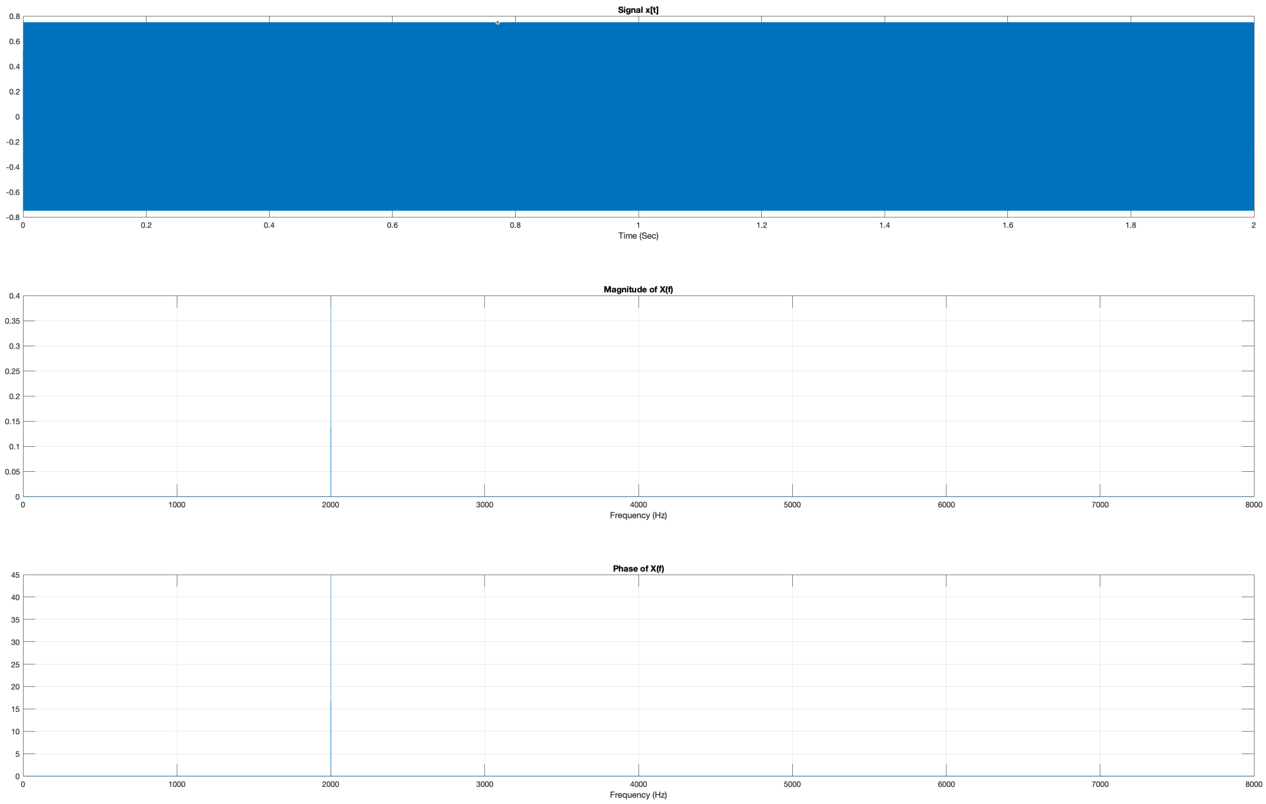

Figure 1: Caption

6. Expected magnitude at f = 2000 Hz? Any error seen in the plot?

   The expected magnitude was .75 which matches the plot

7. Expected phase at f = 2000 Hz? Any error seen in the plot?

   45° is the expected phase. No error seen in the plot

8. Python routine gen_wav

Listing 1: Python Routine for Generating 2000Hz

```
"""
File: EE367-Lab1-1.py
Author: Ethan Vosburg
Date: 01-25-24
Description: This file contains the code for part 1 of Lab 1 for EE367. This
file creates a 2 second long 2000 Hz cosine wave with a 45 degree phase shift.
"""

# Import Statements
import sys
import time
import base64
import random as random
import datetime
```

```python
import time
import math

from cpe367exp1.cpe367_wav import cpe367_wav


def wavgen(dur, freq, rate, channels, amp, phase, delay, fpath_wave_out):
    """This function generates a WAV file

    Arguments:
        dur :               Duration of the wave in seconds
        freq :              Frequency of the wave in Hz
        rate :              Sample rate of the wave in Hz
        channels :          Number of channels in the wave
        amp :               Amplitude of the wave
        phase :             Phase of the wave
        delay :             Delay of the wave in seconds
        fpath_wave_out :    File path of the wave

    Returns:
        True or False

    """
    # Create a new wav file object
    wav_out  = cpe367_wav('wavOut', fpath_wave_out)

    # Configure the wav file object
    wav_out.set_wav_out_configuration(
        channels,
        16,
        rate,
    )

    # Check if the wav file object was configured correctly
    if wav_out == False:
        print("Error setting wav configuration")
        return False

    # Open the wav file
    working_file = wav_out.open_wav_out()
    if working_file == False:
        print("Error opening file for writing")
        return False

    #  Calculate number of samples needed and the angular frequency
    sample_count = int(dur * rate)
    w1 = 2 * math.pi * freq / rate

    # Generate the samples and write them to the wav file
    for i in range(sample_count):
        working_sample = amp * math.cos(w1 * (i - delay * rate) + (phase/360 * 2 * math.pi))
        working_sample = int(round(working_sample))

        working_file = wav_out.write_wav(working_sample)
        if working_file == False: break
        if working_file == False: break

    # Close the wav file
    wav_out.close_wav()
    return True

# Run the main function
def main():
    try:
        fpath_wav_out = 'Lab1-1.wav'
        wavgen(2, 2000, 16000, 1, 24576, 45, 0, fpath_wav_out)
    except:
        print("Error: ", sys.exc_info()[0])
        return False

```

```
    return True

# Checking if program is run as main
if __name__ == '__main__':
    if (main() == True):
        print("Waveform Generated Successfully")

    # Exit the program
    quit()
```

2 Impact of Delay on Frequency Spectra

1. What is the equivalent delay?
   A 45°delay is equivalent to $\frac{1}{8}$ the period or $62.5\mu s$
2. What is the equivalent phase shift, theta, (in radians)?
   A 45°phase shift is equivalent to $\frac{\pi}{4}$ radians.
3. Describe similarities and differences between the delayed and original version
   The similarities are that they have the same amplitude and frequency. The differences are that they are out of phase and are offset at any given time from their non-shifted versions.
4. Describe similarities and differences between the delayed version and the one with -A. Explain why these similarities or differences arise.
   A negative A is the equivalent of shifting by π in other words it will be even more offset from our mere $\frac{\pi}{4}$ offset. It should similarly still share the same frequency and magnitude.

3 Impact of Amplitude Modulation on Frequency Spectra

1. What mathematical identity allows you to predict the spectrum of x(t)?
   $Acos\alpha * Bcos\beta = A * B(1/2)[cos(\alpha + \beta) + cos(\alpha - \beta)]$
2. What are your values for A and B?
   A = 128, B = 128
3. What are your values for θ_1 and θ_2?
   θ_1 = +45°and θ_2 = -45°
4. If the order of the cosine functions is swapped then would x(t) change?
   If the order of the cosine functions were swapped, x(t) would not change.

5. MatLab Plots

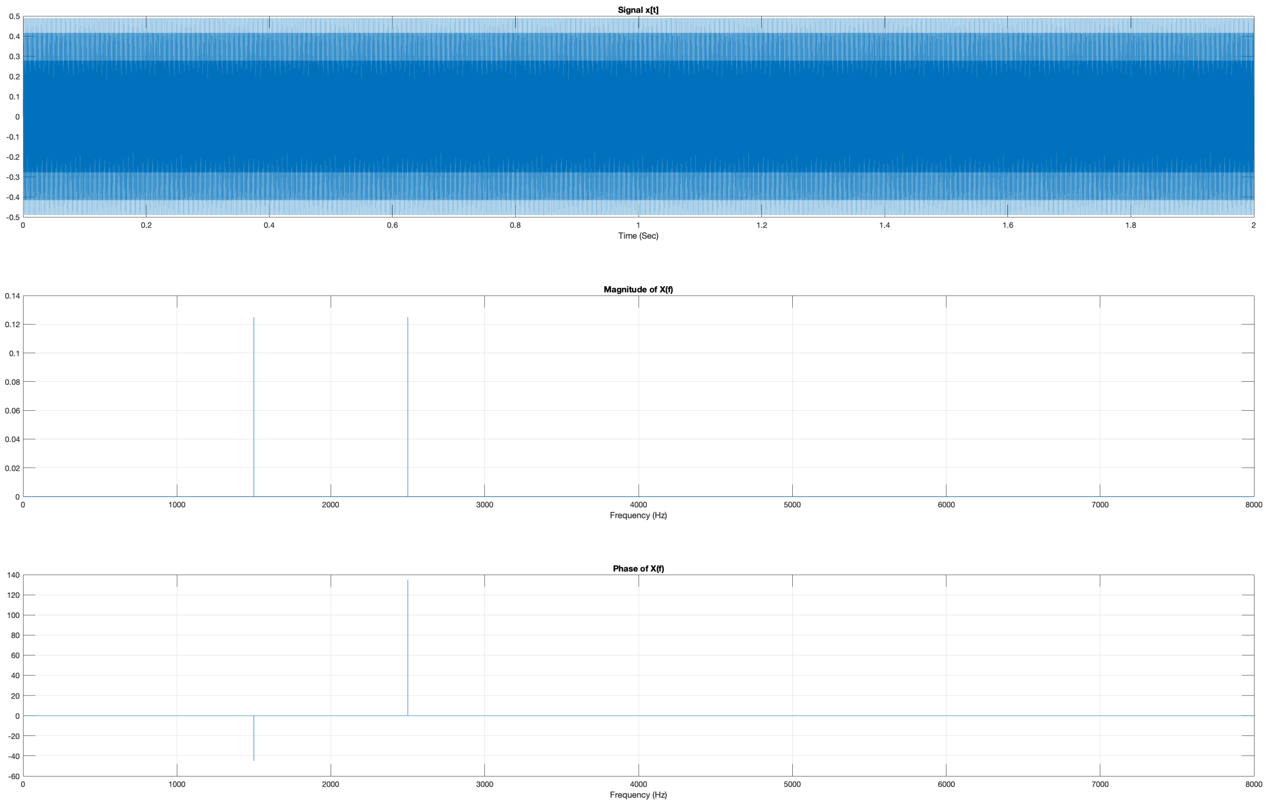

Figure 2: Caption

6. Python routine ‘gen_wav’, with comments

Listing 2: Python Routine for Generating Modulating Signal

```
"""
File: EE367-Lab1-3.py
Author: Ethan Vosburg
Date: 01-25-24
Description: This file contains the code for part 1 of Lab 1 for EE367. This
file creates a 2 second long modulated wave with a 2000 Hz cosine wave and a
500 Hz cosine wave.
"""

# Import Statements
import sys
import time
import base64
import random as random
import datetime
import time
import math

from cpe367exp1.cpe367_wav import cpe367_wav


```

```python
def wavgenmod(dur,
              freq1,
              freq2,
              rate,
              channels,
              amp1,
              amp2,
              phase1,
              phase2,
              delay,
              fpath_wave_out):
    """This function generates a WAV file

    Arguments:
        dur :               Duration of the wave in seconds
        freq1 :             Frequency of the wave in Hz
        freq2 :             Frequency of the wave in Hz
        rate :              Sample rate of the wave in Hz
        channels :          Number of channels in the wave
        amp :               Amplitude of the wave
        phase :             Phase of the wave
        delay :             Delay of the wave in seconds
        fpath_wave_out :    File path of the wave

    Returns:
        True or False

    """
    # Create a new wav file object
    wav_out  = cpe367_wav('wavOut', fpath_wave_out)

    # Configure the wav file object
    wav_out.set_wav_out_configuration(
        channels,
        16,
        rate,
    )

    # Check if the wav file object was configured correctly
    if wav_out == False:
        print("Error setting wav configuration")
        return False

    # Open the wav file
    working_file = wav_out.open_wav_out()
    if working_file == False:
        print("Error opening file for writing")
        return False

    #  Calculate number of samples needed and the angular frequency
    sample_count = int(dur * rate)
    w1 = 2 * math.pi * freq1 / rate
    w2 = 2 * math.pi * freq2 / rate

    # Generate the samples and write them to the wav file
    for i in range(sample_count):
        wave1 = amp1 * math.cos(w1 * (i - delay * rate) + (phase1/360 * 2 * math.pi))
        wave2 = amp2 * math.cos(w2 * (i - delay * rate) + (phase2/360 * 2 * math.pi))
        working_sample = wave1 * wave2
        working_sample = int(round(working_sample))

        working_file = wav_out.write_wav(working_sample)
        if working_file == False: break
        if working_file == False: break

    # Close the wav file
    wav_out.close_wav()
    return True

# Run the main function
```

```python
def main():
    try:
        fpath_wav_out = 'Lab1-3.wav'
        wavgenmod(2, 2000, 500, 16000, 1, 128, 128, 45, -45, 0.00025, fpath_wav_out)
    except:
        print("Error: ", sys.exc_info()[0])
        return False

    return True

# Checking if program is run as main
if __name__ == '__main__':
    if (main() == True):
        print("Waveform Generated Successfully")

    # Exit the program
    quit()
```

4 Impact of Absolute Value on Frequency Spectra

1. Describe the frequency components that are present in this signal. Is there a pattern? What are the lowest two frequencies present? Is DC present?

   The lowest two frequencies present in this signal are 320 Hz and 640Hz. Yes, DC is present and has a magnitude of 0.477305. Since the frequency of the cosine function is 160Hz, we see double this frequency on the positive side and our graphs show this perfectly. There is a pattern present in the data, the frequency found in the magnitude response increases by 320Hz every time.

2. MatLab plots

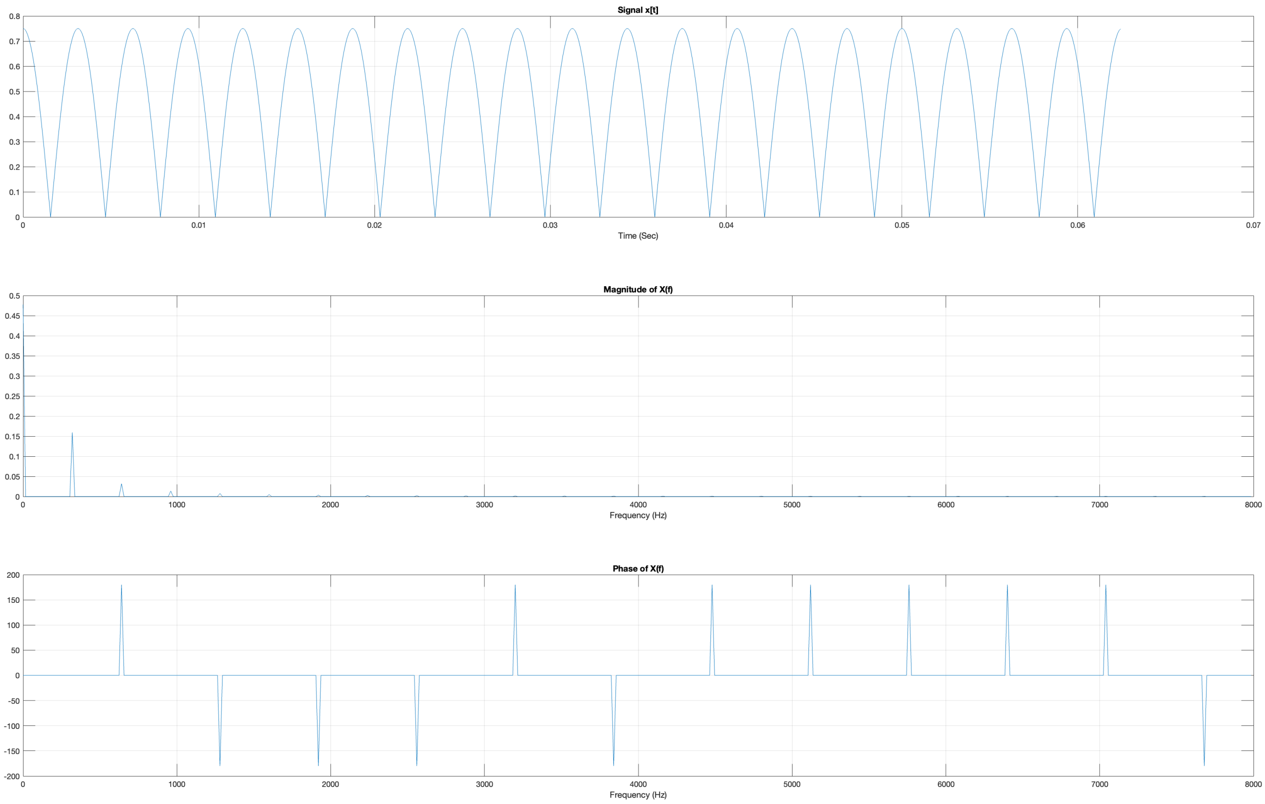

Figure 3: Caption

3. Python routine ‘gen_wav’, with comments

Listing 3: Python Routine for Generating Absolute 160Hz

```
"""
File: EE367-Lab1-4.py
Author: Ethan Vosburg
Date: 01-25-24
Description: This file contains the code for part 1 of Lab 1 for EE367. This
file creates a 1/16 second long 160 Hz cosine wave.
"""

# Import Statements
import sys
import time
import base64
import random as random
import datetime
import time
import math

from cpe367exp1.cpe367_wav import cpe367_wav


def wavgen(dur, freq, rate, channels, amp, phase, delay, fpath_wave_out):
```

```
    """This function generates a WAV file

    Arguments:
        dur :               Duration of the wave in seconds
        freq :              Frequency of the wave in Hz
        rate :              Sample rate of the wave in Hz
        channels :          Number of channels in the wave
        amp :               Amplitude of the wave
        phase :             Phase of the wave
        delay :             Delay of the wave in seconds
        fpath_wave_out :    File path of the wave

    Returns:
        True or False

    """
    # Create a new wav file object
    wav_out  = cpe367_wav('wavOut', fpath_wave_out)

    # Configure the wav file object
    wav_out.set_wav_out_configuration(
        channels,
        16,
        rate,
    )

    # Check if the wav file object was configured correctly
    if wav_out == False:
        print("Error setting wav configuration")
        return False

    # Open the wav file
    working_file = wav_out.open_wav_out()
    if working_file == False:
        print("Error opening file for writing")
        return False

    #  Calculate number of samples needed and the angular frequency
    sample_count = int(dur * rate)
    w1 = 2 * math.pi * freq / rate

    # Generate the samples and write them to the wav file
    for i in range(sample_count):
        working_sample = abs(amp * math.cos(w1 * (i - delay * rate) + (phase/360 * 2 * math.pi)
            ))
        working_sample = int(round(working_sample))

        working_file = wav_out.write_wav(working_sample)
        if working_file == False: break
        if working_file == False: break

    # Close the wav file
    wav_out.close_wav()
    return True

# Run the main function
def main():
    try:
        fpath_wav_out = 'Lab1-4.wav'
        wavgen(0.0625, 160, 16000, 1, 24576, 0, 0, fpath_wav_out)
    except:
        print("Error: ", sys.exc_info()[0])
        return False

    return True

# Checking if program is run as main
if __name__ == '__main__':
    if (main() == True):
        print("Waveform Generated Successfully")
```

```

    # Exit the program
    quit()
```