TCET 3102-E316 (Analog and Digital Communications) Lab 1

March 13, 2019

Name: Christ-Brian Amedjonekou
Date: 2/04/2019
TCET 3102-E316 (Analog and Digital Communications) Lab 1
Spring 2019, Section: E316, Code: 37251
Instructor: Song Tang

References:

- http://www.thefouriertransform.com/series/coefficients.php
- https://flothesof.github.io/Fourier-series-rectangle.html
- https://docs.scipy.org/doc/scipy/reference/tutorial/fftpack.html
- https://plot.ly/matplotlib/fft/
- https://docs.scipy.org/doc/scipy/reference/tutorial/fftpack.html#one-dimensional-discrete-fourier-transforms

0.0.1 Modules (Packages)

```
In [1]: # These are the packages I'll need to solve this problem
        import math as m
        import numpy as np
        from matplotlib import pyplot as plt
        from scipy.fftpack import fft, fftfreq
```

0.0.2 Variables for time domain plot

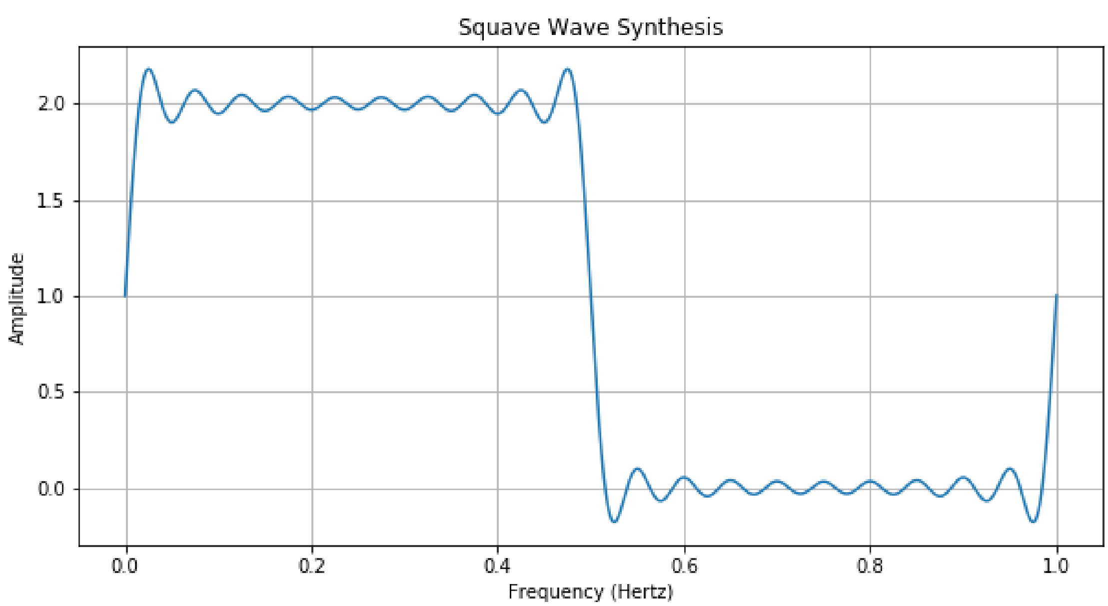
```
In [2]: # These variables are used to create the fourier series
        # Start and Stop indicates my domain for my sampling frequency [samplingfreq; (F_S)]
        # 100 points.
        # n1, n2 are the amount of harmonics I want.
        start, stop, n1, n2, samplingfreq = 0, 100, 9, 29, 100

        # 't' is for time, and is used to create my 100 Hz time vector
        t = np.arange(start, stop, .001) / samplingfreq

        # 'A' represents the amplitude 2 volts
        A = 2
```

```
# 'fundamental' is the DC component of the Fourier Series
fundamental = A/2

# 'signalfreq' is the Signal Frequency (f_0)
signalfreq = 1

# 'omega' is the Angular Velocity (w_0)
omega = 2 * np.pi * signalfreq

# Lambda function
template = lambda p: ((2*A)/(np.pi*(2*p+1))) * np.sin((2*p+1) * omega * t)

# harmonics1, harmonics2 are AC component of the Fourier series
harmonics1 = sum([template(p) for p in range(n1+1)])
harmonics2 = sum([template(p) for p in range(n2+1)])

# ffs1, ffs2 are the fourier series
ffs1 = lambda n: (fundamental + harmonics1)
ffs2 = lambda w: (fundamental + harmonics2)
```

0.0.3 RUN 1: Plot of the fourier series in the time domain

- **Step 1: Plot Fourier Series w/ 9 members (waves)**

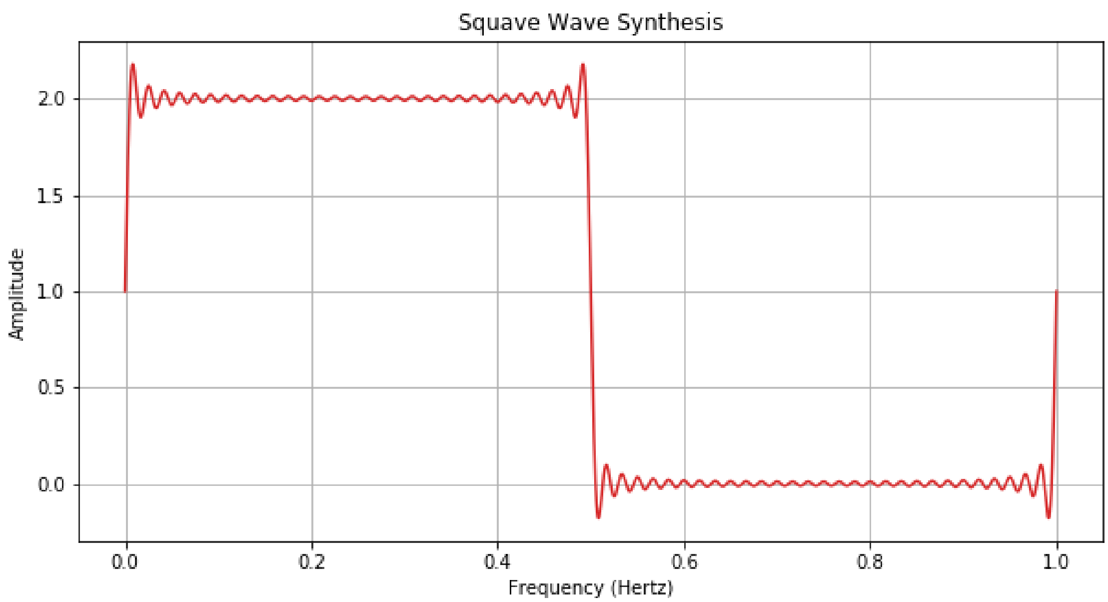
In [3]:
```
# Creates figure 1 and its subplot
fig1, ax1 = plt.figure(figsize= (10,5)), plt.subplot()
ax1.plot(t, ffs1(n1))
ax1.set(xlabel= 'Frequency (Hertz)', ylabel= 'Amplitude',
        title= 'Squave Wave Synthesis');
ax1.grid(True)
```



0.0.4 RUN 2: Plot of the fourier series in the frequency domain [Fast Fourier Transform (FFT)]

- **Step 1: Plot Fourier Series w/ 20 additional members (waves)**

```
In [4]: # Creates figure 2 and its subplot
        fig2, ax2 = plt.figure(figsize= (10,5)), plt.subplot();
        ax2.plot(t, ffs2(n2), color= 'tab:red')
        ax2.set(xlabel= 'Frequency (Hertz)', ylabel= 'Amplitude',
                title= 'Squave Wave Synthesis');
        ax2.grid(True)
```



- **Step 2: Change the numbers F_S and f_0 and observe their influence.**
  - This is a scaling problem. By changing f_0, you must change F_S by the same scaling.
  - f_0 was assigned to be scaled by 100. $f_0 = 100\ Hz$
  - Therefore, F_S must also be scaled by 100. $F_S = 10000\ Hz$
  - Shown below:

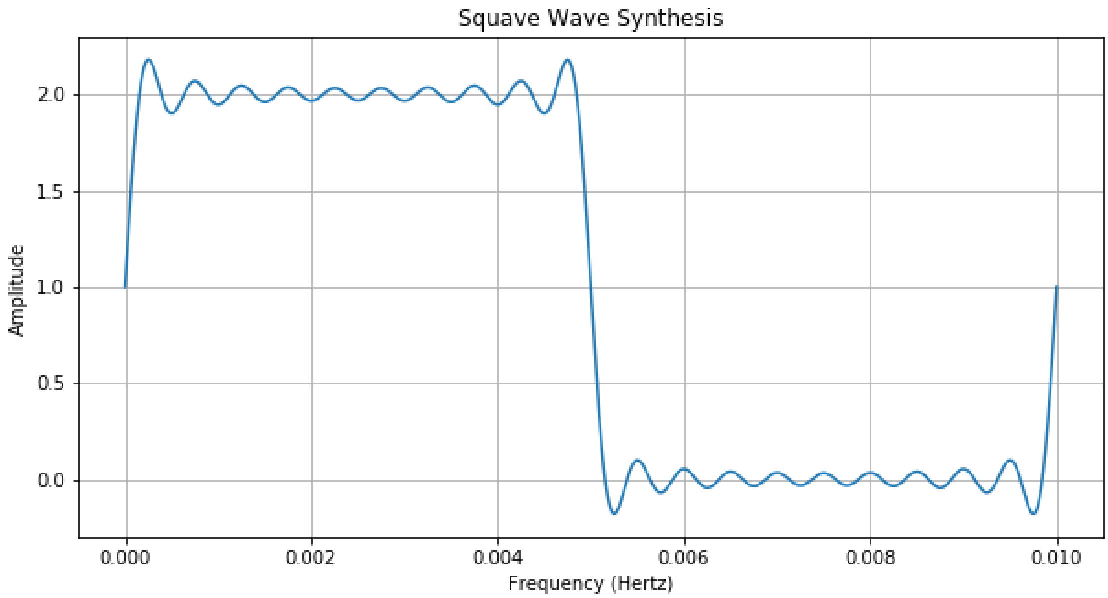
Code

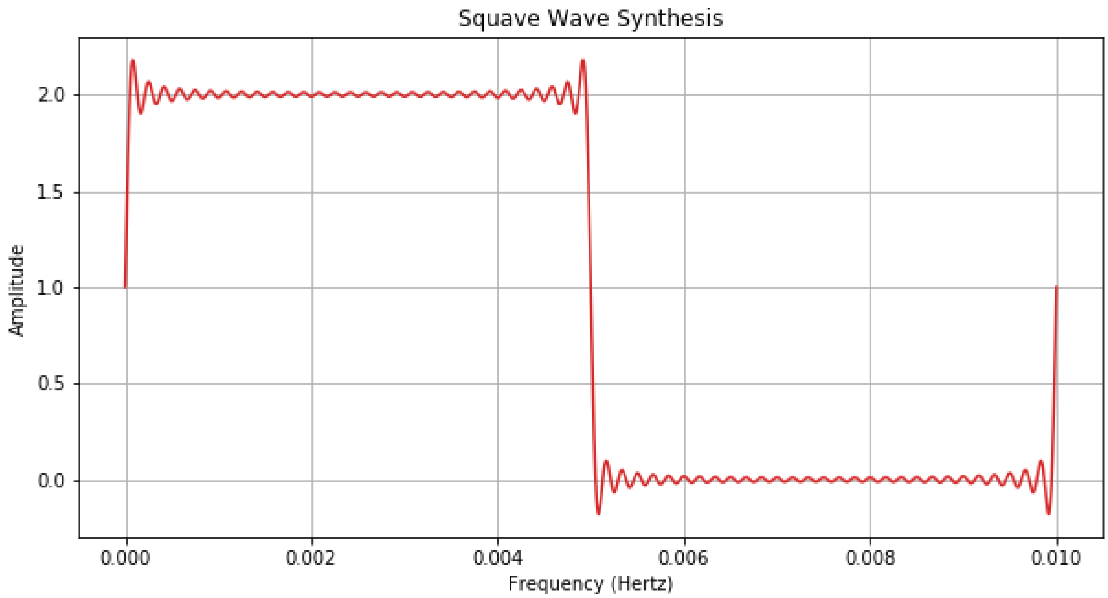
```
In [5]: # These variables are used to create the fourier series
        # Start and Stop indicates my domain for my sampling frequency [samplingfreq; (F_S)],
        # n1, n2 are the amount of harmonics I want.
        start, stop, n1, n2, samplingfreq = 0, 100, 9, 29, 10000
```

```
# 't' is for time, and is used to create my 100 Hz time vector
t = np.arange(start, stop, .001) / samplingfreq

# 'A' represents the amplitude 2 volts
A = 2

# 'fundamental' is the DC component of the Fourier Series
fundamental = A/2

# 'signalfreq' is the Signal Frequency (f_0)
signalfreq = 100

# 'omega' is the Angular Velocity (w_0)
omega = 2 * np.pi * signalfreq

# Lambda function
template = lambda p: ((2*A)/(np.pi*(2*p+1))) * np.sin((2*p+1) * omega * t)

# harmonics1, harmonics2 are AC component of the Fourier series
harmonics1 = sum([template(p) for p in range(n1+1)])
harmonics2 = sum([template(p) for p in range(n2+1)])

# ffs1, ffs2 are the fourier series
ffs1 = lambda n: (fundamental + harmonics1)
ffs2 = lambda w: (fundamental + harmonics2)
```

Plots

In [6]:
```
# Creates figure and its subplots
# Subplot 1
fig4, ax4 = plt.figure(figsize= (10,5)), plt.subplot()
ax4.plot(t, ffs1(n1))
ax4.set(xlabel= 'Frequency (Hertz)', ylabel= 'Amplitude',
        title= 'Squave Wave Synthesis')
ax4.grid(True)

# Subplot 2
fig5, ax5 = plt.figure(figsize= (10,5)), plt.subplot()
ax5.plot(t, ffs2(n2), color= 'tab:red')
ax5.set(xlabel= 'Frequency (Hertz)', ylabel= 'Amplitude',
        title= 'Squave Wave Synthesis');
ax5.grid(True)
```





- **Step 3: Using the original signal from RUN 1, observer the frequency spectrum of the signal as follows:**

Code

```
In [7]: # These variables are used to create the fourier series
        # Start and Stop indicates my domain for my sampling frequency [samplingfreq; (F_S)],
```

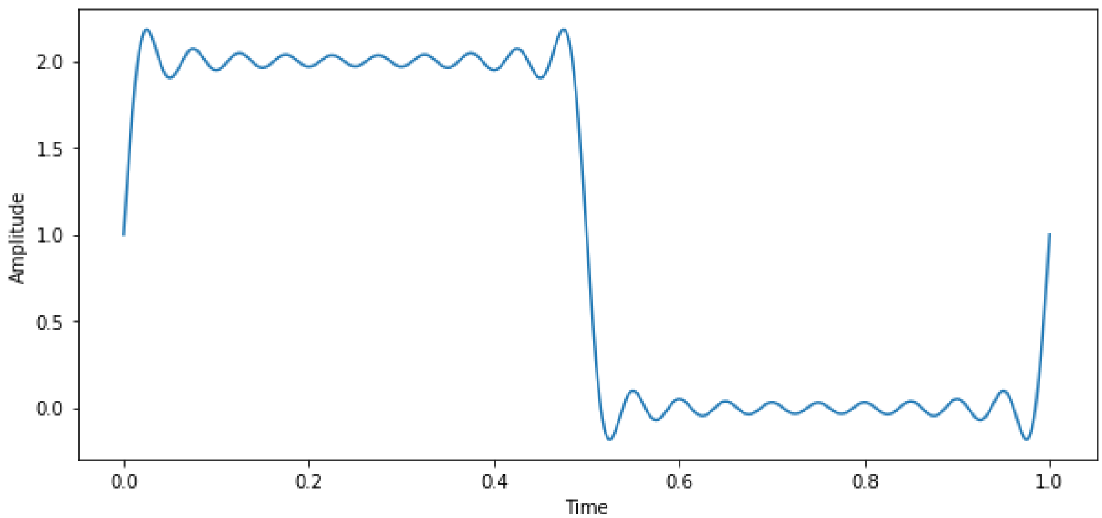
```
# n1, n2 are the amount of harmonics I want.
start, stop, n1, n2, samplingfreq = 0, 100, 9, 29, 100

# 't' is for time, and is used to create my 100 Hz time vector
t = np.arange(start, stop, .001) / samplingfreq

# 'A' represents the amplitude 2 volts
A = 2

# 'fundamental' is the DC component of the Fourier Series
fundamental = A/2

# 'signalfreq' is the Signal Frequency (f_0)
signalfreq = 1

# 'omega' is the Angular Velocity (w_0)
omega = 2 * np.pi * signalfreq

# Lambda function
template = lambda p: ((2*A)/(np.pi*(2*p+1))) * np.sin((2*p+1) * omega * t)

# harmonics1, harmonics2 are AC component of the Fourier series
harmonics1 = sum([template(p) for p in range(n1+1)])

# ffs1, ffs2 are the fourier series
ffs1 = lambda n: (fundamental + harmonics1)
```

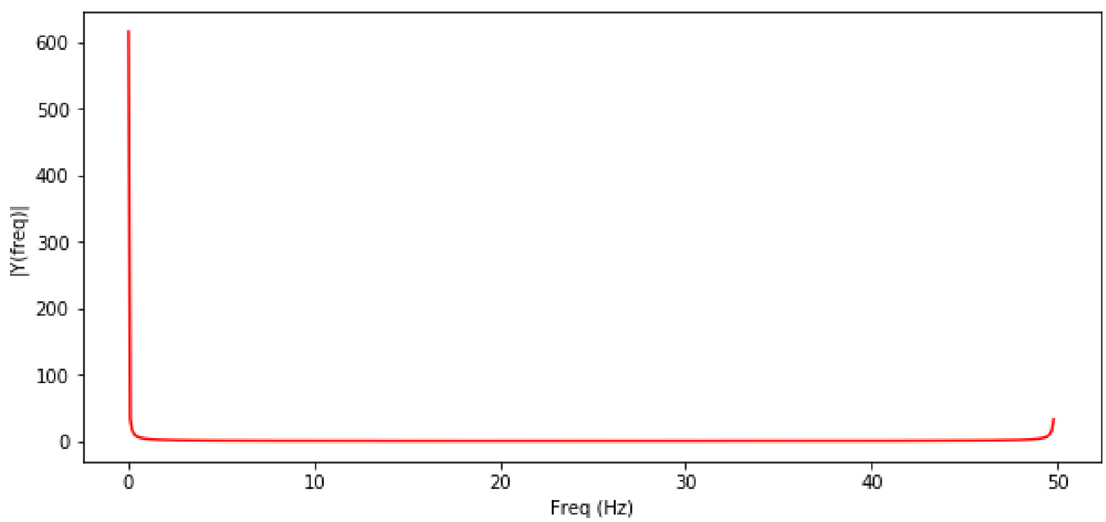
In [8]:
```
# Creating the Fast Fourier Transform
fft1 = fft(np.array(ffs1(n1)), 512)
frq = np.linspace(0, 255, 512)/256 * (samplingfreq/2)
```

```
/Users/Chris/anaconda/lib/python3.6/site-packages/scipy/fftpack/basic.py:153: FutureWarning: Us
  x = x[index]
```

Plot

In [9]:
```
fig, ax = plt.subplots(2, 1, figsize= (10, 10))
ax[0].plot(t,ffs1(n1))
ax[0].set_xlabel('Time')
ax[0].set_ylabel('Amplitude')
ax[1].plot(frq,abs(fft1),'r') # plotting the spectrum
ax[1].set_xlabel('Freq (Hz)')
ax[1].set_ylabel('|Y(freq)|');
```





0.0.5 Lab Questions/Requirements

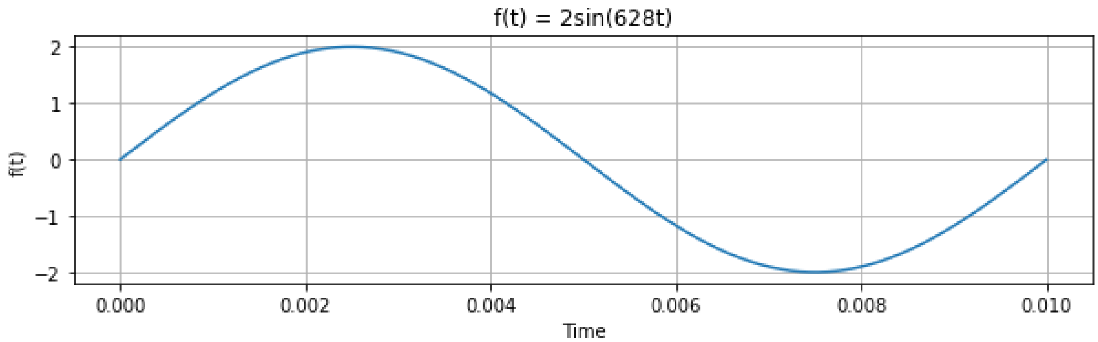
1. **Depict signals $2\sin(628\mathbf{t})$ and $2\cos(628\mathbf{t})$ both in the time domain and the frequency domain. In the frequency domain, show both axes with angular velocity (ω) and frequency (f)**

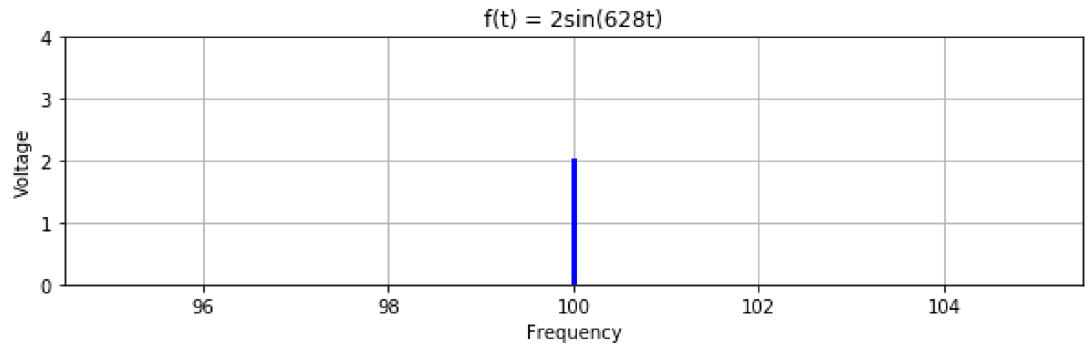
CODE

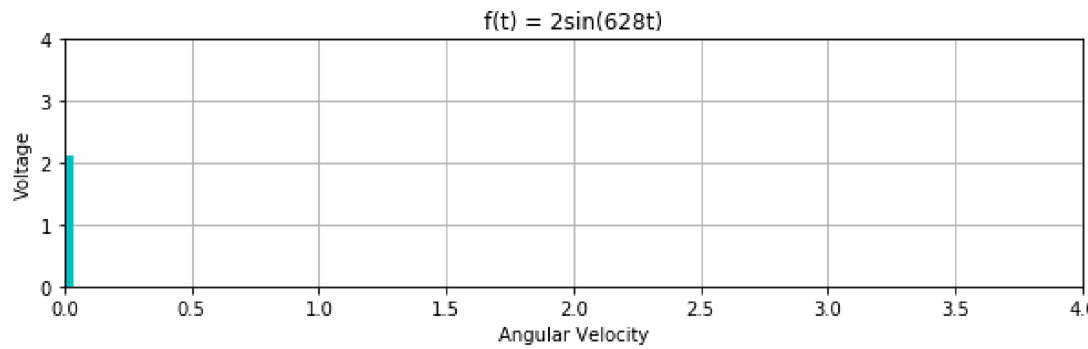
```
In [10]: #### Creates an array with values for the functions 2sin(628t) and 2cos(628t)
         start, stop, n1, n2, samplingfreq = 0, 100, 9, 29, 10000
         t = np.arange(start, stop, .001) / samplingfreq
         sine = 2 * np.sin(628 * t)
         cosine = 2 * np.cos(628 * t)
```

Plot for $2\sin(628 * t)$

```
In [11]: # Subplot 1

         fig8, ax8 = plt.figure(figsize= (10, 10)), plt.subplot(3, 1, 1)
         ax8.plot(t, sine)
         ax8.set(xlabel = 'Time', ylabel= 'f(t)', title= 'f(t) = 2sin(628t)')
         ax8.grid(True)

         # Subplot 2

         bx8 = plt.subplot(3, 1, 2)
         bx8.plot([100, 100],[0, 2], color= 'b', lw= 3)
         bx8.set(xlabel = 'Frequency', ylabel= 'Voltage',
                 title= 'f(t) = 2sin(628t)',
                 ylim = (0, 4))
         bx8.grid(True)

         # Subplot 3

         cx8 = plt.subplot(3, 1, 3)
         cx8.plot([0, 0],[0, 2], color= 'c', lw= 9)
         cx8.set(xlabel = 'Angular Velocity', ylabel= 'Voltage',
                 title= 'f(t) = 2sin(628t)',
                 xlim= (0, 4), ylim = (0, 4))
         cx8.grid(True)
         plt.subplots_adjust(hspace=0.5)
```



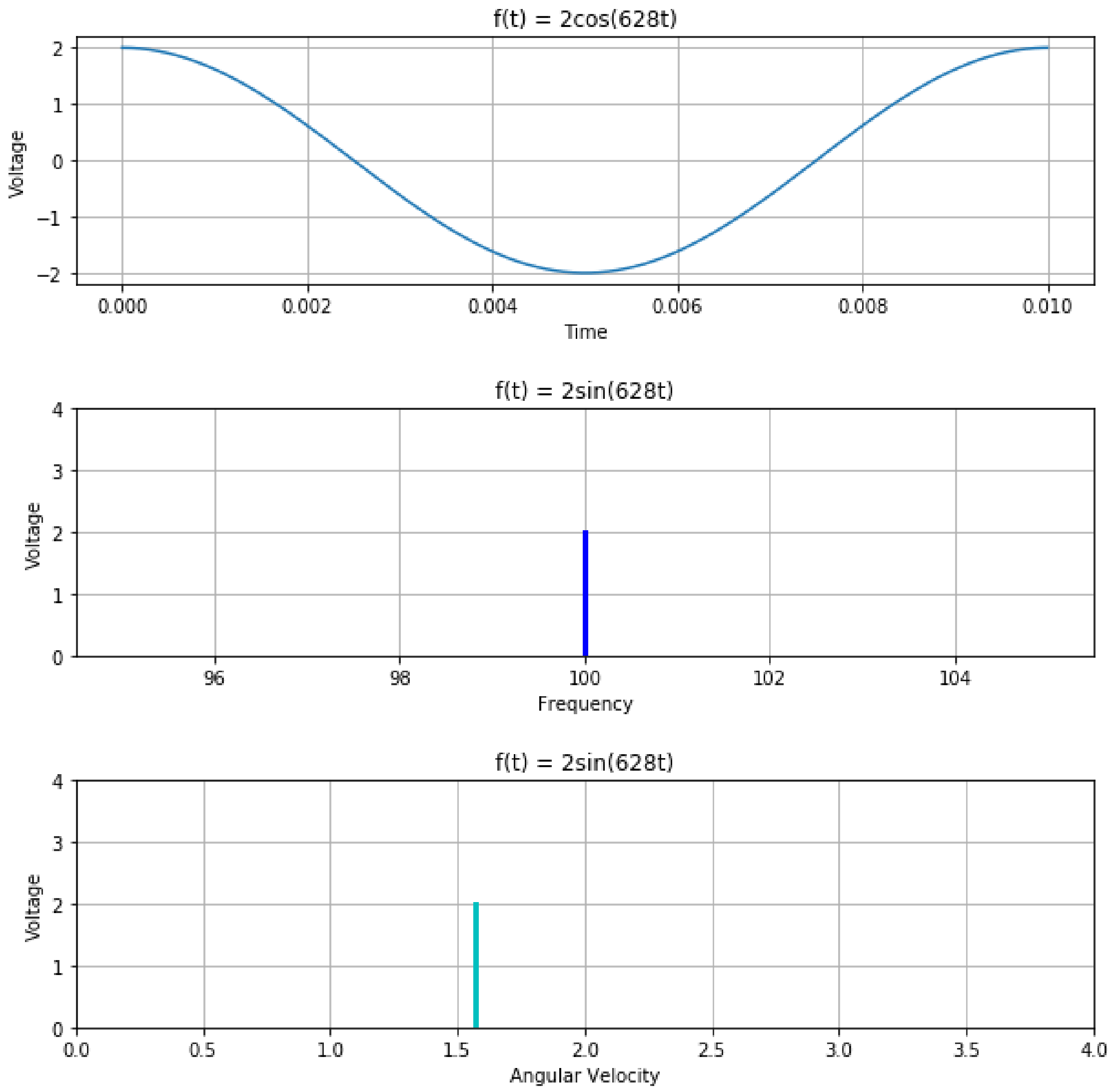




Plot for $2\cos(628 * t)$

In [12]:
```
# Subplot 1

fig9, ax9 = plt.figure(figsize= (10, 10)), plt.subplot(3, 1, 1)
ax9.plot(t, cosine)
ax9.set(xlabel = 'Time', ylabel= 'Voltage', title= 'f(t) = 2cos(628t)')
ax9.grid(True)

# Subplot 2

bx9 = plt.subplot(3, 1, 2)
bx9.plot([100, 100],[0, 2], color= 'b', lw= 3)
bx9.set(xlabel = 'Frequency', ylabel= 'Voltage', title= 'f(t) = 2sin(628t)',
```

```
        ylim= (0, 4))
bx9.grid(True)

# Subplot 3

cx9 = plt.subplot(3, 1, 3)
cx9.plot([np.pi/2, np.pi/2],[0, 2], color= 'c', lw= 3)
cx9.set(xlabel = 'Angular Velocity', ylabel= 'Voltage', title= 'f(t) = 2sin(628t)',
        xlim= (0, 4), ylim = (0, 4))
cx9.grid(True)
plt.subplots_adjust(hspace=0.5)
```



2. **Square Wave Analysis: using *Python* generate a square wave with *A* = *2 Volts* and *T* = *1 ms*.**

- Check RUN 1, step 1 and associated code for the results

3. **Based on the experimental results, how do the number of harmonic terms included in the square wave synthesis influence the waveform? Show waveform w/ different number of series terms to explain your answer.**

   - Basically, the more harmonics you have the more square your wave looks.
   - Check RUN 2 step 1 for the plot demonstrating this phenomenon.

4. **Vary F_S and f_0 and observe their influence on the waveform. Show figure and explain.**

   - Check figure in RUN 2, Step 2 to observe the influence of varying F_S and f_0 on the waveform
   - As aforementioned, by changing the signal frequency (f_0) you must also change the sampling frequency (F_S) by the equivalent scale in order to obtain a similar waveform shape as the previous unscaled waveform. Anything else produces a quirky wavefore or more as shown below:

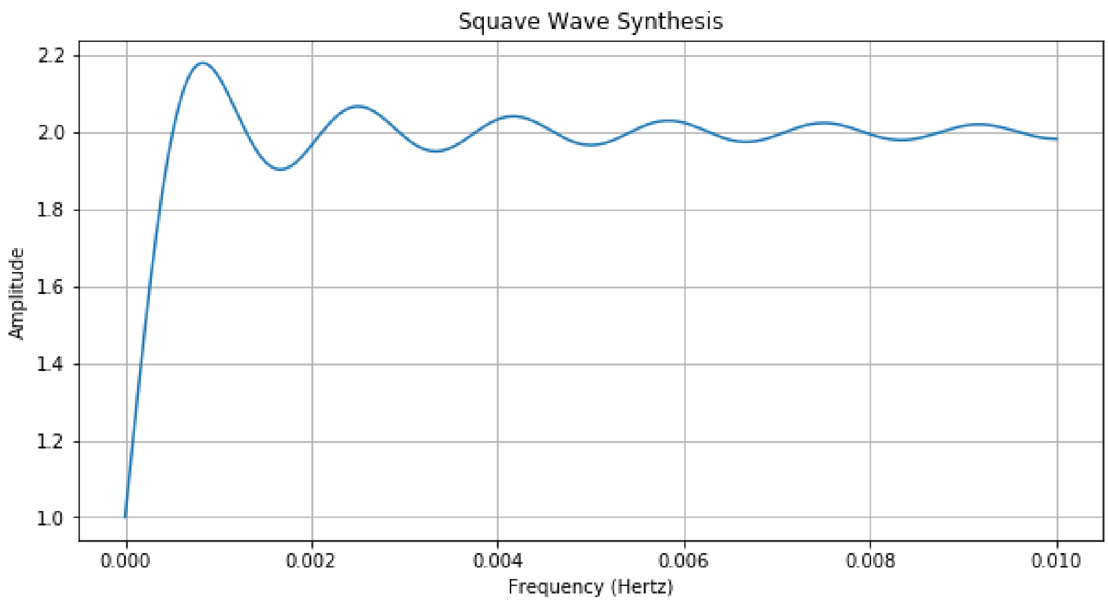
Code

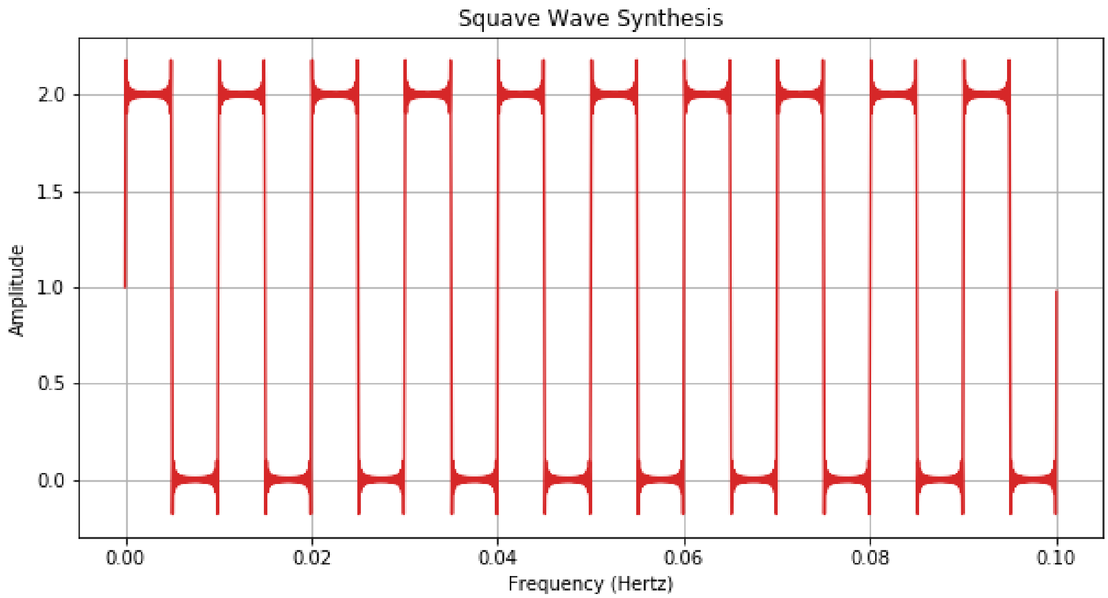
In [13]:
```
# These variables are used to create the fourier series
# Start and Stop indicates my domain for my sampling frequency [samplingfreq; (F_S)],
# n1, n2 are the amount of harmonics I want.
start, stop, n1, n2, samplingfreq1, samplingfreq2 = 0, 100, 29, 29, 10000, 1000

# 't' is for time, and is used to create my 100 Hz time vector
t1 = np.arange(start, stop, .001) / samplingfreq1
t2 = np.arange(start, stop, .001) / samplingfreq2

# 'A' represents the amplitude 2 volts
A = 2

# 'fundamental' is the DC component of the Fourier Series
fundamental = A/2

# 'signalfreq' is the Signal Frequency (f_0)
signalfreq1, signalfreq2 = 10, 100

# 'omega' is the Angular Velocity (w_0)
omega1 = 2 * np.pi * signalfreq1
omega2 = 2 * np.pi * signalfreq2

# Lambda function
template = lambda p, t, omega: ((2*A)/(np.pi*(2*p+1))) * np.sin((2*p+1) * omega * t)

# harmonics1, harmonics2 are AC component of the Fourier series
harmonics1 = sum([template(p, t1, omega1) for p in range(n1+1)])
harmonics2 = sum([template(p, t2, omega2) for p in range(n2+1)])
```

```
# ffs1, ffs2 are the fourier series
ffs1 = lambda n: (fundamental + harmonics1)
ffs2 = lambda w: (fundamental + harmonics2)
```

Plot

In [14]:
```
# Creates figure and its subplots
# Subplot 1
fig6, ax6 = plt.figure(figsize= (10,5)), plt.subplot()
ax6.plot(t1, ffs1(n1))
ax6.set(xlabel= 'Frequency (Hertz)', ylabel= 'Amplitude',
        title= 'Squave Wave Synthesis')
ax6.grid(True)

# Subplot 2
fig7, ax7 = plt.figure(figsize= (10,5)), plt.subplot()
ax7.plot(t2, ffs2(n2), color= 'tab:red')
ax7.set(xlabel= 'Frequency (Hertz)', ylabel= 'Amplitude',
        title= 'Squave Wave Synthesis');
ax7.grid(True)
```




Squave Wave Synthesis
Amplitude
2.0
1.5
1.0
0.5
0.0
0.00
0.02
0.04
0.06
0.08
0.10
Frequency (Hertz)