Signals and Systems – Problem Set 6

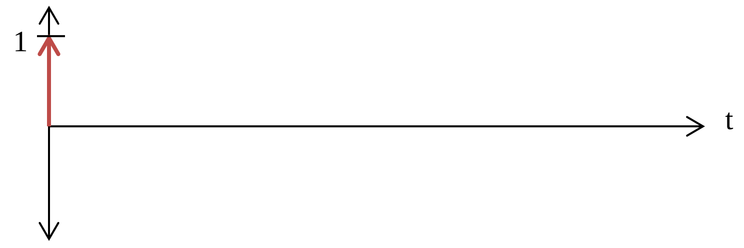
Mika Ichiki-Welches

1. If you generate an impulse in a system, it will produce in result an impulse response. In the example of the gunshot in the shooting range, the gunshot is the impulse, the shooting range is the system, and the impulse response is the gunshot noise heard in the room. In order to characterize the properties of the system you can find its transfer function. The transfer function is a spectrum that represents the system response, or more specifically, the amplitude multiplier and phase shift it allots at each frequency. If you have the impulse response, you can find its transfer function by taking its DFT to analyze it at each frequency. Once you have the transfer function, you can apply it to different impulses, like the violin sound, to see how the system (the shooting range in this case) would affect them.
2. The model $y(t) = \frac{1}{2}x(t-1) + \frac{1}{4}x(t-10)$ takes in an input x(t) and adds two scaled, shifted versions of the signal to produce an output, y(t). One version is scaled by ½ and shifted by a phase offset of 1, and the other is scaled by ¼ and shifted by 10. The system that applies this transfer function could be described as an echo chamber. Perhaps the impulse x(t) bounces off of a nearby wall and takes 1 second to reach the recording device, halving in amplitude by the time that it does. It may also bounce off of another wall that's farther away, and reach the recording device 10 seconds after the impulse is made, now at an amplitude 4 times smaller than the original impulse. The expression for the impulse response is represented as h(t), where δ(t) is the impulse. The impulse response, h(t), can be expressed as:

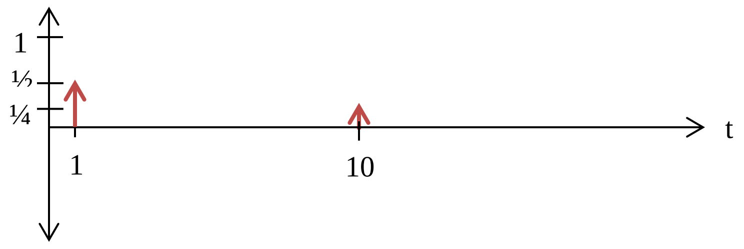
$$h(t) = \frac{1}{2}\delta(i-1) + \frac{1}{4}\delta(i-10)$$

where N is the length of x.

If x(t) is:

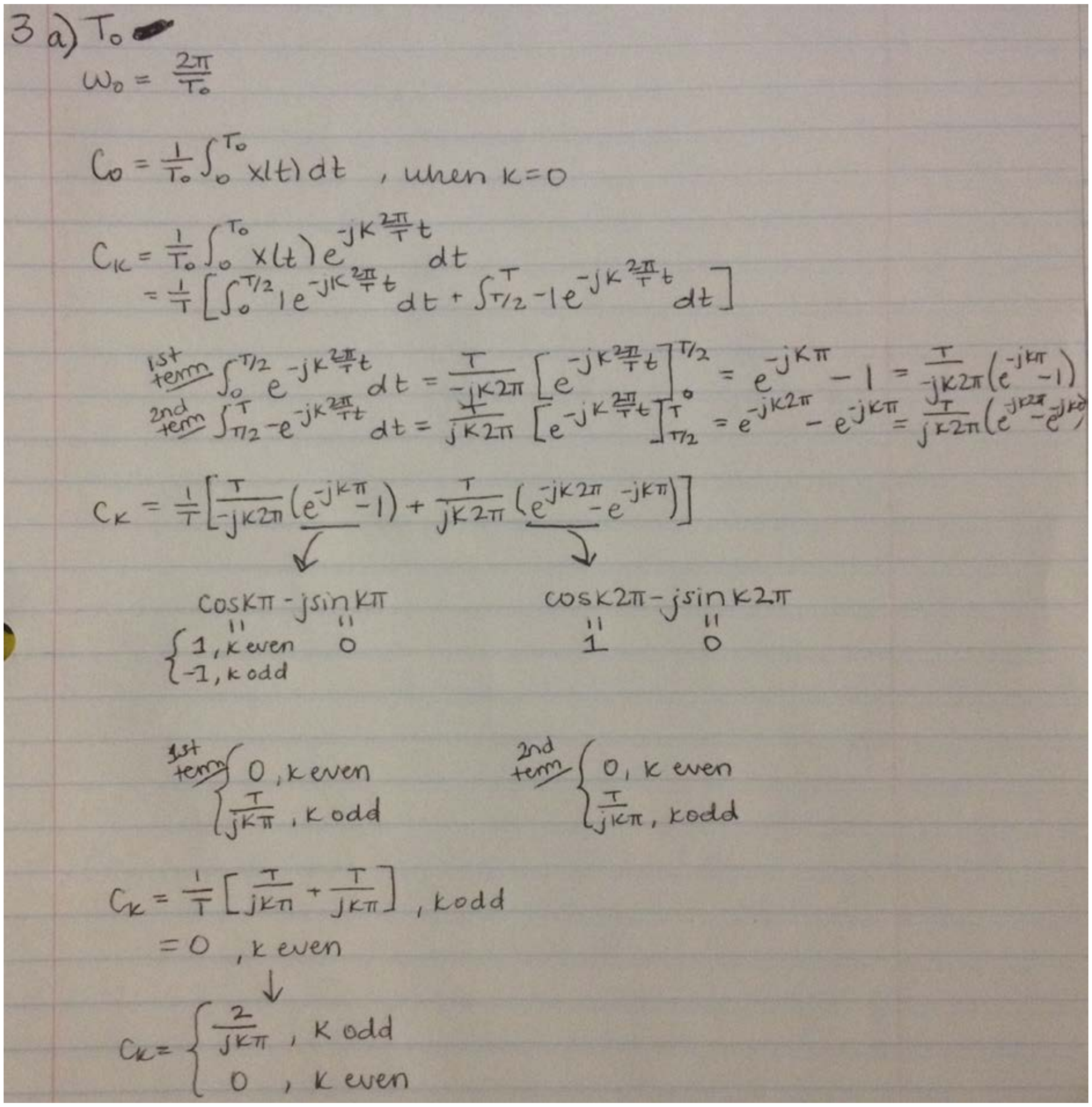


Then the impulse response y(t) becomes:



3.

a. Fourier series representation of a square wave:



3 a) T_0

$\omega_0 = \frac{2\pi}{T_0}$

$C_0 = \frac{1}{T_0}\int_0^{T_0} x(t)\,dt$, when $k=0$

$C_k = \frac{1}{T_0}\int_0^{T_0} x(t)\, e^{-jk\frac{2\pi}{T}t}\,dt$

$= \frac{1}{T}\left[\int_0^{T/2} 1 e^{-jk\frac{2\pi}{T}t}\,dt + \int_{T/2}^{T} -1 e^{-jk\frac{2\pi}{T}t}\,dt\right]$

1st term $\int_0^{T/2} e^{-jk\frac{2\pi}{T}t}\,dt = \frac{T}{-jk2\pi}\left[e^{-jk\frac{2\pi}{T}t}\right]_0^{T/2} = e^{-jk\pi} - 1 = \frac{T}{-jk2\pi}(e^{-jk\pi}-1)$

2nd term $\int_{T/2}^{T} -e^{-jk\frac{2\pi}{T}t}\,dt = \frac{T}{jk2\pi}\left[e^{-jk\frac{2\pi}{T}t}\right]_{T/2}^{T} = e^{-jk2\pi} - e^{-jk\pi} = \frac{T}{jk2\pi}(e^{-jk2\pi} - e^{-jk\pi}$

$C_k = \frac{1}{T}\left[\frac{T}{-jk2\pi}(\underline{e^{-jk\pi}}-1) + \frac{T}{jk2\pi}(\underline{e^{-jk2\pi}} - e^{-jk\pi})\right]$

$\cos k\pi - j\sin k\pi$ with $\cos k\pi = \begin{cases} 1, & k \text{ even} \\ -1, & k \text{ odd}\end{cases}$, $\sin k\pi = 0$

$\cos k2\pi - j\sin k2\pi$ with $\cos k2\pi = 1$, $\sin k2\pi = 0$

1st term $\begin{cases} 0, & k \text{ even} \\ \frac{T}{jk\pi}, & k \text{ odd}\end{cases}$

2nd term $\begin{cases} 0, & k \text{ even} \\ \frac{T}{jk\pi}, & k \text{ odd}\end{cases}$

$C_k = \frac{1}{T}\left[\frac{T}{jk\pi} + \frac{T}{jk\pi}\right]$, k odd

$= 0$, k even

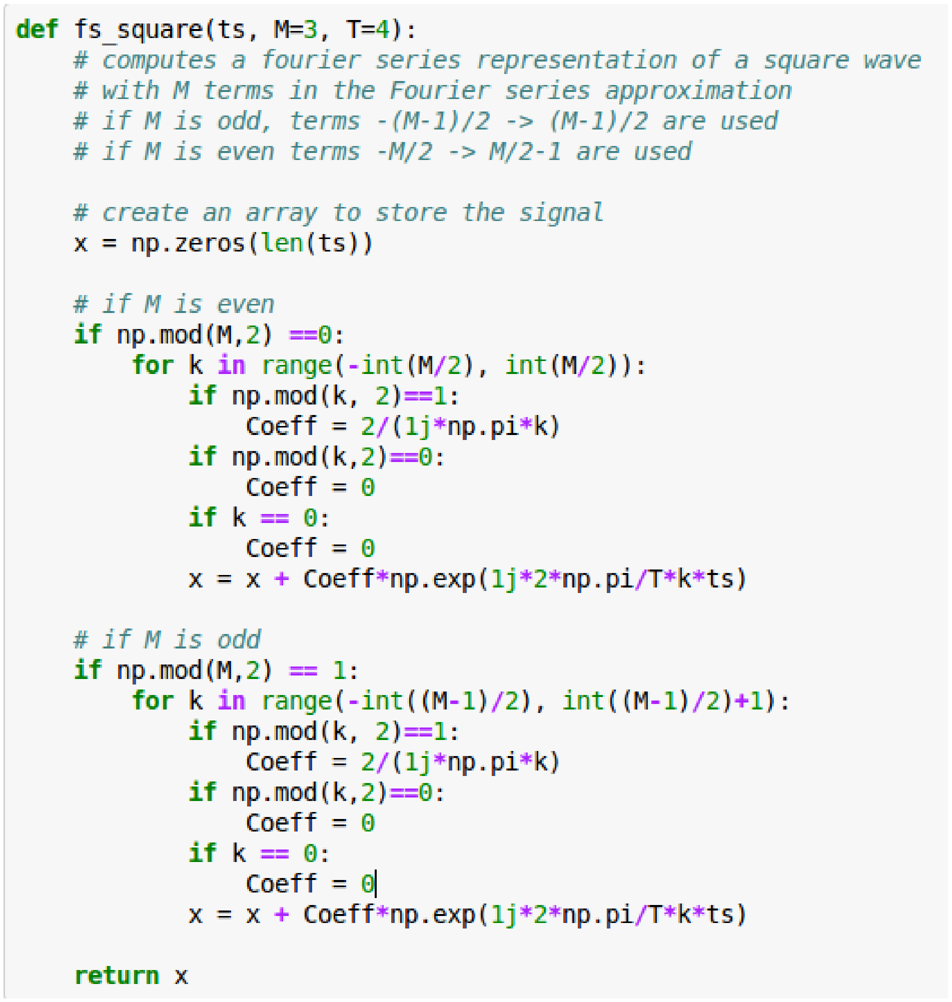
$\downarrow$

$C_k = \begin{cases} \frac{2}{jk\pi}, & k \text{ odd} \\ 0, & k \text{ even}\end{cases}$

which results in a Fourier series of:

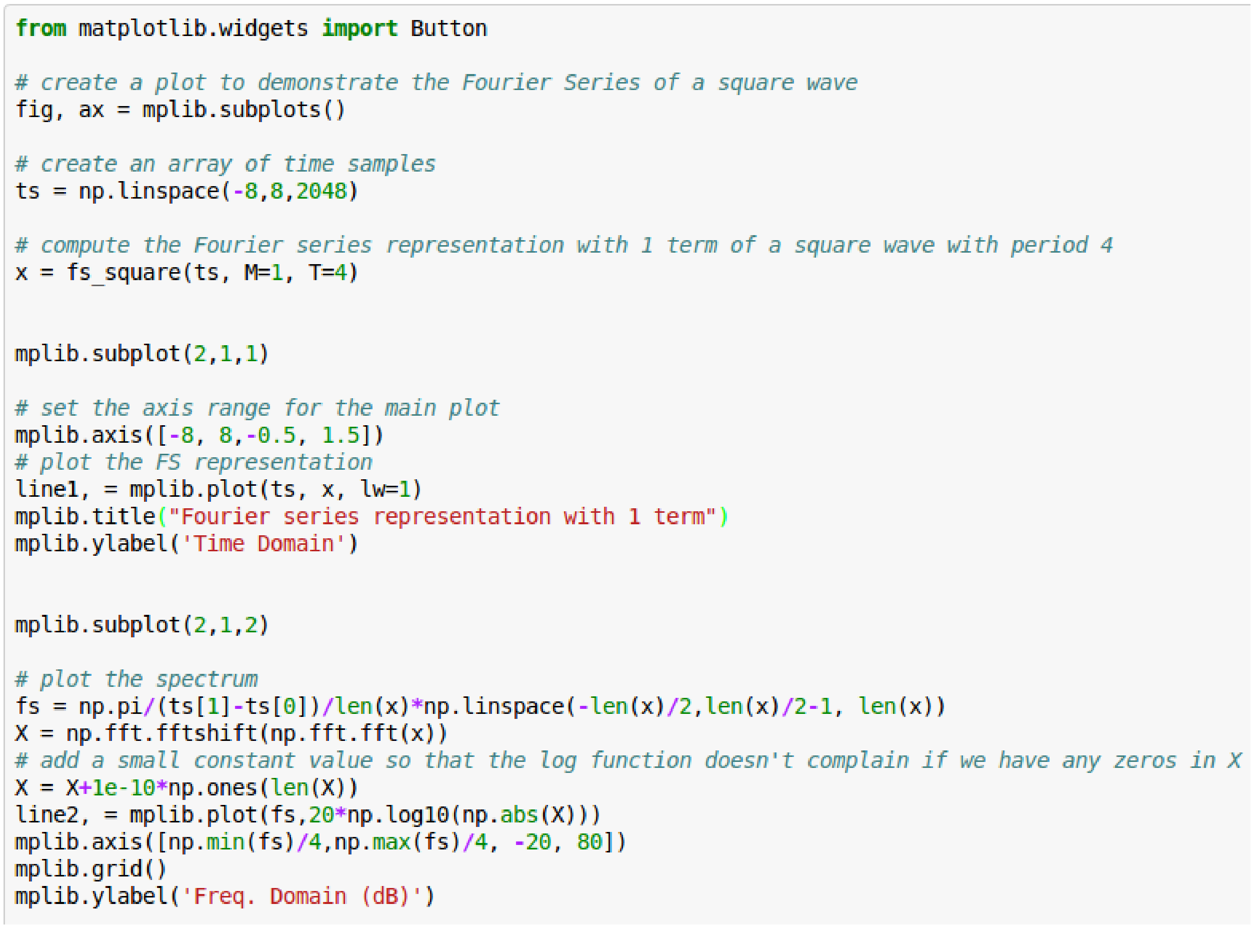
$$\tilde{x}_k(t) = \begin{cases} \sum_{k=-K}^{K} \frac{2}{jk\pi} e^{j\frac{2\pi}{T}kt}, & k\ odd \\ 0, & k\ even \end{cases}$$

b. Code for a square wave Fourier series function:



```
def fs_square(ts, M=3, T=4):
    # computes a fourier series representation of a square wave
    # with M terms in the Fourier series approximation
    # if M is odd, terms -(M-1)/2 -> (M-1)/2 are used
    # if M is even terms -M/2 -> M/2-1 are used

    # create an array to store the signal
    x = np.zeros(len(ts))

    # if M is even
    if np.mod(M,2) ==0:
        for k in range(-int(M/2), int(M/2)):
            if np.mod(k, 2)==1:
                Coeff = 2/(1j*np.pi*k)
            if np.mod(k,2)==0:
                Coeff = 0
            if k == 0:
                Coeff = 0
            x = x + Coeff*np.exp(1j*2*np.pi/T*k*ts)

    # if M is odd
    if np.mod(M,2) == 1:
        for k in range(-int((M-1)/2), int((M-1)/2)+1):
            if np.mod(k, 2)==1:
                Coeff = 2/(1j*np.pi*k)
            if np.mod(k,2)==0:
                Coeff = 0
            if k == 0:
                Coeff = 0
            x = x + Coeff*np.exp(1j*2*np.pi/T*k*ts)

    return x
```

Code to plot the square wave Fourier series:

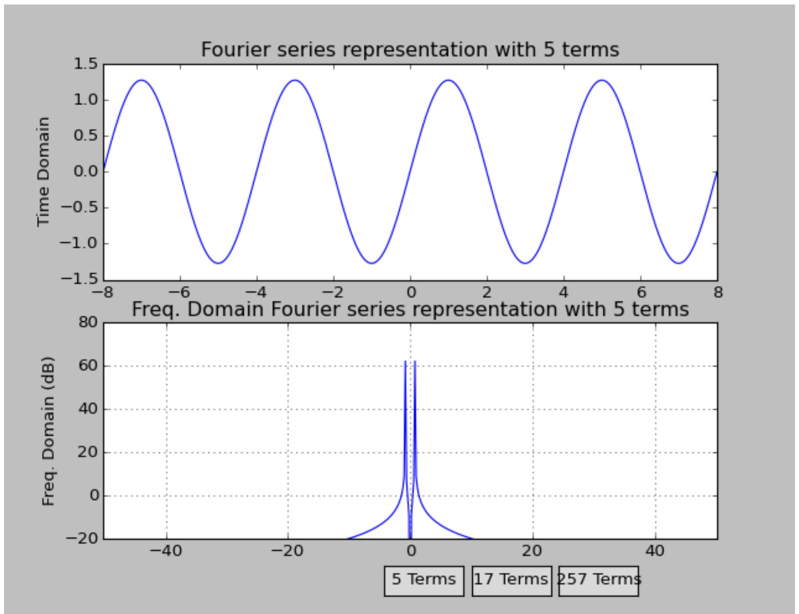


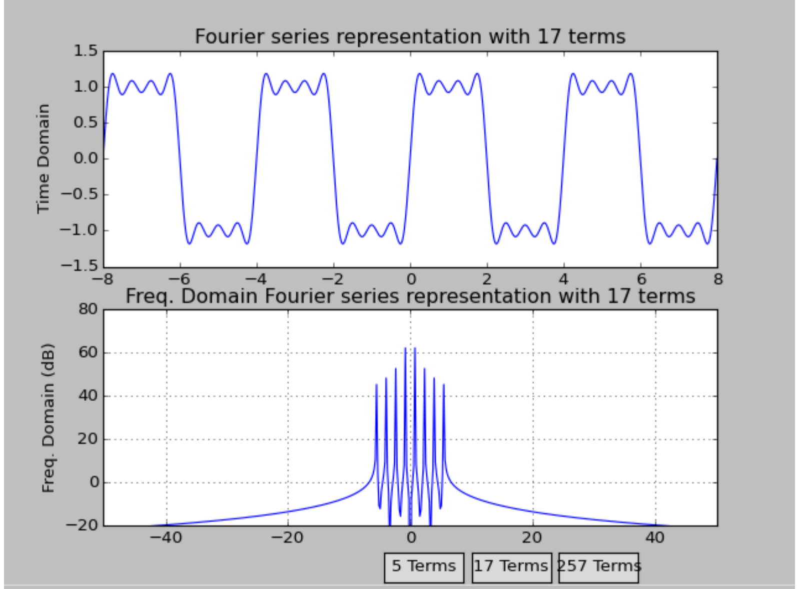
```python
from matplotlib.widgets import Button

# create a plot to demonstrate the Fourier Series of a square wave
fig, ax = mplib.subplots()

# create an array of time samples
ts = np.linspace(-8,8,2048)

# compute the Fourier series representation with 1 term of a square wave with period 4
x = fs_square(ts, M=1, T=4)


mplib.subplot(2,1,1)

# set the axis range for the main plot
mplib.axis([-8, 8,-0.5, 1.5])
# plot the FS representation
line1, = mplib.plot(ts, x, lw=1)
mplib.title("Fourier series representation with 1 term")
mplib.ylabel('Time Domain')


mplib.subplot(2,1,2)

# plot the spectrum
fs = np.pi/(ts[1]-ts[0])/len(x)*np.linspace(-len(x)/2,len(x)/2-1, len(x))
X = np.fft.fftshift(np.fft.fft(x))
# add a small constant value so that the log function doesn't complain if we have any zeros in X
X = X+1e-10*np.ones(len(X))
line2, = mplib.plot(fs,20*np.log10(np.abs(X)))
mplib.axis([np.min(fs)/4,np.max(fs)/4, -20, 80])
mplib.grid()
mplib.ylabel('Freq. Domain (dB)')
```

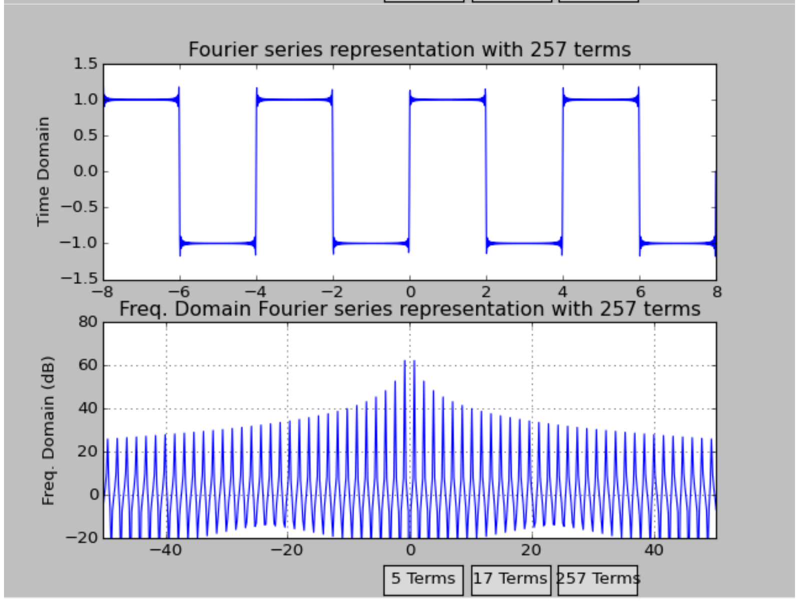
```python
# this is the function that gets called whenever the slider is changed
def on_change(M):

    mplib.subplot(211)
    mplib.title("Fourier series representation with %d terms" % (M) )
    mplib.subplot(212)
    mplib.title("Freq. Domain Fourier series representation with %d terms" % (M) )

    # compute a FS representation for the number of FS terms from the slider
    x = fs_square(ts, M, T=4)
    # refresh the line
    line1.set_ydata(x)
    X = np.fft.fftshift(np.fft.fft(x))
    # add a small constant value so that the log function doesn't complain if we have any zeros in X
    X = X+1e-10*np.ones(len(X))
    line2.set_ydata(20*np.log10(np.abs(X)))
    # redraw to update
    mplib.draw()
```

```python
class PlotFS:

    M_257 = 257
    M_17 = 17
    M_5 = 5

    # this method maximizes the number of FS terms
    def maximize(self, event):
        self.M = self.M_257
        on_change(self.M)

    # this method makes the number of FS terms equal to 17
    def middle(self, event):
        self.M = self.M_17
        on_change(self.M)

    # this method makes the number of FS terms equal to 5
    def minimize(self, event):
        self.M = self.M_5
        on_change(self.M)

# this object handles the button presses
callback = PlotFS()

# make the buttons
axmax = mplib.axes([0.70, 0.005, 0.1, 0.05])
axmid = mplib.axes([0.59, 0.005, 0.1, 0.05])
axmin = mplib.axes([0.48, 0.005, 0.1, 0.05])

bmin = Button(axmin, "%d Terms" % callback.M_5)
bmid = Button(axmid, "%d Terms" % callback.M_17)
bmax = Button(axmax, "%d Terms" % callback.M_257)

bmax.on_clicked(callback.maximize)
bmid.on_clicked(callback.middle)
bmin.on_clicked(callback.minimize)

mplib.show()
```

Square wave plots at M = 5, 17, and 257:





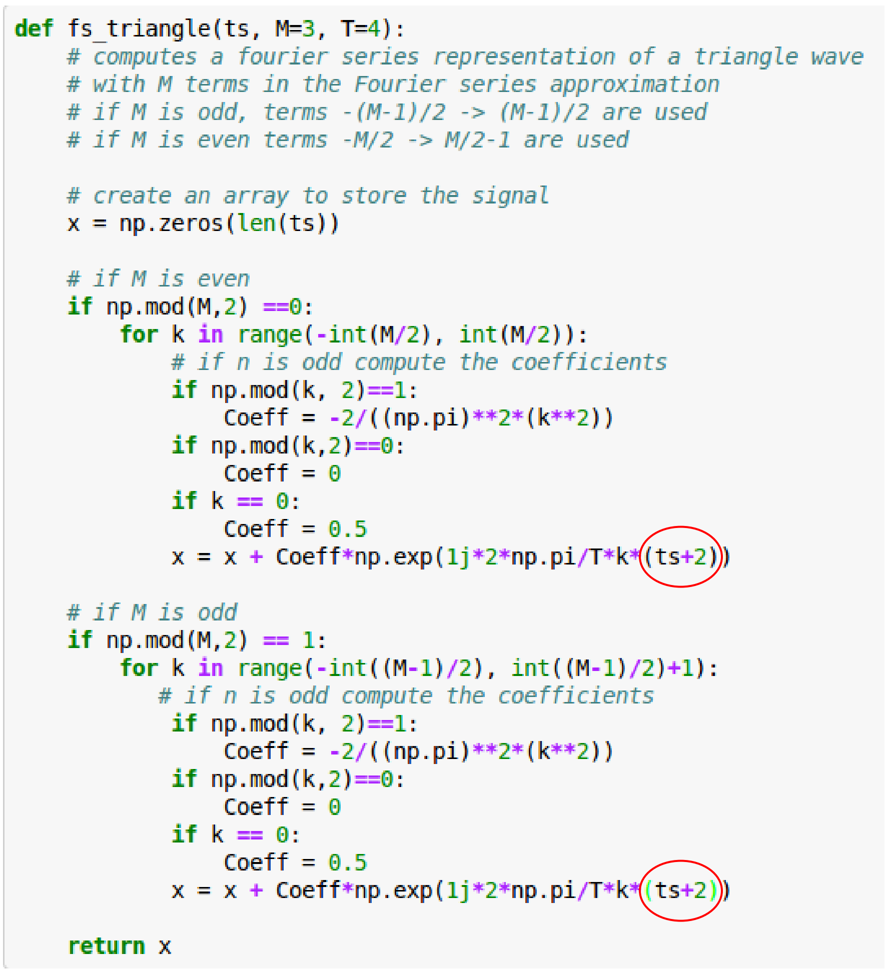


c. Even with 257 terms being used in the Fourier series of the square wave, the edges (or discontinuous points) of the square wave appear a bit pointed and darkened. This is because although I am using the maximum number of terms in my Fourier series function, the series is still only an approximation. It will never actually reach a true square wave unless the number of terms was infinite, because changing the state instantaneously from 0 to 1 or vice versa would require an infinite bandwidth.

4.

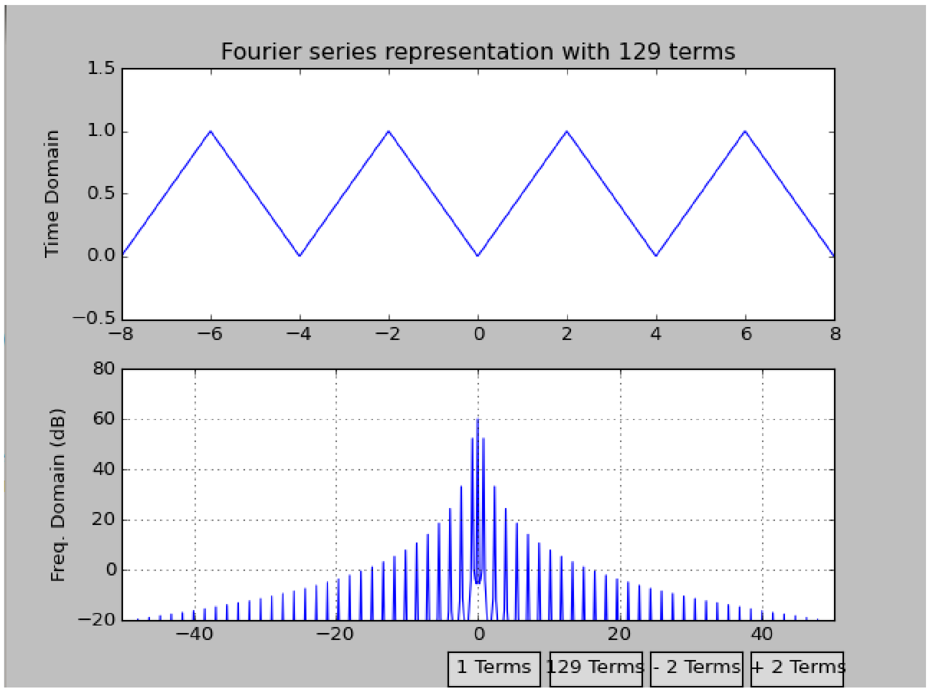
a. As the integral in a shifted signal still takes the same area under a periodic signal, the magnitude of the coefficient does not change. Rather, in defining the Fourier series representation, $x_k(t)$, the phase shift is accounted for in the exponential term. $C_k e^{j\frac{2\pi}{T}kt}$ becomes $C_k e^{j\frac{2\pi}{T}k(t-T_1)}$

b. Modified code for triangle wave Fourier series:

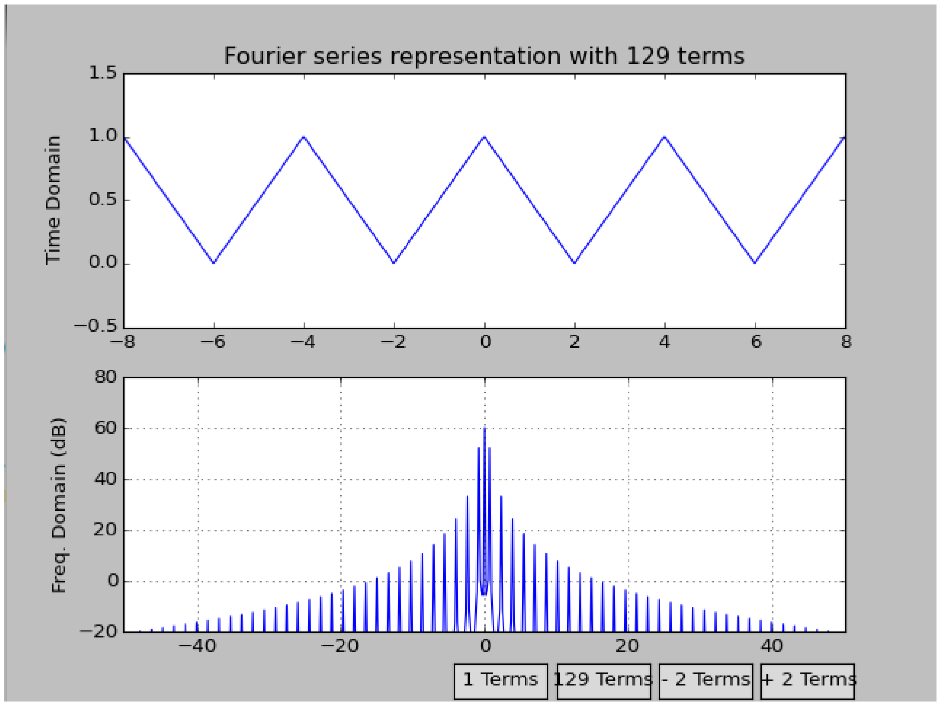


```
def fs_triangle(ts, M=3, T=4):
    # computes a fourier series representation of a triangle wave
    # with M terms in the Fourier series approximation
    # if M is odd, terms -(M-1)/2 -> (M-1)/2 are used
    # if M is even terms -M/2 -> M/2-1 are used

    # create an array to store the signal
    x = np.zeros(len(ts))

    # if M is even
    if np.mod(M,2) ==0:
        for k in range(-int(M/2), int(M/2)):
            # if n is odd compute the coefficients
            if np.mod(k, 2)==1:
                Coeff = -2/((np.pi)**2*(k**2))
            if np.mod(k,2)==0:
                Coeff = 0
            if k == 0:
                Coeff = 0.5
            x = x + Coeff*np.exp(1j*2*np.pi/T*k*(ts+2))

    # if M is odd
    if np.mod(M,2) == 1:
        for k in range(-int((M-1)/2), int((M-1)/2)+1):
           # if n is odd compute the coefficients
            if np.mod(k, 2)==1:
                Coeff = -2/((np.pi)**2*(k**2))
            if np.mod(k,2)==0:
                Coeff = 0
            if k == 0:
                Coeff = 0.5
            x = x + Coeff*np.exp(1j*2*np.pi/T*k*(ts+2))

    return x
```

Original triangle wave plot:



Triangle wave plot shifted by t = 2 (as seen in the code modification):


Fourier series representation with 129 terms
Time Domain
Freq. Domain (dB)
1 Terms
129 Terms
- 2 Terms
+ 2 Terms