Problem Set 6

Lindsey Vanderlyn

1) Explain why a gunshot convolved w/ a violin will sound as though the violin is being played in a shooting range.

When the gun is fired, it creates an impulse or a wave with roughly even energy at all frequency levels. This tests the system, in this case the shooting range for the frequency response at all frequency levels. Convolving a transform - the system response at each frequency - with a new sound / wave will apply the transfermation caused by the room to the new wave.

2) An echo chamber can - for the purpose of this question - be modeled as $y(t) = \frac{1}{2}x(t-1) + \frac{1}{4}x(t-10)$ where $x(t)$ is the input. Why does this act as an echo chamber? And what is a function & graph for this system's impulse response?

See back for math Answer

This equation does make sense for an echo chamber as the $y(t)$ signal is made up of the input with some fraction of time delay on the signal added back in, as if the input where to have bounced off a surface, or echoed back to be part of the signal again. As for the frequency response an equation for that would be:

$$y(t) = x * \boxed{h(t)} \; ? = \frac{1}{2}x(t-1) + \frac{1}{4}x(t-10)$$

freq $\mathcal{F}(y(F)) / \mathcal{F}(x) = \mathcal{F}(h(t))$ time $\mathcal{F}^{-1}$ $h(t)$

In order to actually solve for h, it

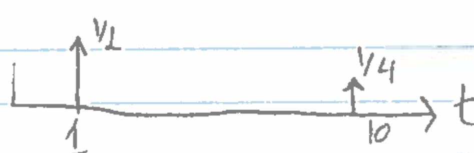
Seems like we need to convert the time based equation to the frequency domain so we could simply divide the system output (now in frequency domain) by the impulse used to generate it and the convert that back to the time domain where it would represent the system response. From inter

#2 Answer

Graph:

$$y(t) = x * h(t) \rightarrow h(t) = \tfrac{1}{2}\partial(t-1)+\tfrac{1}{4}\partial(t-10)$$

or an impulse at $t=1$ w/ amplitude $1/2$ and an impulse at $t=10$ w/ amplitude $1/4$

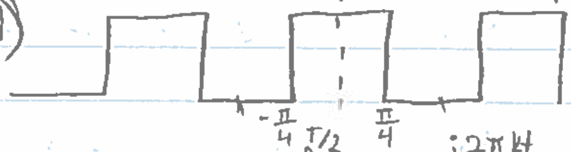


Edit:

Impulse response is an impulse of $1/2$ at $t=1$ and an impulse of $1/4$ at $t=10$ b/c when an impulse is added to the system, that is how it is changed

3) (Find the Fourier series representation of

a)



$$C_k = \frac{1}{T}\int_{-T/2}^{T/2} x(t)\, e^{\frac{-j2\pi kt}{T}}\, dt \qquad x(t) = \begin{cases} 1 & \text{if } -\frac{T}{4} \le t \le \frac{T}{4} \\ \text{else} \\ 0 \end{cases}$$

$$\boxed{\tilde{x}_K(t) = \sum_{k=-K}^{K} C_k\, e^{\frac{j2\pi kt}{T}}}$$

$$C_k = \frac{1}{T}\int_{-T/4}^{T/4} A\, e^{\frac{j2\pi kt}{T}}$$

$$\frac{1}{T}\, x(t)\, e^{-\frac{2\pi mt}{T}} = \sum_{k=-\infty}^{\infty} C_k\, e^{\frac{j2\pi kt}{T}}\, \frac{1}{T}\, e^{\frac{-2\pi}{T} jmt}$$

$$= \frac{1}{T}\sum_{k=-\infty}^{\infty} C_k\, e^{\frac{j2\pi}{T}(k-m)t}$$

$$\frac{1}{T}\int_{-T/4}^{T/4} A\, e^{\frac{-j2\pi mt}{T}}\, dt = \sum_{k=-\infty}^{\infty} \frac{C_k}{T}\int_{-T/2}^{T/2} e^{\frac{j2\pi}{T}(k-m)t}\, dt$$

where $T = \pi$

$$\boxed{\tilde{x}(t) = \sum_{k=-K}^{K} \frac{1}{T}\int_{-T/4}^{T/4} 1\, e^{\frac{-j2\pi kt}{T}}\, dt\, e^{\frac{j2\pi kt}{T}}}$$

$$C_k = \frac{1}{T}\cdot\frac{T}{-j2\pi kT}\cdot e^{-j2\pi kT/4} - \frac{1}{j2\pi k}\, e^{j2\pi kT/4}$$

$$C_k = \frac{1}{T}\cdot\frac{T}{\pi k}\cdot\left[\frac{1}{2j}\, e^{\frac{-j\,2\pi k\cancel{T}/4}{\cancel{T}}} - \frac{1}{2j}\, e^{\frac{j2\pi k\cancel{T}/4}{\cancel{T}}}\right]$$

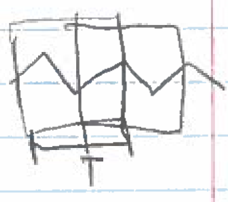
$$C_k = \frac{1}{-\pi k}\cdot\sin\left(\pi/2\; k\right)$$

$$\boxed{\tilde{x}(t) = \sum_{k=-K}^{K} \frac{1}{-\pi k}\sin\left(\pi/2\; k\right)\cdot e^{\frac{j2\pi kt}{T}}}$$

3) b. See attachment at end

3) c. Describe what you see at the discontinuous parts ie the corners. How does this reconsile w/ (10) in the notes?

The edges develop a sort of spike, this is b/c discontinuity is incredibly hard to model with continuous functions. As the number of terms increases, the spikes at the corners decrease in area, so although they never disapear, their effect eventually (at infinity) becomes so small it can be ignored.



4) a. Suppose $x(t)$ has a period T, and a Fourier series representation w/ coefficients C_k. Consider a new function $y(t) = x(t-T_1)$ where $|T_1| < T$; thus $y(t)$ is a delayed version of $x(t)$. Find the Fourier series coefficient for $y(t)$ in terms of C_k.

$$C_k = \frac{1}{T}\int_{-T/2}^{T/2} x(t)\, e^{-j\frac{2\pi}{T}kt}\, dt$$

$$= \frac{1}{T}\int_{-T/2-W}^{T/2-W} x(t)\, e^{-j\frac{2\pi}{T}kt}\, dt$$

$f(t) = t - T_1$

$$C_{new} = \frac{1}{T}\int_{-T/2}^{T/2} x(f(t))\cdot e^{-j\frac{2\pi}{T}kt}\, dt$$

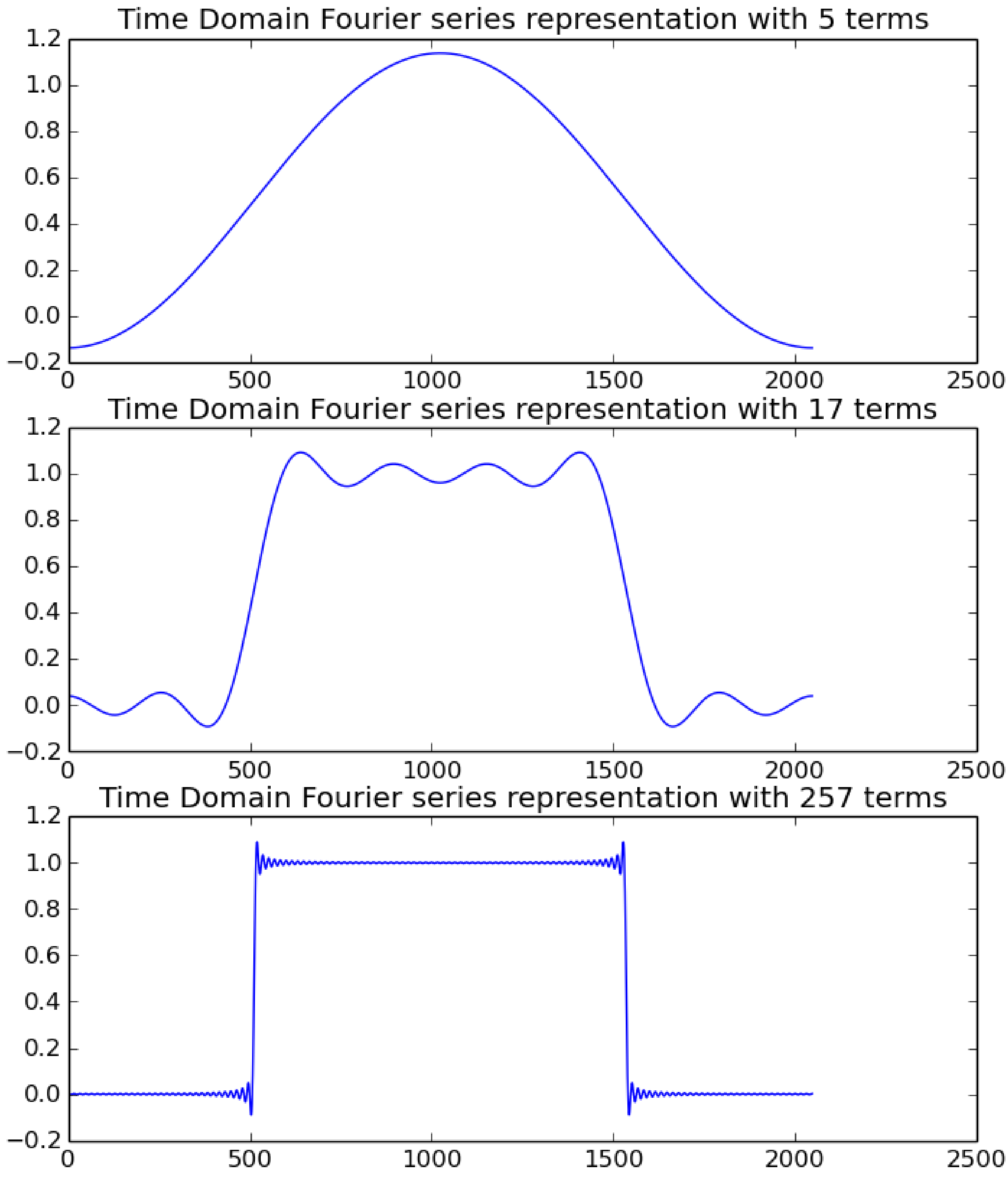
to add a phase shift, multiply C_k by $e^{-j\frac{2\pi}{T}k\phi}$

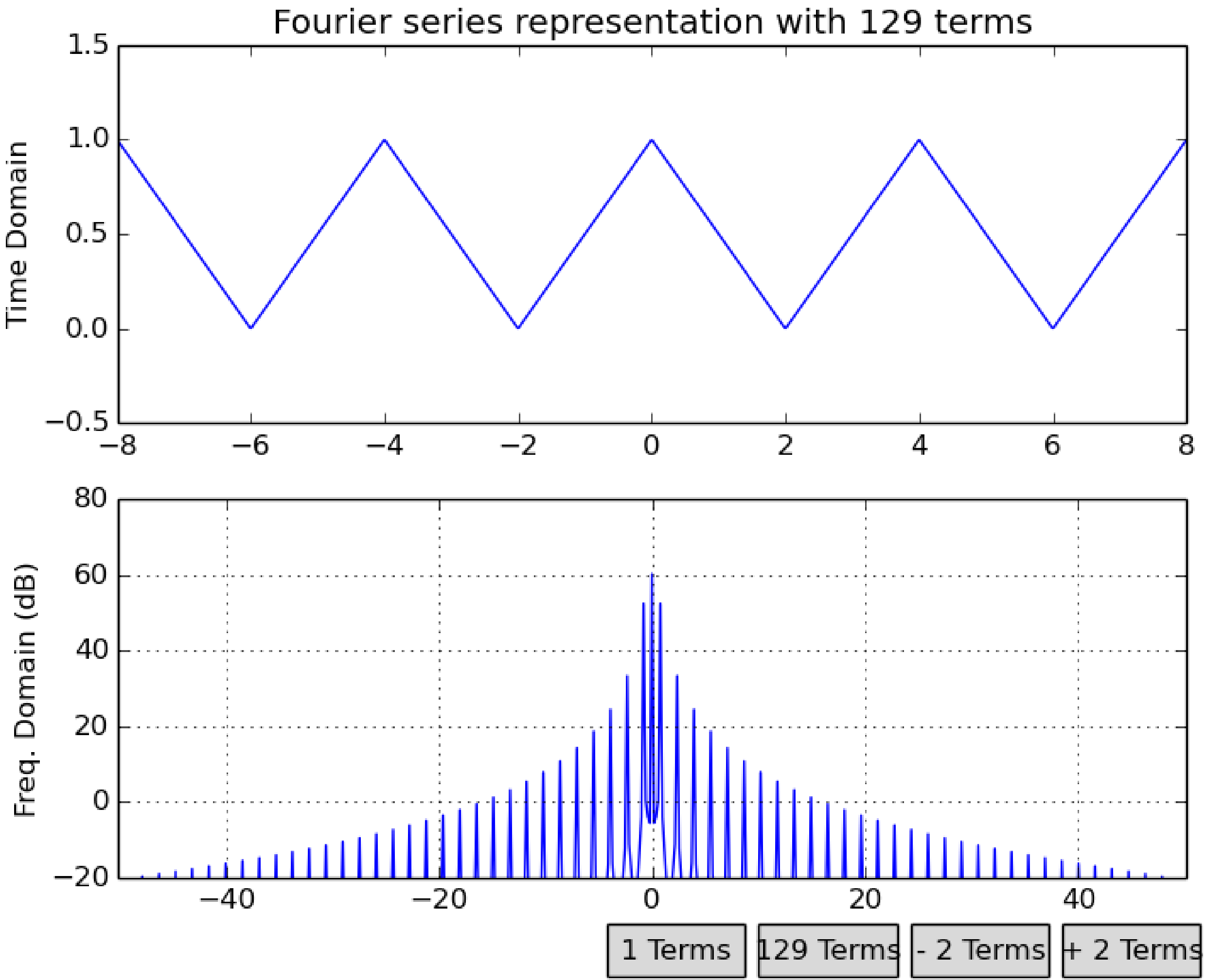
$\boxed{New = C_k e^{-j\frac{2\pi}{T}k\phi}}$ where ϕ is desired shift

this makes sense b/c we

represent the periodic signal w/ complex exponentials.

4) b. See Attachment at end


Time Domain Fourier series representation with 5 terms
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
500
1000
1500
2000
2500
Time Domain Fourier series representation with 17 terms
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
500
1000
1500
2000
2500
Time Domain Fourier series representation with 257 terms
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
500
1000
1500
2000
2500


Fourier series representation with 129 terms
Time Domain
1.5
1.0
0.5
0.0
−0.5
−8
−6
−4
−2
0
2
4
6
8
Freq. Domain (dB)
80
60
40
20
0
−20
−40
−20
0
20
40
1 Terms
129 Terms
- 2 Terms
+ 2 Terms

```
from __future__ import print_function
import numpy as np
import matplotlib.pyplot as mplib


def fs_triangle(ts, M=3, T=4, O=0):
    """
    computes a fourier series representation of a triangle wave
    with M terms in the Fourier series approximation
    T is the period and O is the phase offset
    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
            Coeff *= np.exp(-1j*2*np.pi*k*O/T)
            x = x + Coeff*np.exp(1j*2*np.pi/T*k*ts)

    # if M is odd
    elif 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
            Coeff *= np.exp(-1j*2*np.pi*k*O/T)
            x = x + Coeff*np.exp(1j*2*np.pi/T*k*ts)

    return x

    from matplotlib.widgets import Button

# create a plot to demonstrate the Fourier Series of a triangle wave
# we are going to need 2 axes, one for the figure
# and another for the slider
fig, ax = mplib.subplots()
T = 4;
```

```
# create an array of time samples from -6 to 6 and use 6000 points to
represent the wave
ts = np.linspace(-8,8,2048)

# compute the Fourier series representation with 1 term of a square wave
with period 4
x = fs_triangle(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)')


# this is the function that gets called whenever the slider is changed
# the value of the slider is passed as M
def on_change(M):
#    mplib.text(2, 6,"Fourier Series with %d terms" % (M) )
    mplib.subplot(211)
    mplib.title("Fourier series representation with %d terms" % (M) )

    # compute a FS representation for the number
ohttp://localhost:8888/notebooks/FourierSeriesTriangleWave.ipynb#f FS
terms from the slider
    x = fs_triangle(ts, M, T=4, O=2)
    # 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()
```

```
# this class handles the plotting operations in response to button
presses
class PlotFS:

    M = 1
    M_max = 129
    M_min = 1

    # this method increments the number of FS terms plotted by 2
    def add_two(self, event):
        self.M += 2
        # wrap around
        if self.M > self.M_max:
            self.M = self.M_max
        on_change(self.M)

    # this method decrements the number of FS terms plotted by 2
    def sub_two(self, event):
        self.M -= 2
        # wrap around
        if self.M < self.M_min:
            self.M = self.M_min
        on_change(self.M)


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

    # this method minimizes the number of FS terms
    def minimize(self, event):
        self.M = self.M_min
        on_change(self.M)

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

# make the buttons
axprev = mplib.axes([0.7, 0.005, 0.1, 0.05])
axnext = mplib.axes([0.81, 0.005, 0.1, 0.05])
axmax = 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_min)
bmax = Button(axmax, "%d Terms" % callback.M_max)
bnext = Button(axnext, '+ 2 Terms')
bnext.on_clicked(callback.add_two)
bprev = Button(axprev, '- 2 Terms')
bprev.on_clicked(callback.sub_two)
bmax.on_clicked(callback.maximize)
bmin.on_clicked(callback.minimize)
```

```
mplib.show()

def fs_square(ts, M=3, T=4):
    # computes a fourier series representation of a squre 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)):
###########################################################################
#############################
            ##  change the following line to provide the Fourier series
coefficients for the square wave
            ##  Coeff = ??

            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):
###########################################################################
#############################
            ##  change the following line to provide the Fourier series
coefficients for the square wave
            if k != 0:
                Coeff = 1/(np.pi*k)*np.sin(np.pi/2*k)
            else:
                Coeff = 1

            x = x + Coeff*np.exp(1j*2*np.pi/T*k*ts)

    return x-0.5

    from matplotlib.widgets import Button

# create a plot to demonstrate the Fourier Series of a triangle wave
# we are going to need 2 axes, one for the figure
# and another for the slider
fig, ax = mplib.subplots()
T = 1;

# create an array of time samples from -6 to 6 and use 6000 points to
represent the wave
ts = np.linspace(-2,2,2048)

# compute the Fourier series representation with 1 term of a square wave
with period 4
```

```
x = fs_square(ts, 1, T)


mplib.subplot(2,1,1)
# set the axis range for the main plot
mplib.axis([-2, 2,-0.5, 1.5])

# plot the FS representation
line1, = mplib.plot(ts, x, lw=1)
mplib.title("Time Domain Fourier series representation with 1 term" )

mplib.ylabel('Time Domain')
mplib.grid()
mplib.subplot(2,1,2)
mplib.title("Freq. Domain Fourier series representation with 1 term" )

# 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)/2,np.max(fs)/2, -20, 80])
mplib.grid()
mplib.ylabel('Freq. Domain (dB)')


# this is the function that gets called whenever the slider is changed
# the value of the slider is passed as M
def on_change(M):
#    mplib.text(2, 6,"Fourier Series with %d terms" % (M) )
    mplib.subplot(211)
    mplib.title("Time Domain 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
ohttp://localhost:8888/notebooks/FourierSeriesTriangleWave.ipynb#f FS
terms from the slider
    x = fs_square(ts, M, T)
    # 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()
```

```
# this class handles the plotting operations in response to button
presses
class PlotFS:

    M = 1
    M_max = 257
    M_min = 1

    # this method increments the number of FS terms plotted by 2
    def add_two(self, event):
        self.M += 2
        # wrap around
        if self.M > self.M_max:
            self.M = self.M_max
        on_change(self.M)

    # this method decrements the number of FS terms plotted by 2
    def sub_two(self, event):
        self.M -= 2
        # wrap around
        if self.M < self.M_min:
            self.M = self.M_min
        on_change(self.M)


   # this method maximizes the number of FS terms
    def maximize(self, event):

        self.M = self.M_max
        on_change(self.M)

    # this method minimizes the number of FS terms
    def minimize(self, event):
        self.M = self.M_min
        on_change(self.M)

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

# make the buttons
axprev = mplib.axes([0.7, 0.005, 0.1, 0.05])
axnext = mplib.axes([0.81, 0.005, 0.1, 0.05])
axmax = 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_min)
bmax = Button(axmax, "%d Terms" % callback.M_max)
bnext = Button(axnext, '+ 2 Terms')
bprev = Button(axprev, '- 2 Terms')

bnext.on_clicked(callback.add_two)
bprev.on_clicked(callback.sub_two)
bmax.on_clicked(callback.maximize)
bmin.on_clicked(callback.minimize)
```

```
mplib.show()
five = fs_square(ts, M=5, T=4)
seventeen = fs_square(ts, M=17, T=4)
twofiftyseven = fs_square(ts, M=257, T=4)

fig, ax = mplib.subplots()

mplib.subplot(3,1,1)
# plot the FS representation
mplib.subplot(311)
mplib.plot(five)
mplib.title("Time Domain Fourier series representation with 5 terms" )
mplib.subplot(312)
mplib.plot(seventeen)
mplib.title("Time Domain Fourier series representation with 17 terms" )
mplib.subplot(313)
mplib.plot(twofiftyseven)
mplib.title("Time Domain Fourier series representation with 257 terms" )
mplib.show()

tri = fs_triangle(ts, M=200, T=4,O=2)
mplib.plot(tri)
mplib.show()
```