SIGSYS PS06 - Jennifer Wei ~~02~~ 03 · 05 · 2015

1. An audio signal of a gun being fired in a shooting range can be convolved with a ~~vide~~ violin recording to approximate how the violin would sound in the shooting range.

This is because we're working with an LTI system.

$\delta(t) \longrightarrow \boxed{} \longrightarrow h(t)$

impulse — impulse response

$x(t) \longrightarrow \boxed{} \longrightarrow y(t)$

Input signal — Output signal

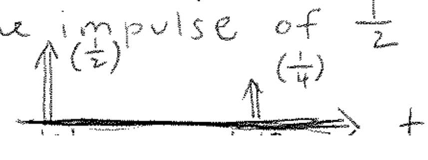
From the reading (and also from class), we know that $y(t) = x * h(t) = h * x(t)$.

The gunshot is essentially an impulse, and the audio from the shooting range is the impulse response. The system is the shooting range.

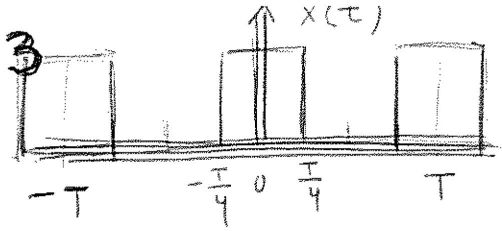
Thus, when the violin is inputted, when convolved with $h(t)$, the violin is affected in a similar manner as the gunshot in the shooting range, and the outputted sound has echoing, like that of the gunshot in the shooting range.

2. $y(t) = \frac{1}{2}x(t-1) + \frac{1}{4}x(t-10)$

$y(t)$ is an echo since it is a delayed, scaled down version of the input, $x(t)$. The impulse response is $h(t) = \frac{1}{2}\delta(t-1) + \frac{1}{4}\delta(t-10)$, with one impulse of $\frac{1}{2}$ at $t=1$ and another at ~~wa~~ $t=10$ of $\frac{1}{4}$



3



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

since $\int_{-T/2}^{-T/4}$ & $\int_{T/4}^{T/2}$ are 0

FOURIER SERIES:

$$x(t) = \sum_{K=0}^{\infty} C_K e^{j\frac{2\pi}{T}kt}$$

Thus, $C_K = \frac{1}{T}\int_{-T/4}^{T/4} e^{-j\frac{2\pi}{T}kt}\,dt$

$C_K = \frac{1}{T}\cdot\frac{-T}{2j\pi k}\, e^{-\frac{j2\pi k}{T}t}\Big|_{-T/4}^{T/4}$

$C_K = \frac{-1}{j2\pi k}\left(e^{-j\frac{2\pi}{T}k\frac{T}{4}} - e^{j\frac{2\pi}{T}k\frac{T}{4}}\right)$

$C_K = \frac{1}{2j\pi k}\left(e^{\frac{j\pi k}{2}} - e^{-\frac{j\pi k}{2}}\right)$

$\sin(\theta) = \frac{1}{2j}\left(e^{j\theta} - e^{-j\theta}\right)$

$\therefore C_K = \left(\frac{1}{\pi k}\right)\sin\left(\frac{\pi k}{2}\right) = \frac{1}{2}\operatorname{sinc}\left(\frac{K}{2}\right)$

b) see graphs below + code

c) At the discontinuous points of the square wave, the representations have visible sinusoidal oscillations since the sine waves that make up the representations are approximations that oscillate around the square wave and closes in on the square wave as more terms are added. However, the deviation (error) will not ever reach zero and 100% match the square wave since the Fourier series approaches but cannot reach ∞.

4a) $y(t) = x(t - T_1)$, where $|T_1| < T$

* worked with others on this question

$C_K = \frac{1}{T}\int_{-T/2}^{T/2} x(t)\, e^{-j\frac{2\pi}{T}kt}\,dt = \frac{1}{T}\int_{-\frac{T}{2}-w}^{\frac{T}{2}-w} x(t)\, e^{-j\frac{2\pi}{T}kt}\,dt$, where w = any #

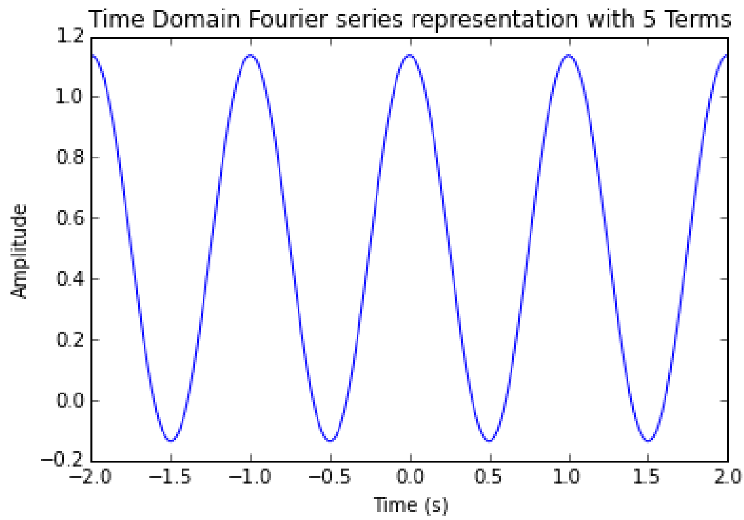
Let $u = t - T_1$ & $du = dt$

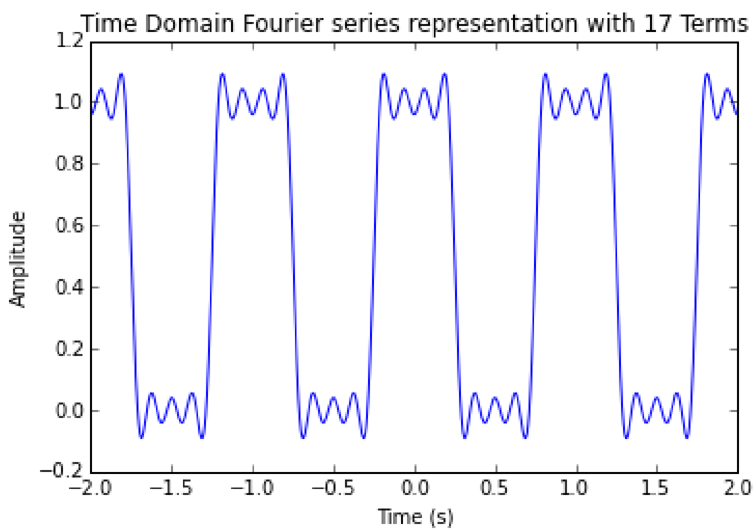
$C_{K_2} = \frac{1}{T}\int_{-T/2-T_1}^{T/2-T_1} x(u)\, e^{-j\frac{2\pi}{T}k(u+T_1)}\,du = \frac{1}{T}\int_{-v}^{v} x(u)\, e^{-j\frac{2\pi}{T}k(u+T_1)}\,du$

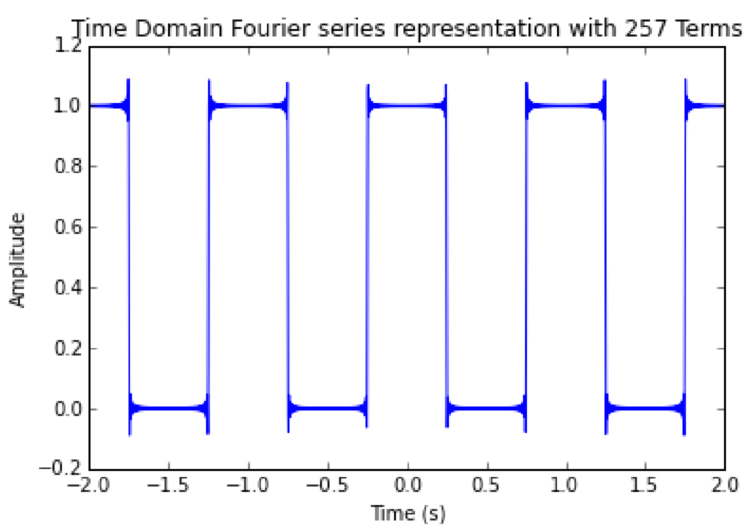
$C_{K_2} = \frac{1}{T}\int_{-v}^{v} x(u)\, e^{-j\frac{2\pi}{T}ku}\, e^{-j\frac{2\pi}{T}kT_1}\,du = e^{-j\frac{2\pi}{T}kT_1}\cdot\frac{1}{T}\int_{-v}^{v} x(u)\, e^{-j\frac{2\pi}{T}ku}\,du$

$\therefore C_{K_2} = e^{-j\frac{2\pi}{T}kT_1}\cdot C_K$

3B


Time Domain Fourier series representation with 5 Terms
Amplitude
Time (s)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Time Domain Fourier series representation with 17 Terms
Amplitude
Time (s)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Time Domain Fourier series representation with 257 Terms
Amplitude
Time (s)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

3B

```
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)):

##########################################################################################
         ##  change the following line to provide the Fourier series coefficients for the square wave
         ##  Coeff = ??
         Coeff = 0.5*np.sinc(k/2.) #Calculated Coefficient from 3a
         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
         ##  Coeff = ??
         Coeff = 0.5*np.sinc(k/2.) #Calculated Coefficient from 3a
         x = x + Coeff*np.exp(1j*2*np.pi/T*k*ts)

   return x

# compute the Fourier series representation with 1 term of a square wave with period 4
T0 = 4
ts = np.linspace(-2,2,2048)
mplib.axis([-8, 8,-1.5, 1.5])

x0 = fs_square(ts, 1, T)
x1 = fs_square(ts, 5, T)
x2 = fs_square(ts, 17, T)
x3 = fs_square(ts, 257, T)

mplib.plot(ts, x) #replace x with x0, x1, x2, x3 to plot Time Domain Fourier series representation
```

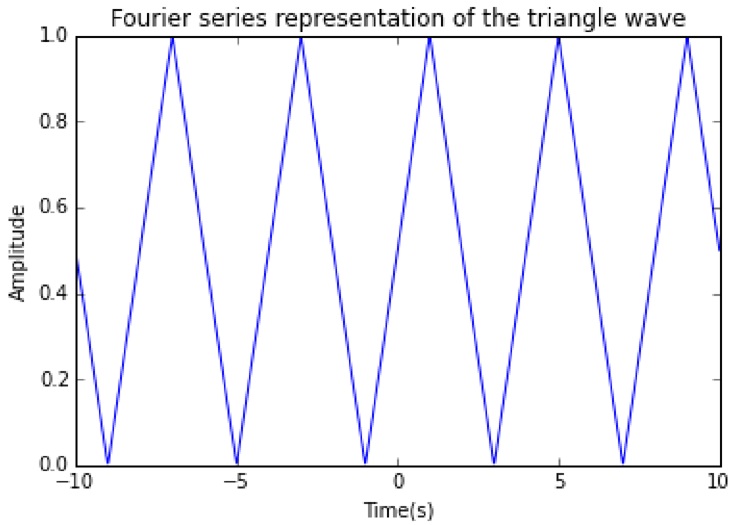
4B

```
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)
            x = x + np.exp(-1j*2*np.pi/T*k)*Coeff*np.exp(-1j*2*np.pi/T*k*ts)#modified fs_triangle
    # 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)
            x = x + np.exp(-1j*2*np.pi/T*k)*Coeff*np.exp(-1j*2*np.pi/T*k*ts)#modified fs_triangle
    return x
```

4B

```
ts = np.linspace(-10,10,2048)
x = fs_triangle(ts, M=100)
mplib.plot(ts,x)
mplib.xlabel("Time(s)")
mplib.ylabel("Amplitude")
mplib.title("Fourier series representation of the triangle wave")
```

```
<matplotlib.text.Text at 0xbc393c8>
```



Above is the generated triangle waves which look quite similar, though it is a bit offcenter in comparison to Figure 2, and I'm not quite sure why that is.