Michael Bozamazo
SS Sys PS06

1. A gun approximates an impulse because it is a short "kick" or burst of energy that has roughly similar amplitudes at all frequencies. An ideal unit impulse would have identical amplitudes at all frequencies.
The sound produced by the gunshot is then roughly an impulse response, or the ~~product of~~ output of the system ~~is~~ from the input of the impulse.
The signal should be renormalized to have ~~amplitude~~ area 1 under the curve.
As we saw in lecture, the picking property of LTI systems means that convolution is also valid in continuous time.
Convolution ~~the~~ of the input signal and the impulse response then produces the output signal after the transfer function by linearity, which is applicable for LTI systems

PS06 #2

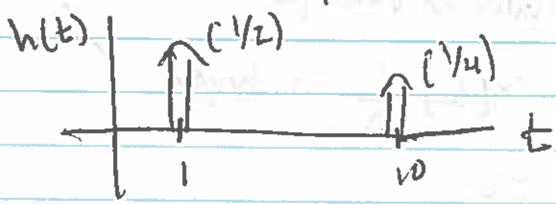
$y(t) = \frac{1}{2}x(t-1) + \frac{1}{4}x(t-10)$

An echo channel is a reasonable name because the signal is repeated back after 10 seconds with lower amplitude.

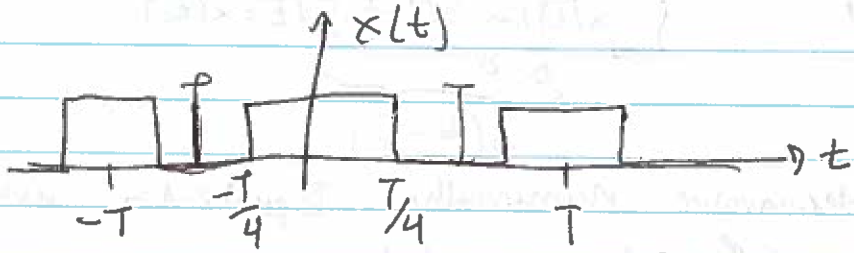
What is the impulse response of the system?

$f(\delta(t)) = h(t)$

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



3.



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

Triangle Wave:

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

$C_k = \begin{cases} -\frac{2}{\pi^2 k^2} & \text{if } k \text{ is odd} \\ 1/2 & \text{if } k=0 \\ 0 & \text{otherwise} \end{cases}$

note $x(t) = \frac{2}{T}|t|$ w/ $-T/2 < t \leq T/2$

Square wave w/ $-T/2 < t \leq T/2$:

$x(t) = \begin{cases} 1 & \text{if } |t| < T/4 \\ 0 & \text{otherwise} \end{cases}$

plug in to integration?

$$x(t) \approx \tilde{x}_K(t) = \sum_{k=-K}^{K} c_k e^{j\frac{2\pi}{T}kt}$$

$$K \to \infty \int_{-T/2}^{T/2} \left| \tilde{x}_K(t) - x(t) \right|^2 dt = 0$$

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

$T/4$

$-T/4$ sum of 2 exponentials → trig identity for sin

→ should be in the notebook

or

$$c_k = \frac{1}{T}\left(\int_{-T/2}^{-T/4} [0] e \cdots + \int_{-T/4}^{T/4} e^{-j\frac{2\pi}{T}kt} dt + \int_{T/4}^{T/2} [0] \cdots \right.$$

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

$$\frac{1}{T}\left[\frac{1}{-j\frac{2\pi}{T}k} e^{-j\frac{2\pi}{T}kt} \right]_{-T/4}^{T/4}$$

$$c_k = \frac{1}{-j\frac{2\pi}{T}k}\left[e^{-j\frac{1}{2}\pi k} - e^{+j\frac{1}{2}\pi k} \right] \qquad \sin(\theta) = \frac{1}{2j}\left(e^{j\theta} - e^{-j\theta}\right)$$

$\frac{1}{T}$ cancels

$$\left(\frac{-1}{\pi k}\right)\left(\frac{1}{2j}\right)\left[e^{j\theta} - e^{-j\theta}\right] \to \theta = -\frac{1}{2}\pi k$$

$$c_k = \left(\frac{-1}{\pi k}\right) \sin\left(-\frac{1}{2}\pi k\right)$$

only non-zero for odd terms

3b. $c_k = \left(\frac{-1}{\pi k}\right) \sin\left(\frac{-1}{2}\pi k\right) = \frac{i}{\pi k} \sin\left(\frac{-1}{2}\pi k\right)$

~~symmetrical~~ wrt origin

& Square wave is symmetric wrt y

$\frac{1}{2}\left(\frac{1}{\pi \frac{k}{2}}\right) \sin\left(\pi \frac{k}{2}\right)$

$\rightarrow \frac{1}{2} \operatorname{sinc}\left(\frac{k}{2}\right)$

~~3c~~. See figs

3c. There is a discontinuity of the derivative at the points where the value immediately shifts from 1 ~~to~~ to 0 or 0 to 1. A large change happens infinitely fast — bc it is discontinuous.

Observed at such points is a poor approximation of the ideal square wave, but a valid approximation of a discretely sampled square wave.

~~Eventually~~ As k goes to infinity, as in (10), the energy ~~stu~~ of both signals should converge, but any discrete approx will appear to fail to approximate the curve, even though it fits the sampled points perfectly. We are working in the discrete domain with computation, so there might be some mismatch between the continuous math and discrete representations.

4a. $y(t) = x(t - T_1)$ where $T_1 < T$ w/ c_k

$y(t)$ is a delayed $x(t)$

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

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

Find $y(t)$ in terms of c_k

$$c_{k(y)} = \frac{1}{T}\int_{-T/2}^{T/2} x(\underbrace{t-T_1}_{s}) e^{-j\frac{2\pi}{T}kt}dt$$

$s = t - T_1$

$t = s + T_1$ $\quad \frac{ds}{dt} = 1$

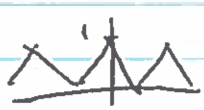
$dt = ds$

substitution of variables

$$c_{k(y)} = \frac{1}{T}\underbrace{\int_{s=-T/2-T_1}^{s=T/2-T_1}}_{\text{shift by } T_1 \text{ by given}} x(s) e^{-j\frac{2\pi}{T}k(s+T_1)} ds$$

$$c_{k(y)} = e^{-j\frac{2\pi}{T}T_1} \cdot \underbrace{\int_{s=-T/2}^{s=T/2} x(s) e^{-j\frac{2\pi}{T}ks}ds}_{c_k}$$

$$c_{k(y)} = e^{-j\frac{2\pi}{T}kT_1}\cdot c_k$$

b. Figure 2:  See final figure for new graph

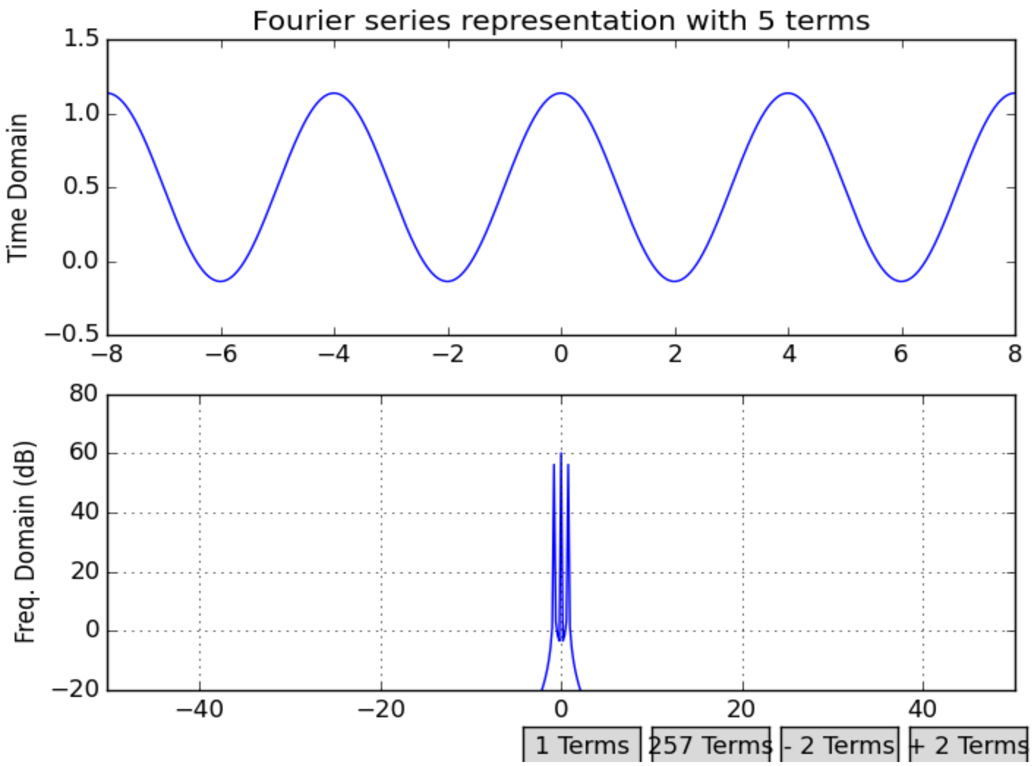
IPWB:  $T = 4$

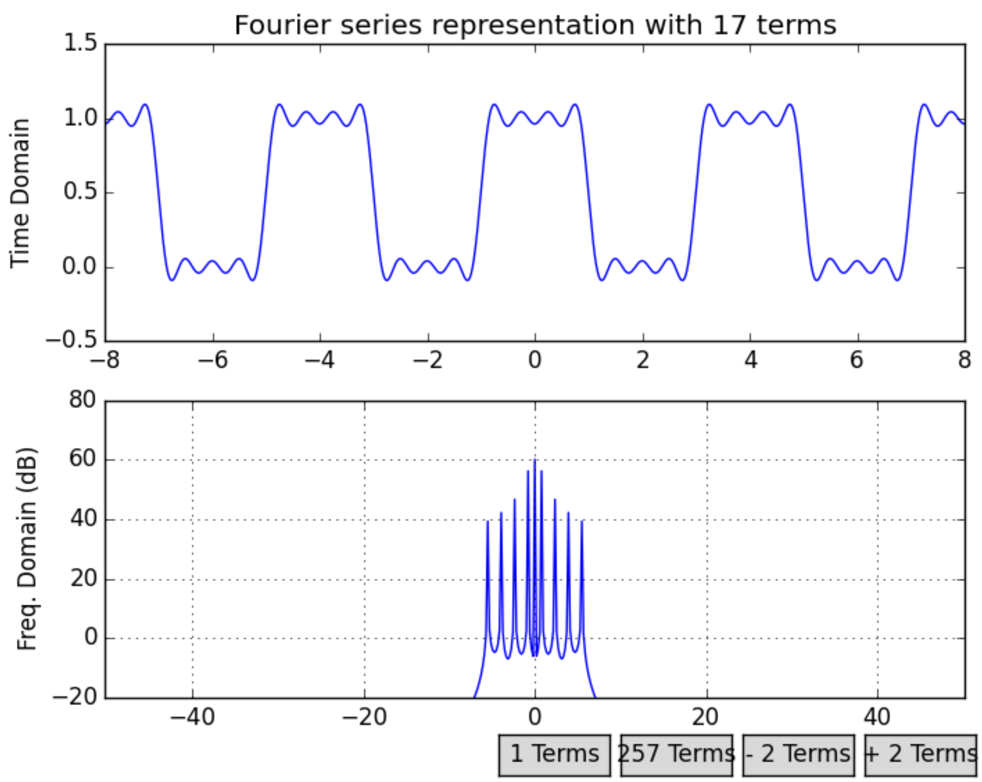
so shift is by $\frac{T_1}{T} = \frac{1}{2}$ or

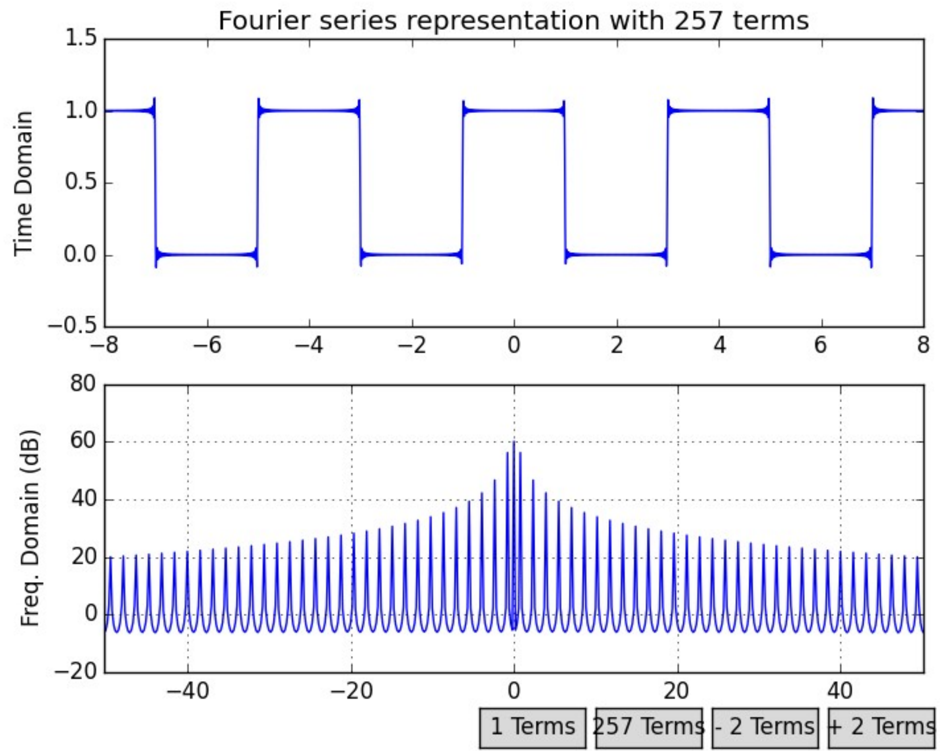
$e^{-j\frac{2\pi}{T}(\frac{1}{2})}$ $\quad T_1 = 2$

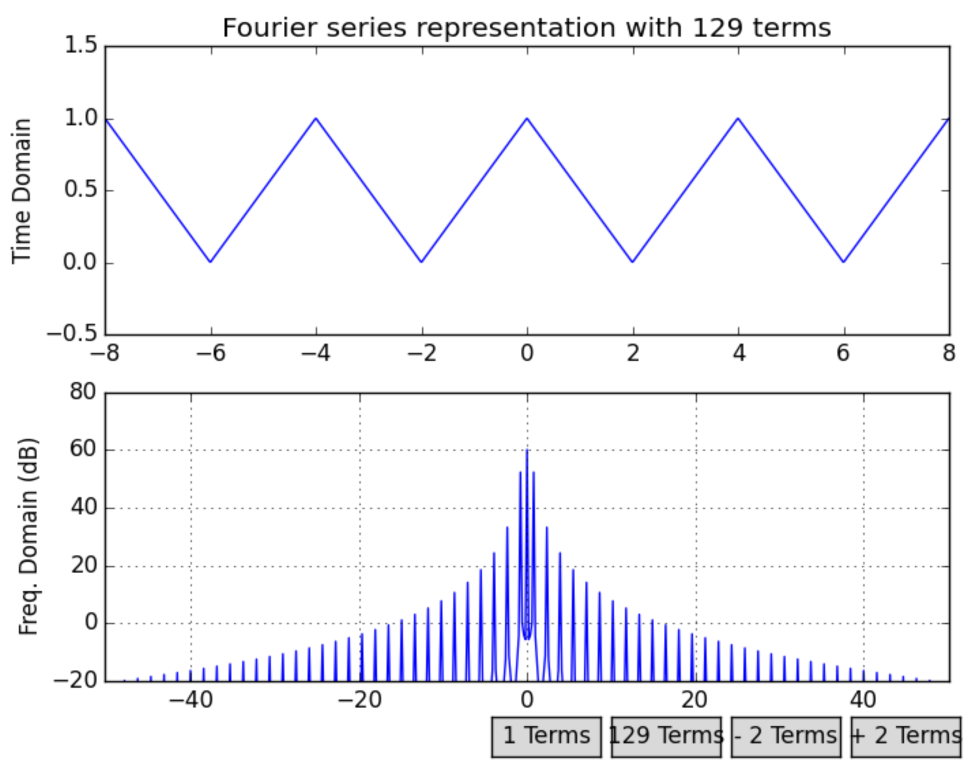
$$c_{k(y)} = e^{-j\pi k} c_k$$

~~this would make sense~~ if


Fourier series representation with 5 terms
Time Domain
Freq. Domain (dB)
1 Terms
257 Terms
- 2 Terms
+ 2 Terms


Fourier series representation with 17 terms
Time Domain
Freq. Domain (dB)
1 Terms
257 Terms
- 2 Terms
+ 2 Terms


Fourier series representation with 257 terms
Time Domain
Freq. Domain (dB)
1 Terms
257 Terms
- 2 Terms
+ 2 Terms


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

Slightly modified fs_triangle: accepts phi - phase offset as input, which modifies the coefficients of the complex exponentials
New line:

```
Coeff *= np.exp(-1j*2*math.pi*float(phi)/T*k)
```

The plotting function stays the same except for the function call, which has phi = 2.

```
def fs_triangle(ts, M=3, T=4, phi = 0):
    # 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

    # phi is the phase offset

    # 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 n == 0:
                Coeff = 0.5
            Coeff *= np.exp(-1j*2*math.pi*float(phi)/T*k)
            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 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*math.pi*float(phi)/T*k)
            x = x + Coeff*np.exp(1j*2*np.pi/T*k*ts)

    return x
```