ESE 531: Homework 4

Noah Schwab

February 20, 2022

Problem solutions with figures are shown below. Work and code is shown in attachments at end of document.

4.52

$y[n] = X_c(nT - \frac{T}{L})$

4.58

a) $h_1[0] = a, h_1[1] = c, h_1[2] = e$

$h_2[0] = b, h_2[1] = d, h_2[2] = 0$

$h_3[0] = 0, h_3[1] = 1, h_3[2] = 0$

b) The first system requires 10 multiplications per output point since it is upsampled at a rate 2 before being convolved with a 5-point impulse response.

The second system requires 8 multiplications per output since the upsampling takes places after $h_1[n]$ and $h_2[n]$ are passed, and $h_3[n]$ is simply a shift filter.

4.65

M = 441

L = 80

$\omega_c = \pi/441$

4.66

$\Omega_p = 11\pi \times 10^3$ rad/sec

$\Omega_s = 77\pi \times 10^3$ rad/sec

$$H_{a1}(j\Omega) = \begin{cases} 1 & : |\Omega| \leq 11\pi \times 10^3 \\ 0 & : |\Omega| > 77\pi \times 10^3 \end{cases}$$

Matlab Problem

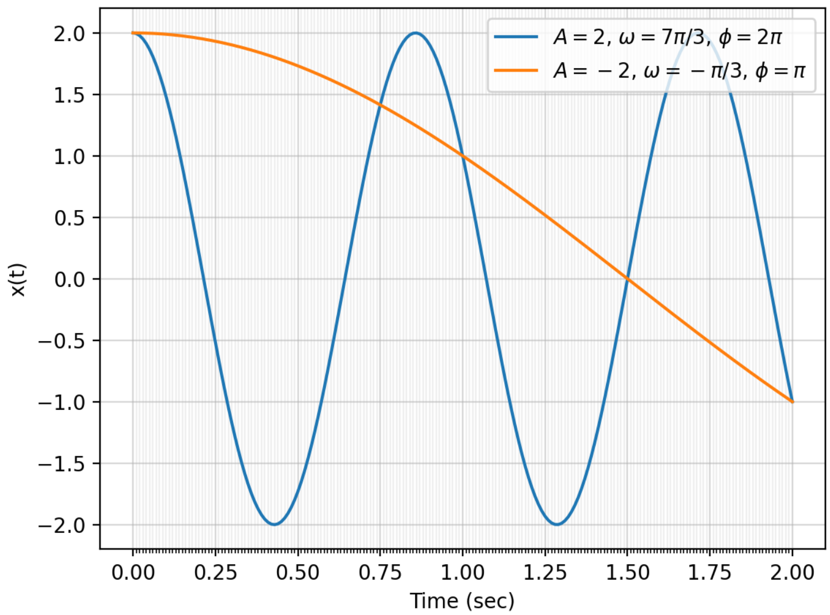
a) Fitting a Sine Wave

(i) $A = 2$

$\phi = 2\pi$ $\omega = \pi/3$

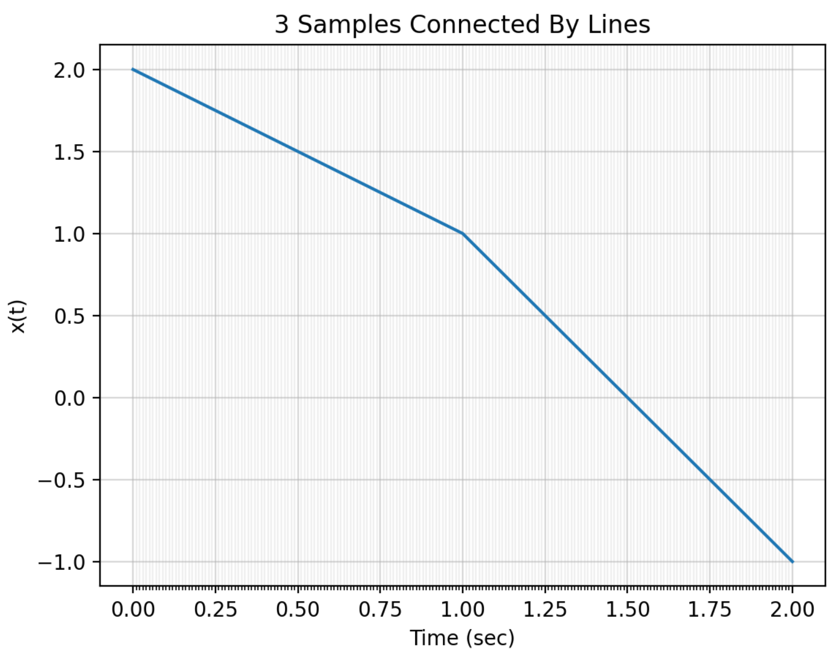
(ii) Yes, theoretically we have 3 unknowns, and 3 unique input-output pairs. As long as the variables are well-defined, we should always be able to solve the system.

(iii) Fitted Sinusoids

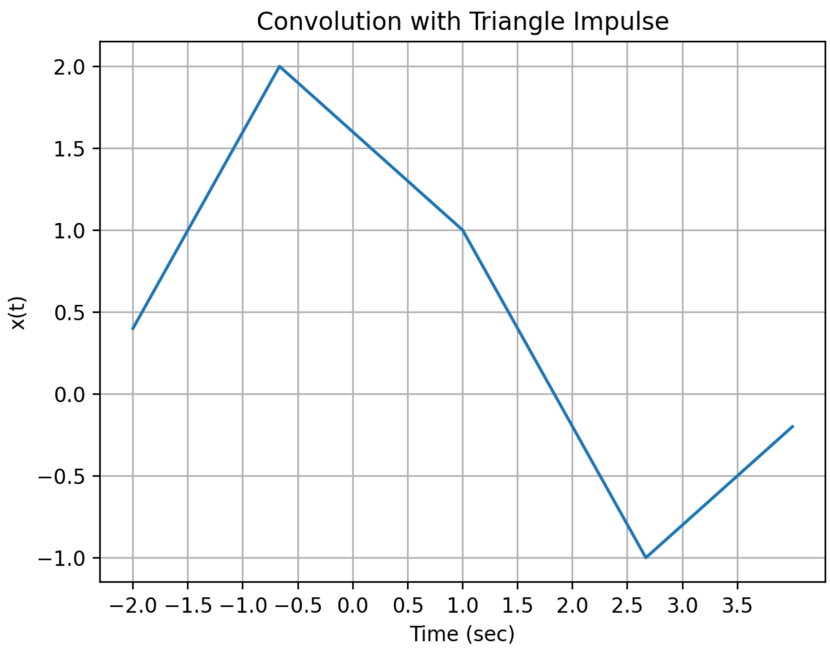


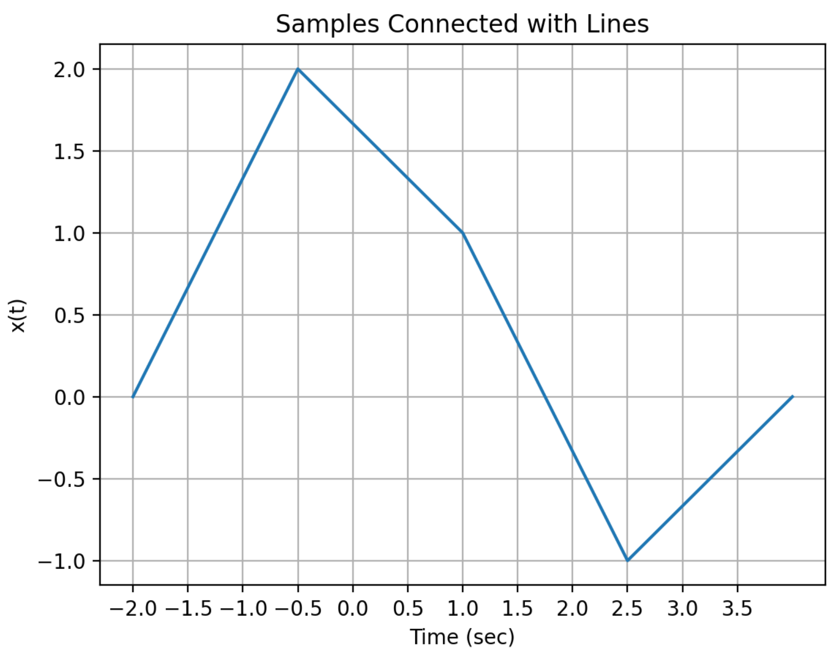
b) Linear and Polynomial Interpolation

(i) Samples Connected With a Line



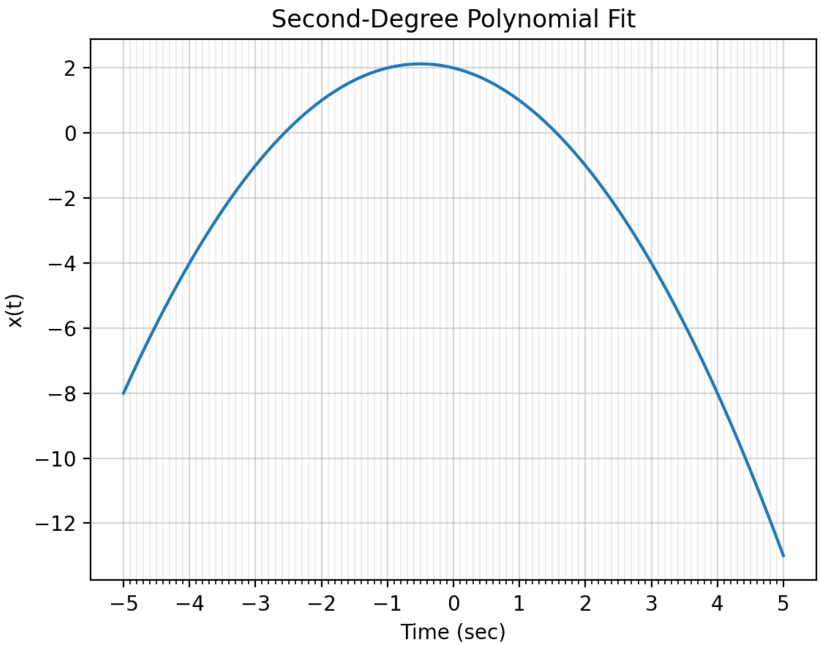
(ii) Linear Interpolation





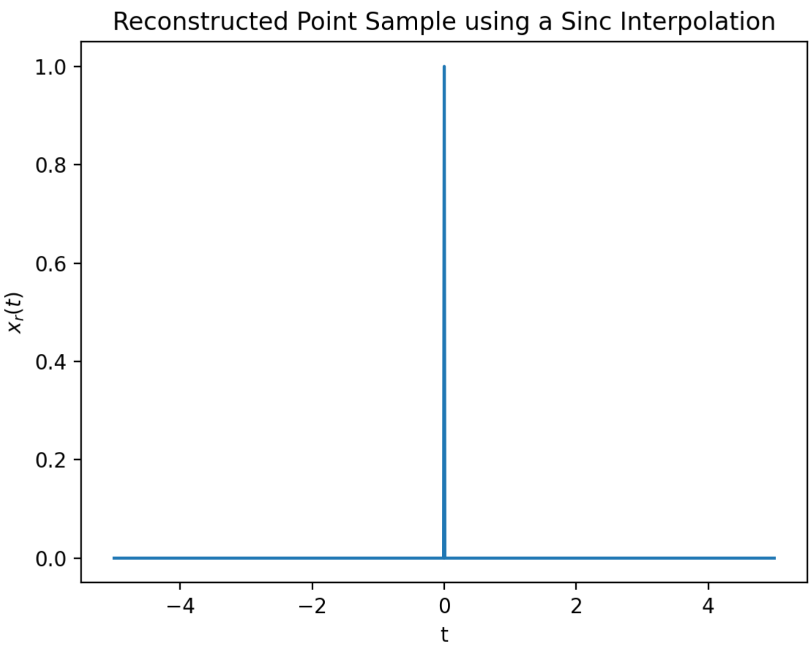
(iii) Polynomial Fit

As we can see in the figure below, the second-degree polynomial fit does a poor job extending to values beyond $0 \leq t \leq 2$. It does not resemble the sinusoid since it must extend to $\pm\infty$ as the domain extends to $\pm\infty$. Therefore, it is not practical for fitting to sinusoidal values.



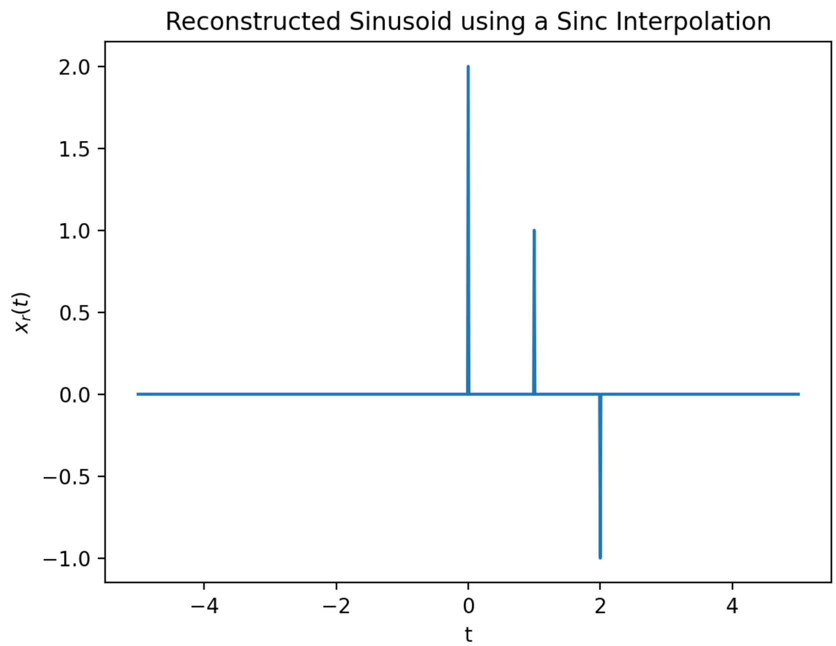
c) Ideal Low-Pass Filtering

(i) Single-Point Reconstruction



(ii) Three-Point Reconstruction

Using Sinc Interpolation, we obtain a signal that is a series of delta spikes at the corresponding points in time. It resembles a sampled sinusoid, such as the one part from (a).


Reconstructed Sinusoid using a Sinc Interpolation
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
$x_r(t)$
−4
−2
0
2
4
t

Noah Schwab
ESE 531

02/20/22

HW 4: Filter Banks and Reconstruction

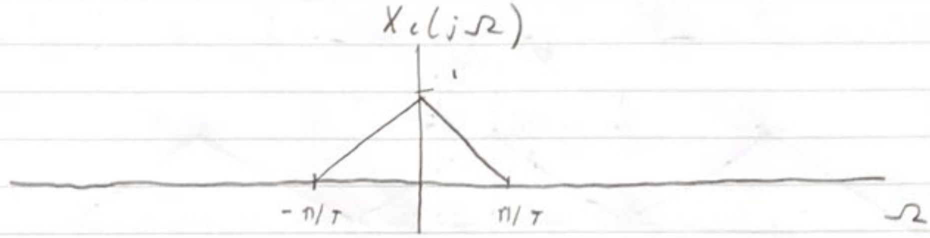
4. 52

$$X_c(j\Omega) = 0, \quad |\Omega| \geq \pi/T$$

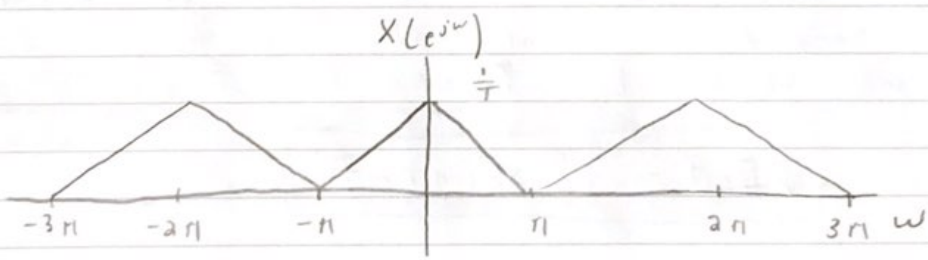
$$H(e^{j\omega}) = \begin{cases} e^{-j\omega}, & |\omega| < \pi/L \\ 0, & \pi/L < |\omega| \leq \pi \end{cases}$$

$x_c(t)$ is bandlimited at $\Omega_N = \pi/T$, which means it satisfies Nyquist

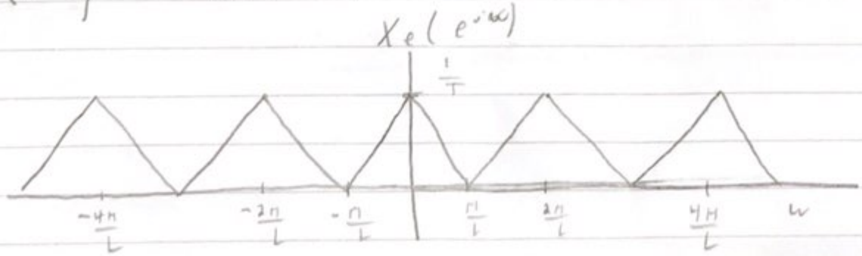
Consider the following $X_c(j\Omega)$



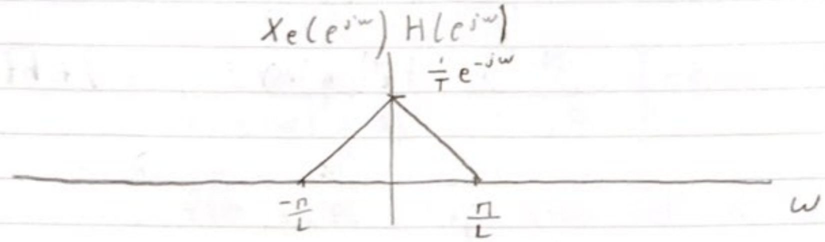
$$X(e^{j\omega}) = \frac{1}{T} \sum_{n=-\infty}^{\infty} X_c\left(j\left(\frac{\omega}{T} - \frac{2\pi n}{T}\right)\right)$$



$$X_e(e^{j\omega}) = X(e^{j\omega L})$$

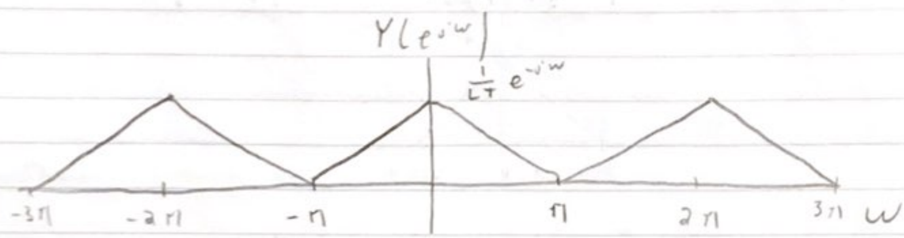


$$X_e(e^{j\omega})H(e^{j\omega}) = \begin{cases} X_e(e^{j\omega})e^{-j\omega}, & |\omega| < \pi/L \\ 0, & \pi/L < |\omega| \le \pi \end{cases}$$



$$Y(e^{j\omega}) = \frac{1}{L}\sum_{i=0}^{L-1} G\left(e^{j(\omega/L - 2\pi i/L)}\right),$$

where $G(e^{j\omega}) = X_e(e^{j\omega})H(e^{j\omega})$



We see that $y[n]$ is a version of $x_c(t)$ that is sampled at a rate of $f_s = \frac{1}{T}$ and shifted by $\frac{T}{L}$.

$$\boxed{y[n] = x_c\left(nT - \frac{T}{L}\right)}$$

4.58

a) we want to determine $h_1[n]$, $h_2[n]$, $h_3[n]$ s.t

$$\left(x[n] * h_1[n] \rightarrow \boxed{\uparrow 2}\right) + \left(x[n] * h_2[n] \rightarrow \boxed{\uparrow 2}\right) * h_3[n]$$

$$= \left(x[n] \rightarrow \boxed{\uparrow 2}\right) * h[n]$$

where $h[n]$ is shown in Figure P4.58-1

$h[n]$ can be expressed in the following form:

$$h[n] = a\delta[n] + b\delta[n-1] + c\delta[n-2] + d\delta[n-3] + e\delta[n-4]$$

we can solve for $y_1[n]$:

$$\begin{aligned} y_1[n] &= w[n] * h[n] \\ &= x\left[\tfrac{n}{2}\right] * h[n] \\ &= \sum_{k=-\infty}^{\infty} x\left[\tfrac{n}{2} - \tfrac{k}{2}\right] h[n] = \sum_{k=0}^{4} x\left[\tfrac{n}{2} - \tfrac{k}{2}\right] h[k] \end{aligned}$$

$h[n] = 0$ for $n < 0$ and $n > 4$

$$\begin{aligned} y_1[n] = {} & x\left[\tfrac{n}{2}\right] h[0] + x\left[\tfrac{n}{2} - \tfrac{1}{2}\right] h[1] + x\left[\tfrac{n}{2} - 1\right] h[2] \\ & + x\left[\tfrac{n}{2} - \tfrac{3}{2}\right] h[3] + x\left[\tfrac{n}{2} - 2\right] h[4] \end{aligned}$$

From the definition of $h[n]$:

$$\boxed{y_1[n] = a\,x\left[\tfrac{n}{2}\right] + b\,x\left[\tfrac{n}{2} - \tfrac{1}{2}\right] + c\,x\left[\tfrac{n}{2} - 1\right] + d\,x\left[\tfrac{n}{2} - \tfrac{3}{2}\right] + e\,x\left[\tfrac{n}{2} - 2\right]}$$

Our goal is to design the system s.t. $y_2[n] = y_1[n]$

Start w/ the top system:

$$X[n] * h_1[n] = \sum_{k=-\infty}^{\infty} X[n-k]\, h_1[k]$$

$h_1[k]$ is limited to $0 \leq n \leq 2$:

$$= \sum_{k=0}^{2} X[n-k]\, h_1[k]$$

$$= x[n]\, h_1[0] + X[n-1]\, h_1[1] + X[n-2]\, h_1[2]$$

$$w_1[n] = \underline{h_1[0]\, X[\tfrac{n}{2}] + h_1[1]\, X[\tfrac{n}{2}-\tfrac{1}{2}] + h_1[2]\, X[\tfrac{n}{2}-1]}$$

Now the bottom system:

$$x[n] * h_2[n] = \sum_{k=0}^{2} X[n-k]\, h_2[k]$$

$$= h_2[0]\, x[n] + h_2[1]\, x[n-1] + h_2[2]\, x[n-2]$$

$$w_2[n] = h_2[0]\, X[\tfrac{n}{2}] + h_2[1]\, X[\tfrac{n}{2}-\tfrac{1}{2}] + h_2[2]\, X[\tfrac{n}{2}-1]$$

$$w_3[n] = w_2[n] * h_3[n] = \sum_{k=0}^{2} w_2[n-k]\, h_3[k]$$

$$= h_3[0]\, w_2[n] + h_3[1]\, w_2[n-1] + h_3[2]\, w_2[n-2]$$

$$\begin{aligned} w_3[n] = {} & h_3[0]\left(h_2[0]\, x[\tfrac{n}{2}] + h_2[1]\, x[\tfrac{n}{2}-\tfrac{1}{2}] + h_2[2]\, x[\tfrac{n}{2}-1]\right) \\ & + h_3[1]\left(h_2[0]\, x[\tfrac{n}{2}-\tfrac{1}{2}] + h_2[1]\, x[\tfrac{n}{2}-1] + h_2[2]\, x[\tfrac{n}{2}-\tfrac{3}{2}]\right) \\ & + h_3[2]\left(h_1[0]\, x[\tfrac{n}{2}-1] + h_2[1]\, x[\tfrac{n}{2}-\tfrac{3}{2}] + h_2[2]\, x[\tfrac{n}{2}-2]\right) \end{aligned}$$

Grouping like terms of $x[n]$:

$$\begin{aligned} w_3[n] = \; & h_3[0]h_2[0]\,x[\tfrac{n}{2}] \\ & + \left(h_3[0]h_2[1] + h_3[1]h_2[0]\right) x[\tfrac{n}{2}-\tfrac{1}{2}] \\ & + \left(h_3[0]h_2[2] + h_3[1]h_2[1] + h_3[2]h_2[0]\right) x[\tfrac{n}{2}-1] \\ & + \left(h_3[1]h_2[2] + h_3[2]h_2[1]\right) x[\tfrac{n}{2}-\tfrac{3}{2}] \\ & + h_3[2]h_2[2]\,x[\tfrac{n}{2}-2] \end{aligned}$$

Solving for $y_2[n]$ and simplifying:

$$y_2[n] = w_1[n] + w_3[n]$$

$$\begin{aligned} y_2[n] = \; & \left(h_3[0]h_2[0] + h_1[0]\right) x[\tfrac{n}{2}] \\ & + \left(h_3[0]h_2[1] + h_3[1]h_2[0] + h_1[1]\right) x[\tfrac{n}{2}-\tfrac{1}{2}] \\ & + \left(h_3[0]h_2[2] + h_3[1]h_2[1] + h_3[2]h_2[0] + h_1[2]\right) x[\tfrac{n}{2}-1] \\ & + \left(h_3[1]h_2[2] + h_3[2]h_2[1]\right) x[\tfrac{n}{2}-\tfrac{3}{2}] \\ & + h_3[2]h_2[2]\,x[\tfrac{n}{2}-2] \end{aligned}$$

Now we can compare $y_1[n]$ and $y_2[n]$ and solve the system:

$$\begin{aligned} h_3[0]h_2[0] + h_1[0] &= a \\ h_3[0]h_2[1] + h_3[1]h_2[0] + h_1[1] &= b \\ h_3[0]h_2[2] + h_3[1]h_2[1] + h_3[2]h_2[0] + h_1[2] &= c \\ h_3[1]h_2[2] + h_3[2]h_2[1] &= d \\ h_3[2]h_2[2] &= e \end{aligned}$$

Now we can design $h_1[n]$, $h_2[n]$, and $h_3[n]$ to satisfy this system s.t. $y_1[n] = y_2[n]$

$h_1[0] = a$ $h_1[1] = c$ $h_1[2] = e$

$h_2[0] = b$ $h_2[1] = d$ $h_2[2] = 0$

$h_3[0] = 0$ $h_3[1] = 1$ $h_3[2] = 0$

b) For the first system, the signal is upsampled by 2 and then convolved w/ a 5-point impulse response, which requires 10 multiplications per output point.

The second system requires 8 multiplications per output point since the upsampling comes after $h_1[n]$ and $h_2[n]$ are applied and $h_3[n]$ is simply a shift filter.

4.65

$$H(e^{j\omega}) = \begin{cases} L, & |\omega| \le \omega_c \\ 0, & \text{else} \end{cases}$$

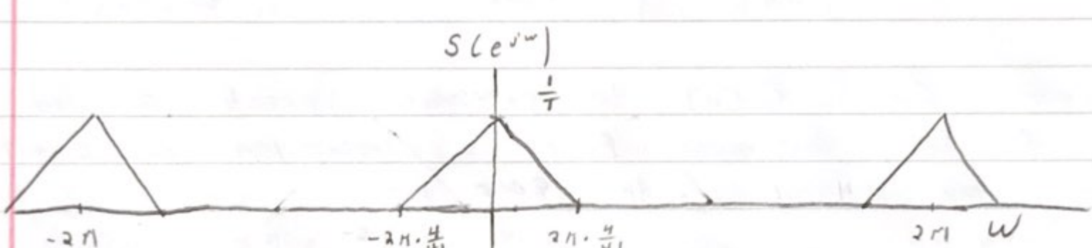
$$T = \frac{1}{44.1} \text{ ms} = \frac{1}{44.1 \times 10^3} \text{ s}$$

$$S(e^{j\omega}) = \frac{1}{T} \sum_{k=-\infty}^{\infty} S_c\left(j\left(\frac{\omega}{T} - \frac{2\pi k}{T}\right)\right)$$

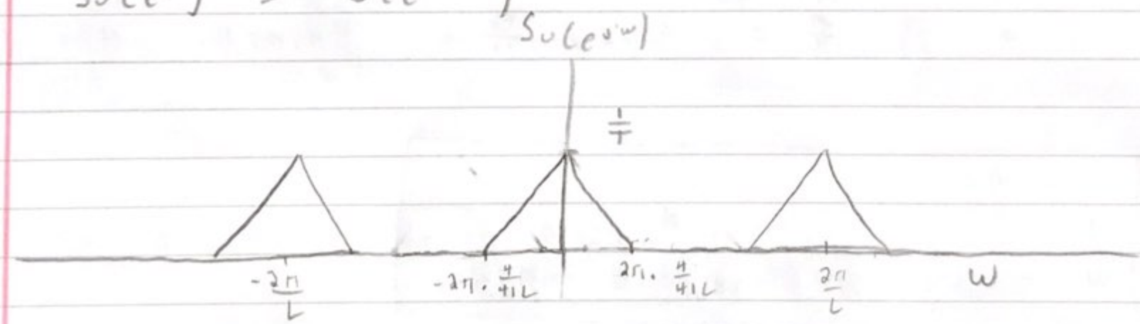
$$= (44.1 \times 10^3) \sum_{k=-\infty}^{\infty} S_c\left(j\left(44.1 \times 10^3 \omega - 44.1 \times 10^3 \times 2\pi k\right)\right)$$

$$\Omega_N = 2\pi \times 4000 \text{ rad/s}$$
$$\Omega_s = \frac{2\pi}{T} = 2\pi \times 44100 \text{ rad/s}$$

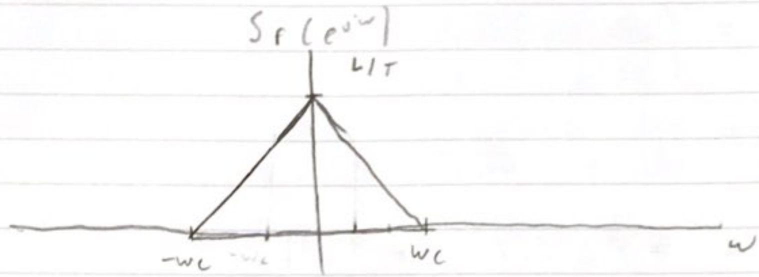
$\Omega_s \ge 2\Omega_N \implies$ Aliasing will not occur



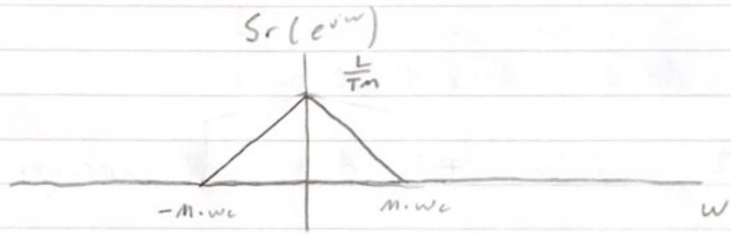
$$S_u(e^{j\omega}) = S(e^{j\omega L})$$



$$S_f(e^{j\omega}) = H(e^{j\omega})\, S_u(e^{j\omega})$$



$$S_r(e^{j\omega}) = \frac{1}{M}\sum_{i=0}^{M-1} S_f\left(e^{j(\omega/M - 2\pi i/M)}\right)$$



In order for $s_r[n]$ to represent speech sampled at 8 kHz, we must find a combination of L and M that maps 44100 Hz to 8000 Hz

$$\frac{TM}{L} = T'$$

$$\frac{M}{L} = \frac{T'}{T} = \frac{f_s}{f_s'} = \frac{44100 \text{ Hz}}{8000 \text{ Hz}} = \frac{441}{80}$$

$$\boxed{\begin{aligned} M &= 441 \\ L &= 80 \\ \omega_c &= \min(\pi/80,\ \pi/441) = \pi/441 \end{aligned}}$$

4.66

From the first system, we can define $H_{a0}(j\Omega)$ s.t. it satisfies Nyquist. Given that the C/D converter samples at 44 kHz, $H_{a0}(j\Omega)$ must introduce a bandlimit Ω_N that satisfies Nyquist: $\Omega_s \geq 2\Omega_N$

$$\Omega_s = \frac{2\pi}{T} = \pi \times 88 \times 10^3 \text{ rad/s}$$

$$\Rightarrow \quad \Omega_N = \pi \times 44 \times 10^3 \text{ rad/s}$$

$$H_{a0}(j\Omega) = \begin{cases} 1, & |\Omega| < \pi \times 44 \times 10^3 \\ 0, & |\Omega| \geq \pi \times 44 \times 10^3 \end{cases}$$

The maximum frequency that $H_{a0}(j\Omega)$ may pass is $\pi \times 44 \times 10^3$ rad/s to avoid aliasing.

Now look at the second system. In this case, $M = 4$ for the downsampling. We can design an anti-aliasing filter that has significant attenuation up until $M\Omega_N = 4\pi \times 44 \times 10^3$ rad/s, and does not pass beyond that. As demonstrated in Figure 4.49, this simple antialiasing filter will successfully avoid aliasing down the line when a combo of a sharp filter and a downsample effectively low-pass filter at Nyquist.

$$w_N = \frac{\pi}{4} \Longrightarrow \boxed{\begin{aligned} \Omega_p &= 11\pi \times 10^3 \text{ rad/s} \\ \Omega_s &= 2\pi/T - \Omega_N = 88\pi \times 10^3 - 11\pi \times 10^3 \text{ rad/s} = 77\pi \times 10^3 \text{ rad/s} \end{aligned}}$$

$$\boxed{H_{a1}(j\Omega) = \begin{cases} 1, & |\Omega| \leq 11\pi \times 10^3 \\ 0, & |\Omega| > 77\pi \times 10^3 \end{cases}}$$

Matlab Problem 1

$x(0) = 2$
$x(1) = 1$
$x(2) = -1$

a) $x(t) = A\cos(\omega t + \phi)$

i) $2 = A\cos(\phi) \implies \phi = \cos^{-1}\left(\frac{2}{A}\right)$
$1 = A\cos(\omega + \phi)$
$-1 = A\cos(2\omega + \phi)$

$A = \frac{2}{\cos\phi}$

$\frac{2\cos(\omega+\phi)}{\cos\phi} = 1 \qquad \frac{2\cos(2\omega+\phi)}{\cos\phi} = -1$

$\frac{\cancel{2}\cos(2\omega+\phi)}{\cancel{2}\cos(\omega+\phi)} = -1$

$A = 2$
$\phi = 2\pi n$
$1 = 2\cos(\omega + 2\pi n) = 2\cos\omega$
$\omega = \cos^{-1}\left(\frac{1}{2}\right) = \frac{\pi}{3}$

$\implies \boxed{\begin{array}{c} A = 2 \\ \phi = 2\pi \\ \omega = \frac{\pi}{3} \end{array}}$

ii) 3 equations and 3 unknowns $\implies$ theoretically, a solution is always available

$A = -2$

$\Phi = \pi$

$1 = -2\cos(\omega + \pi)$

$\omega + \pi = \cos^{-1}\left(-\frac{1}{2}\right)$

$\omega = \frac{2\pi}{3} - \pi = -\frac{\pi}{3}$

$$\boxed{\begin{array}{l} A = -2 \\ \Phi = \pi \\ \omega = -\frac{\pi}{3} \end{array}}$$

```python
# ESE 531: HW4 Problem 2

# libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

# part a
t_array = np.arange(0, 2.01, 0.01)

# first solved sinusoid
A1 = 2
phi1 = 2 * np.pi
w1 = 7 * np.pi / 3

x1 = [A1 * np.cos(w1 * t + phi1) for t in t_array]

# second solved sinusoid
A2 = -2
phi2 = np.pi
w2 = -np.pi / 3

x2 = [A2 * np.cos(w2 * t + phi2) for t in t_array]


# plot both solved sinusoids
fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.plot(t_array, x1, '-', label='$A=2$, $\omega=7\pi/3$, $\phi=2\pi$')
ax.plot(t_array, x2, '-', label='$A=-2$, $\omega=-\pi/3$, $\phi=\pi$')
ax.legend(loc='upper right')

major_ticks = np.arange(0, 2.1, 0.25)
minor_ticks = t_array

ax.set_xticks(major_ticks)
ax.set_xticks(minor_ticks, minor=True)

# And a corresponding grid
ax.grid(which='both')

# Or if you want different settings for the grids:
ax.grid(which='minor', alpha=0.2)
ax.grid(which='major', alpha=0.5)


ax.set_xlabel('Time (sec)')
ax.set_ylabel('x(t)')
plt.show()

# part b

# connect points with straight lines
x_d = [2, 1, -1]

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.plot(range(3), x_d, '-')

major_ticks = np.arange(0, 2.1, 0.25)
minor_ticks = t_array

ax.set_xticks(major_ticks)
ax.set_xticks(minor_ticks, minor=True)
```

```python
# And a corresponding grid
ax.grid(which='both')

# Or if you want different settings for the grids:
ax.grid(which='minor', alpha=0.2)
ax.grid(which='major', alpha=0.5)

ax.set_title("3 Samples Connected By Lines")
ax.set_xlabel('Time (sec)')
ax.set_ylabel('x(t)')
plt.show()

# zero-insert samples and convolve with a triangular pulse
tri_impulse = [0.2, 0.4, 0.6, 0.8, 1.0, 0.8, 0.6, 0.4, 0.2]
x_d = [2, 0, 0, 0, 0, 1, 0, 0, 0, 0, -1]

lin_interp = np.convolve(x_d, tri_impulse, mode='full')

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.plot(np.linspace(-1, 3.1, len(lin_interp)), lin_interp)

major_ticks = np.linspace(-1, 3.1, len(lin_interp))

ax.set_xticks(np.arange(-1, 3.1, 0.5))

# Or if you want different settings for the grids:
ax.grid()
ax.set_title("Convolution with Triangle Impulse")
ax.set_xlabel('Time (sec)')
ax.set_ylabel('x(t)')
plt.show()

# plot x_d but connected with lines
x_d_ext = [0, 2, 1, -1, 0]

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.plot(np.linspace(-1, 3.1, len(x_d_ext)), x_d_ext)

major_ticks = np.linspace(-1, 3.1, len(lin_interp))

ax.set_xticks(np.arange(-1, 3.1, 0.5))

# Or if you want different settings for the grids:
ax.grid()
ax.set_title("Samples Connected with Lines")
ax.set_xlabel('Time (sec)')
ax.set_ylabel('x(t)')
plt.show()




# fit a second-degree polynomial to the three data
def f(x, a, b, c):
   return a + (b * x) + (c * x ** 2)

popt, pcov = curve_fit(f, np.arange(3), [2, 1, -1])

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

ax.plot(np.arange(-5, 5.01, 0.01), f(np.arange(-5, 5.01, 0.01), *popt))
```

```python
major_ticks = np.arange(-5, 5.1, 1)
minor_ticks = np.arange(-5, 5.1, 0.1)

ax.set_xticks(major_ticks)
ax.set_xticks(minor_ticks, minor=True)

# And a corresponding grid
ax.grid(which='both')

# Or if you want different settings for the grids:
ax.grid(which='minor', alpha=0.2)
ax.grid(which='major', alpha=0.5)

ax.set_title("Second-Degree Polynomial Fit")
ax.set_xlabel('Time (sec)')
ax.set_ylabel('x(t)')
plt.show()

# part c

# sinc interpolator function
def sinc_interp(t, x, Ts):
    result = np.zeros_like(t)
    for i, val in enumerate(t):
        current = x[i] * np.sin(np.pi * (val - i * Ts) / Ts) / (np.pi * (val - i * Ts) * Ts)
        result = result + current
    return result

# interpolate a single point sample
t_array = np.arange(-5, 5.01, 0.01)

# generate single point sample (delta)
point_sample = np.zeros_like(t_array)
point_sample[int(len(point_sample) / 2)] = 1
Ts = 0.01

# periodic sinc
sinc_per = [np.sin(np.pi * t / Ts) / (np.pi * t / Ts) for t in t_array]

# convolve periodic sinc with delta function and plot result
point_sinc = np.convolve(point_sample, sinc_per, mode='same')

plt.plot(t_array, point_sinc)
plt.title("Reconstructed Point Sample using a Sinc Interpolation")
plt.xlabel("t")
plt.ylabel('$x_r(t)$')
plt.show()

# apply sinc interpolation to reconstruct sinusoid from part a
x_d = np.zeros_like(t_array)
x_d[int(len(x_d) / 2)] = 2
x_d[int(len(x_d) / 2) + int(1 / Ts)] = 1
x_d[int(len(x_d) / 2) + int(2 / Ts)] = -1
sinusoid_reconstruct = np.convolve(x_d, sinc_per, mode='same')

plt.plot(t_array, sinusoid_reconstruct)
plt.title("Reconstructed Sinusoid using a Sinc Interpolation")
plt.xlabel("t")
plt.ylabel('$x_r(t)$')
plt.show()
```