

Signals & Systems for Computer Engineering

Prof.Dr.Mustafa KAMAŞAK & Prof.Dr. B.Berk ÜSTÜNDAĞ

Istanbul Technical University

Faculty of Computer Engineering and Informatics

bustundag@itu.edu.tr

BLG354E / CRN: 21560
6th Week Lecture (cont.)

Implementation of Discrete Time Systems

Transfer function of DT-LTI system is given as,

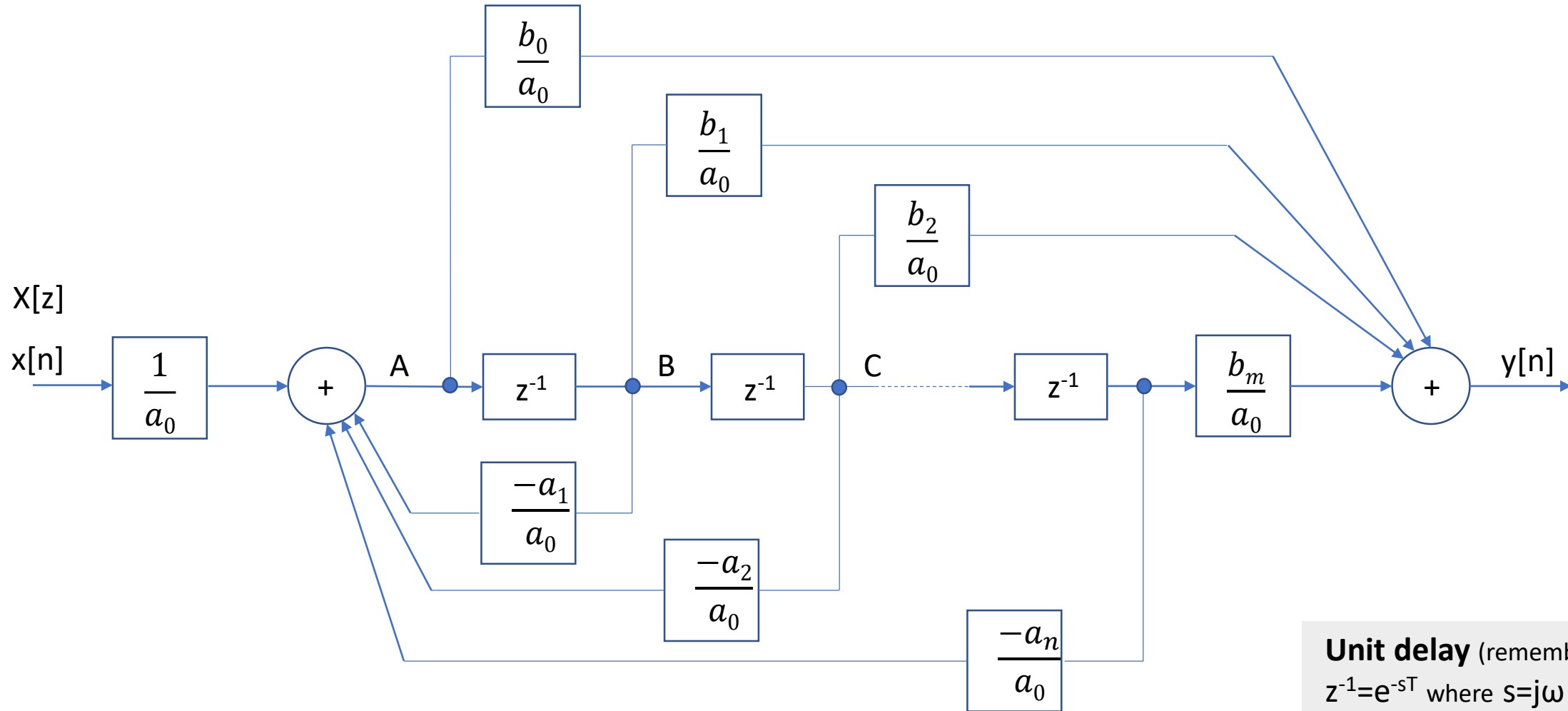
$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + \dots + b_m z^{-m}}{a_0 + a_1 z^{-1} + \dots + a_n z^{-n}}$$

$$H(z) = \frac{b_0 + b_1 z^{-1} + \dots + b_m z^{-m}}{a_0 + a_1 z^{-1} + \dots + a_n z^{-n}} \cdot \frac{A(z)}{A(z)} = \frac{Y(z)}{X(z)}$$

$$Y(z) = \frac{1}{a_0} (b_0 + b_1 z^{-1} + \dots + b_m z^{-m}) A(z)$$

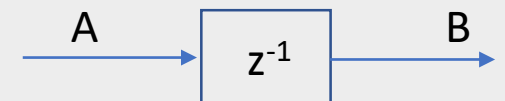
$$A(z) = \frac{1}{a_0} X(z) - \frac{1}{a_0} (a_1 z^{-1} + a_2 z^{-2} + \dots + a_n z^{-n}) A(z)$$

Direct Programming (canonical form)



Unit delay (remember):

$$z^{-1} = e^{-sT} \text{ where } s = j\omega$$



$$B = A \cdot z^{-1} \Rightarrow B(k) = A(k-1)$$

Example:

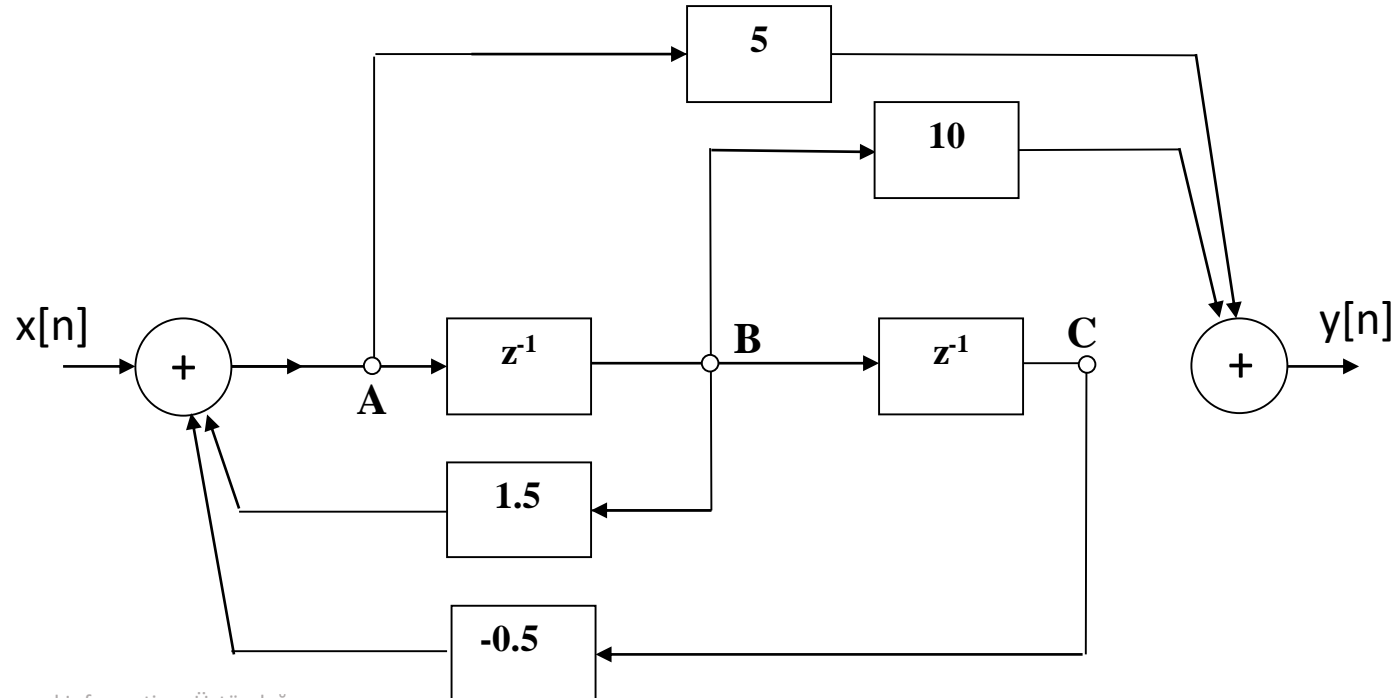
$$H(z) = \frac{Y(z)}{X(z)} = \frac{5(1+2z^{-1})}{(1-z^{-1})(1-0.5z^{-1})}$$

Write the Psuedo code H(z) for the signal sampled at f_s

$$H(z) = \frac{Y(z)}{X(z)} = \frac{(5+10z^{-1})A(z)}{(1-1.5z^{-1}+0.5z^{-2})A(z)}$$

$$Y(z) = (5 + 10 z^{-1}) A(z)$$

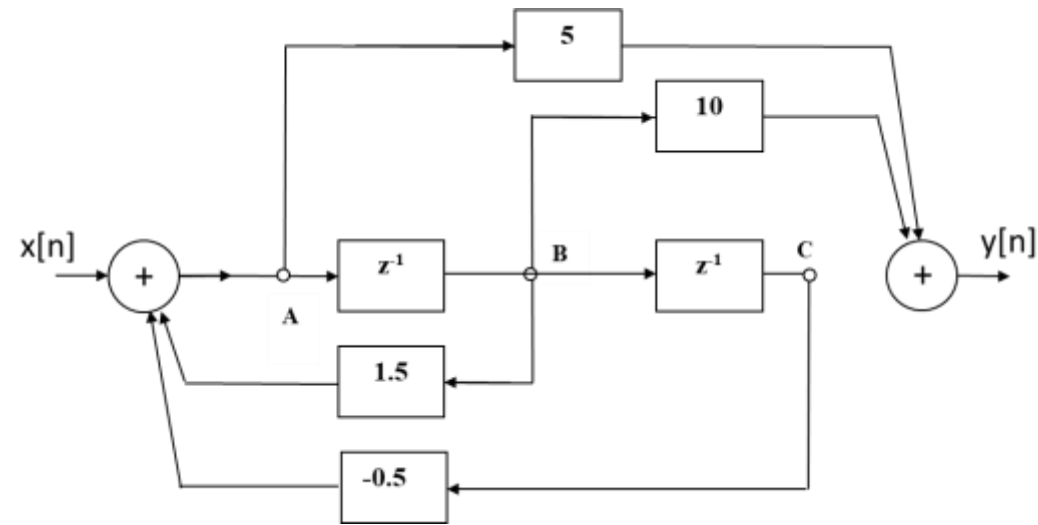
$$A(z) = X(z) + 1.5 z^{-1} A(z) - 0.5 z^{-2} A(z)$$



Pseudo code:
Timer Interrupt @ $T_s=1/f_s$

```
X = READ (ADC)
A = X + 1.5 B - 0.5 C
Y = 5 A + 10 B
Output (Y)
C=B
B=A
Return
```

Proof:



$$Y = 5A + 10B =$$

$$= 5A + 10Az^{-1} = (5 + 10z^{-1})A$$

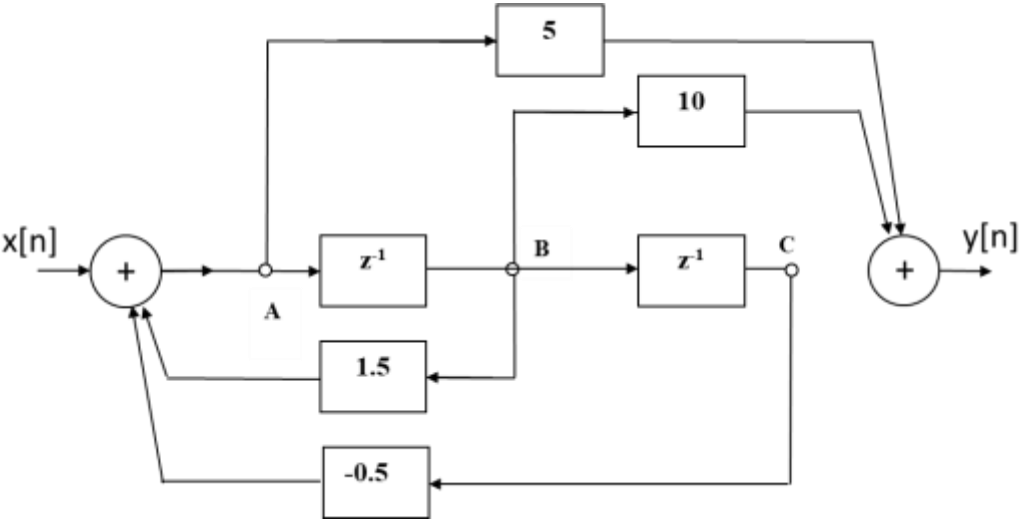
$$A = X + 1.5B - 0.5C$$

$$A = X + 1.5Az^{-1} - 0.5Az^{-2} \Rightarrow A = \frac{X}{1 - 1.5z^{-1} + 0.5z^{-2}}$$

$$Y = \frac{(5 + 10z^{-1})X}{1 - 1.5z^{-1} + 0.5z^{-2}} \Rightarrow T(z) = \frac{Y(z)}{X(z)} = \frac{(5 + 10z^{-1})}{(1 - 1.5z^{-1} + 0.5z^{-2})}$$

Step Response:

$x[n]=u[n] \Rightarrow \begin{cases} \text{for } n<0 \ x[n]=0 \Rightarrow y[n]=0 \\ \text{for } n\geq 0 \ x[n]=1 \end{cases}$



@T_s
t=n·T_s

Input X
A = X + 1.5 B – 0.5 C
Y = 5 A + 10 B
C=B
B=A
Output Y
Return

Initial condition

n	x[n]	A	B	C	y[n]
0	1	1	0	0	5
1	1	2.5	1	0	22.5
2	1	4.25	2.5	1	46.25
3	1	6.125	4.25	2.5	73.125

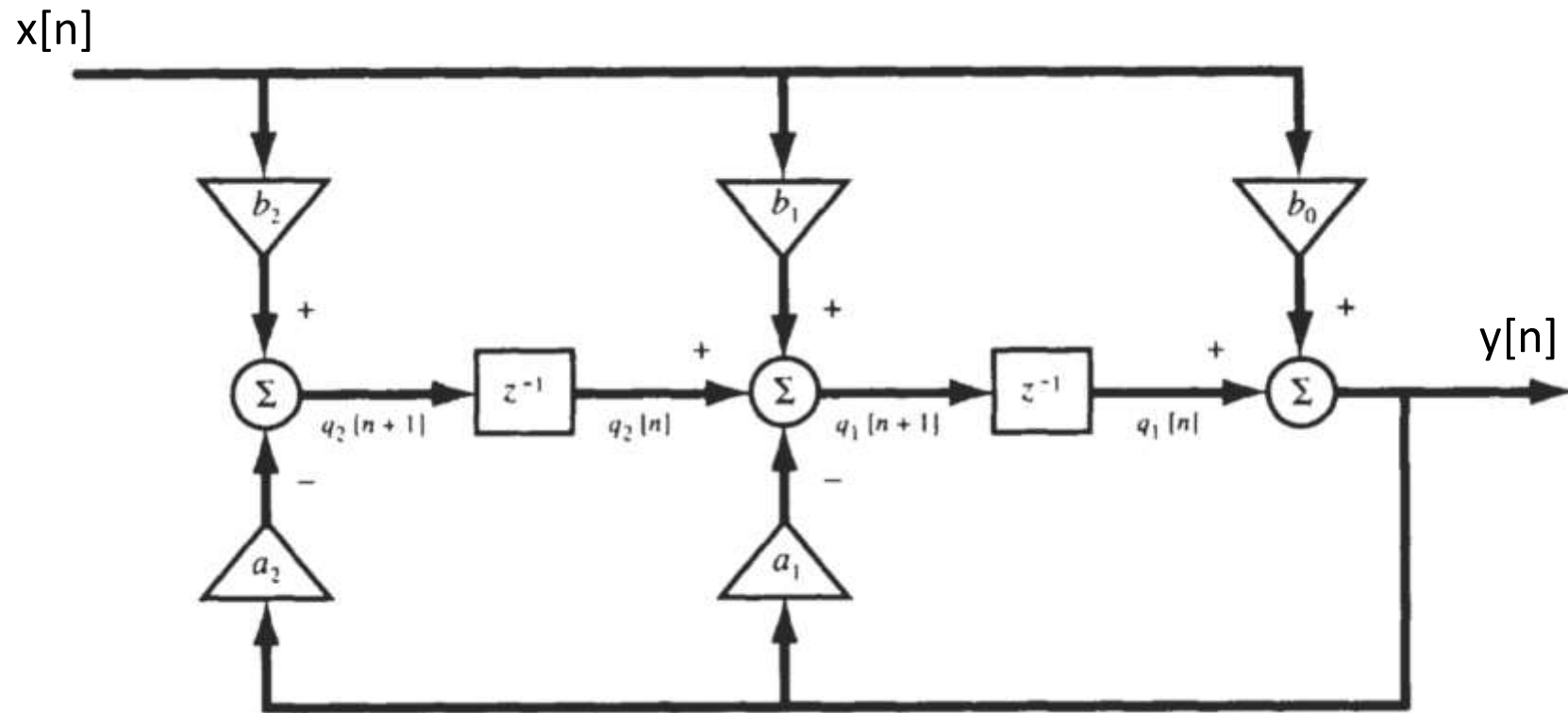
Canonical Simulation

(First form)

$$H(z) = \frac{Y(z)}{X(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

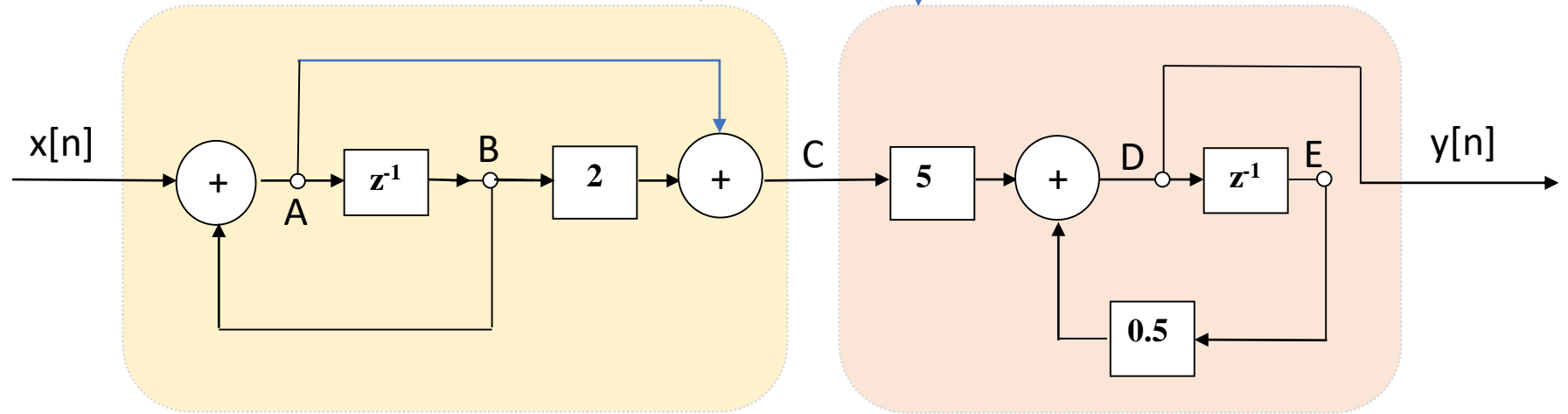
$$(1 + a_1 z^{-1} + a_2 z^{-2})Y(z) = (b_0 + b_1 z^{-1} + b_2 z^{-2})X(z)$$

$$Y(z) = -a_1 z^{-1} Y(z) - a_2 z^{-2} Y(z) + b_0 X(z) + b_1 z^{-1} X(z) + b_2 z^{-2} X(z)$$



Programming of Cascaded sub-systems

$$H(z) = \frac{Y(z)}{X(z)} = \underbrace{\frac{1+2z^{-1}}{1-z^{-1}}}_{\text{Sub-system 1}} \cdot \underbrace{\frac{5}{1-0.5z^{-1}}}_{\text{Sub-system 2}}$$

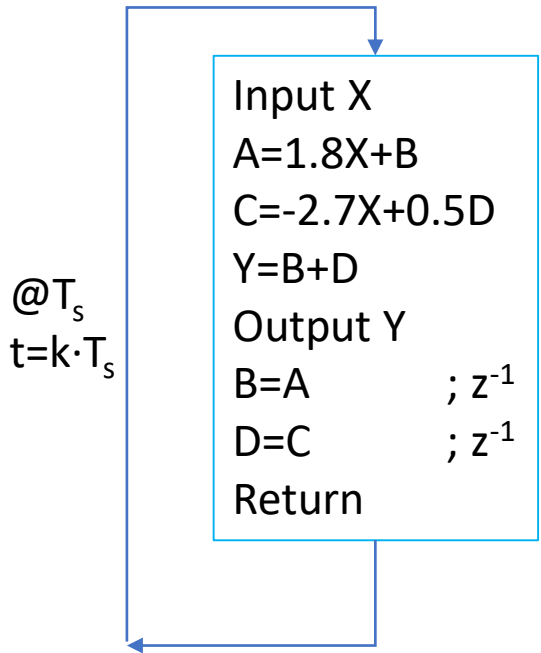


@ T_s
 $t = k \cdot T_s$

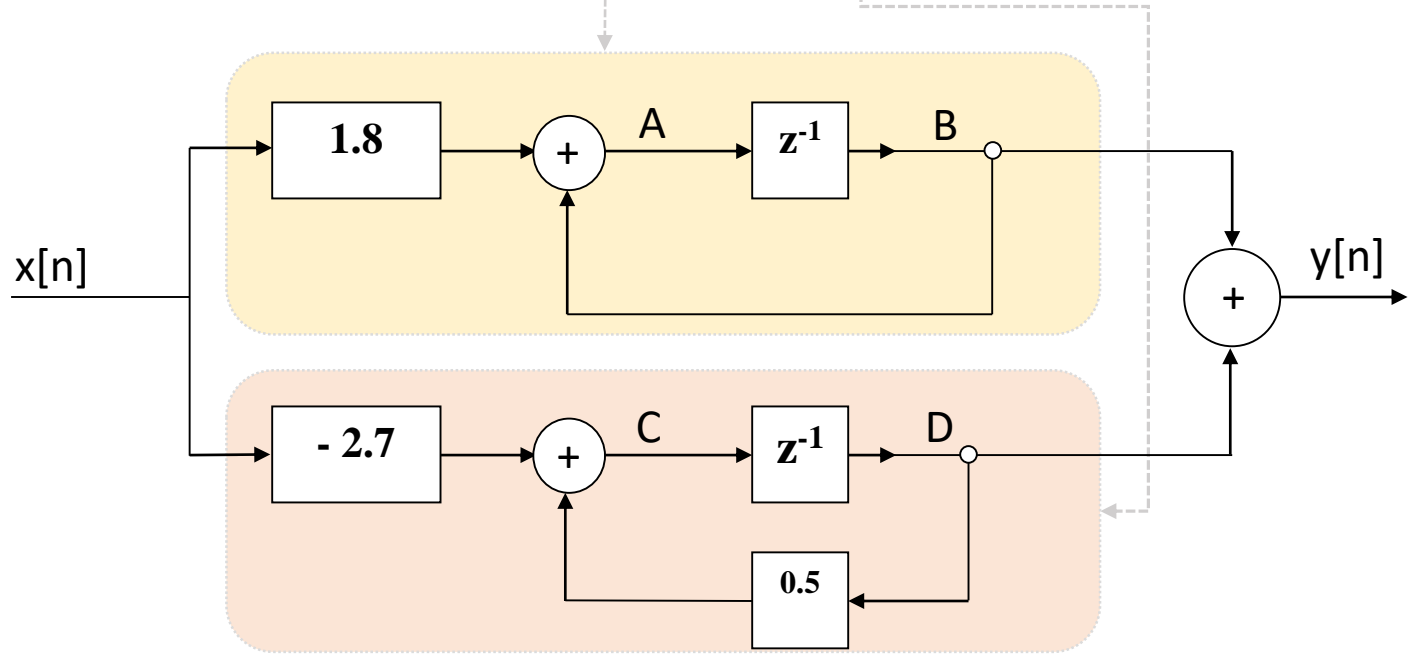
```

Input X
A=X+B
C=A+2B
D=5C+0.5E
Y=D
Output Y
E=D      ;  $z^{-1}$ 
B=A      ;  $z^{-1}$ 
Return
    
```


Programming of Parallel sub-systems



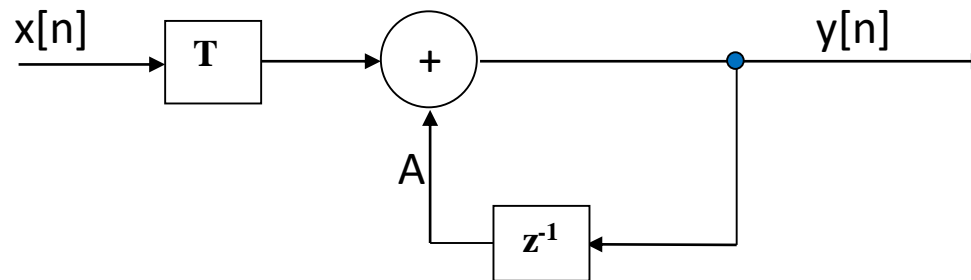
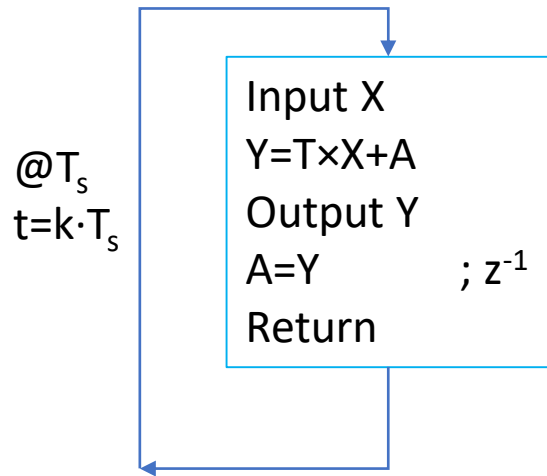
$$H(z) = \frac{Y(z)}{X(z)} = \underbrace{\frac{1.8z^{-1}}{1-z^{-1}}}_{\text{Top Subsystem}} - \underbrace{\frac{2.7z^{-1}}{1-0.5z^{-1}}}_{\text{Bottom Subsystem}}$$



Example:

Digital integrator without delay:

$$H(z) = \frac{Y(z)}{X(z)} = T \left(\frac{1}{1-z^{-1}} \right)$$

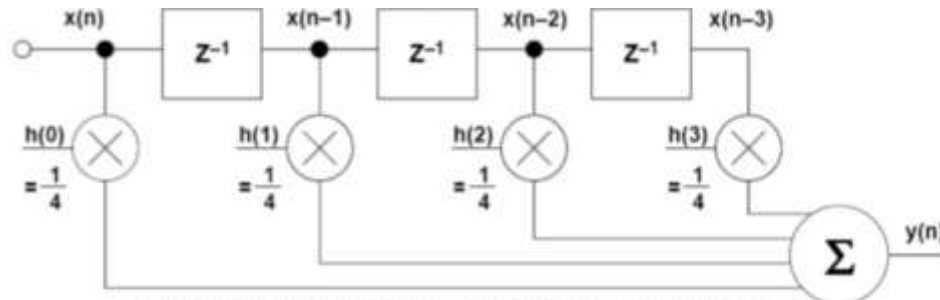


Initial condition: If $y(kT)=0$ for $k<0$ then $A=0$

$$Y[n]=y(kT)=T[x(0)+x(T)+x(2T)+...+x(kT)]$$

- 1) Simulate the system for $x[n]=u[n-2]-u[n-4]+\delta[n-3]$
- 2) What are the differences between integrator with and without Delay ?
- 3) Write the pseudo code for digital integrator with delay

Additional concerns:



$$y(n) = h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + h(3)x(n-3)$$

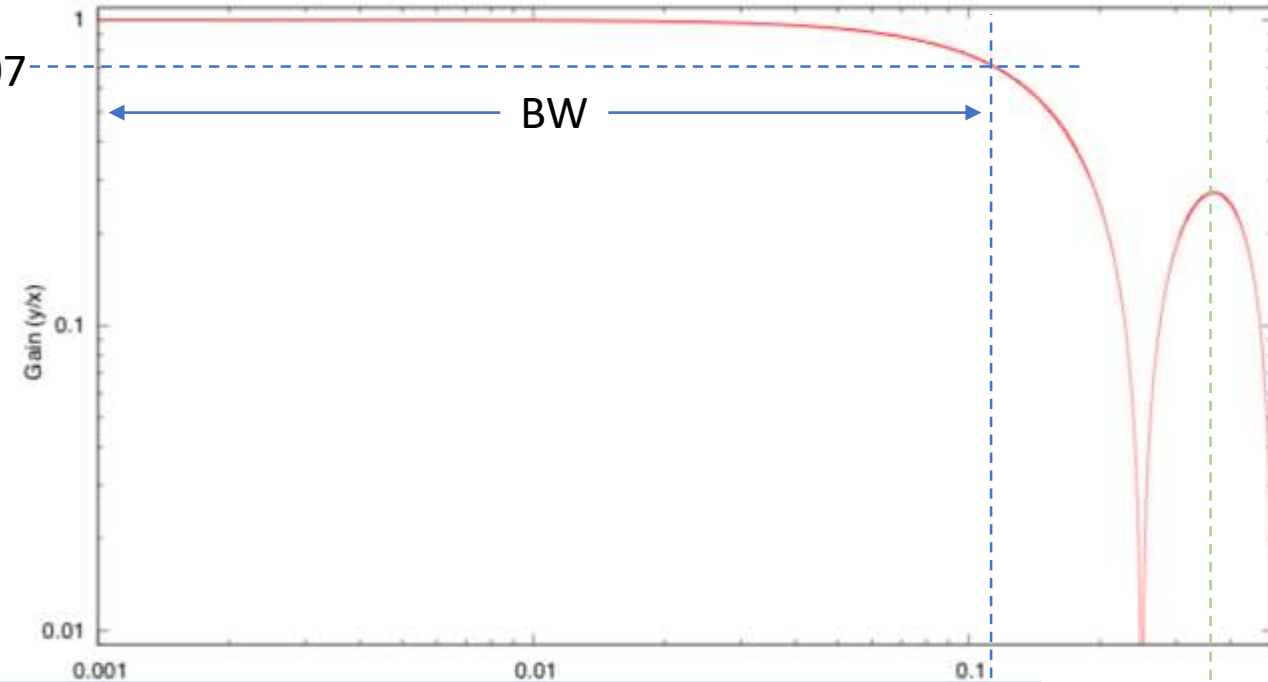
$$= \frac{1}{4}x(n) + \frac{1}{4}x(n-1) + \frac{1}{4}x(n-2) + \frac{1}{4}x(n-3)$$

$$= \frac{1}{4}[x(n) + x(n-1) + x(n-2) + x(n-3)]$$

$$H(z) = \frac{1}{4}(1 + z^{-1} + z^{-2} + z^{-3})$$

0.707

BW



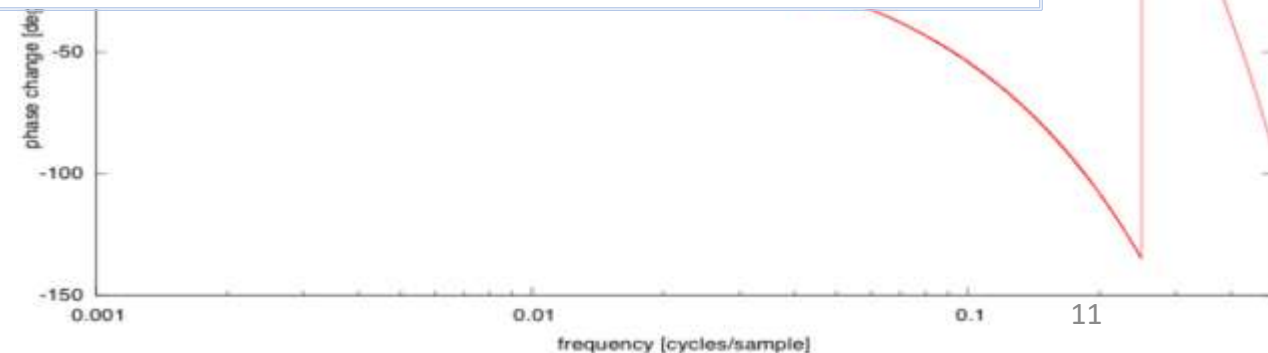
Question:

Design and Analysis problem

(e.g. if BW is given how to find h(i))

gnuplot s

```
1
2
3
4
5
6 set logscale y
7 set yrange [0.009:1.1]
8
9 set xlabel "frequency [cycles/sample]"
10 set logscale x
11 set xrange [0.001:0.5]
12
13 # transfer function
14 H(z) = (1+z**(-1)+z**(-2)+z**(-3))/4
15
16 j=sqrt(-1)
17 amplitude(omega) = abs(H(exp(j*omega)))
18 phase(omega) = imag(log(H(exp(j*omega)))) * 180/pi # phase in degrees
19
20 plot amplitude(2*pi*x) notitle
```



Programming with the assistance of ChatGPT3:

Question to ChatGPT3:



could you please write the transfer function of a second order butterworth type low pass digital filter in z-domain where the cut off frequency is 100Hz.



Sure! The transfer function of a second order Butterworth low pass digital filter in z-domain with a cutoff frequency of 100Hz can be written as:



$$H(z) = (b_0 + b_1z^{-1} + b_2z^{-2}) / (1 + a_1z^{-1} + a_2z^{-2})$$

where the coefficients b_0 , b_1 , b_2 , a_1 , and a_2 are determined as follows:

$$b_0 = (w_0^2) / (4 + 2w_0 + 2w_0^2)$$

$$b_1 = (2w_0^2) / (4 + 2w_0 + 2w_0^2)$$

$$b_2 = (w_0^2) / (4 + 2w_0 + 2w_0^2)$$

$$a_1 = (-8 + 2w_0^2) / (4 + 2w_0 + 2w_0^2)$$

$$a_2 = (4 - 2w_0 + 2w_0^2) / (4 + 2w_0 + 2w_0^2)$$

where $w_0 = \tan(\pi f_c / f_s)$, where f_c is the cutoff frequency (100Hz in this case) and f_s is the sampling frequency.

So, if we assume a sampling frequency of 1000Hz, we can calculate the coefficients as:

$$w_0 = \tan(\pi * 100 / 1000) = 0.157$$

$$b_0 = 0.002484$$

$$b_1 = 0.004968$$

$$b_2 = 0.002484$$

$$a_1 = -1.6907$$

$$a_2 = 0.7328$$

And the transfer function becomes:

$$H(z) = (0.002484 + 0.004968z^{-1} + 0.002484z^{-2}) / (1 - 1.6907z^{-1} + 0.7328z^{-2})$$

I hope this helps! Let me know if you have any further questions.



could you please write me a python code that draws the bode plot for the transfer function you provided above.



Sure! Here's a Python code snippet that uses the `scipy` library to generate the bode plot for the transfer function of a second order Butterworth low pass digital filter with a cutoff frequency of 100Hz:



python Copy code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import freqz

# Define the coefficients of the transfer function
b = [0.002484, 0.004968, 0.002484]
a = [1.0, -1.6907, 0.7328]

# Calculate the frequency response of the filter
w, h = freqz(b, a)

# Convert the angular frequency to Hz
f = w / (2*np.pi)

# Calculate the magnitude and phase of the frequency response
mag = 20*np.log10(np.abs(h))
phase = np.unwrap(np.angle(h)) * 180 / np.pi

# Plot the magnitude and phase responses
fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.semilogx(f, mag, 'b')
ax2.semilogx(f, phase, 'r')
ax1.set_xlabel('Frequency (Hz)')
ax1.set_ylabel('Magnitude (dB)', color='b')
ax2.set_ylabel('Phase (degrees)', color='r')
plt.title('Bode Plot')
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import freqz

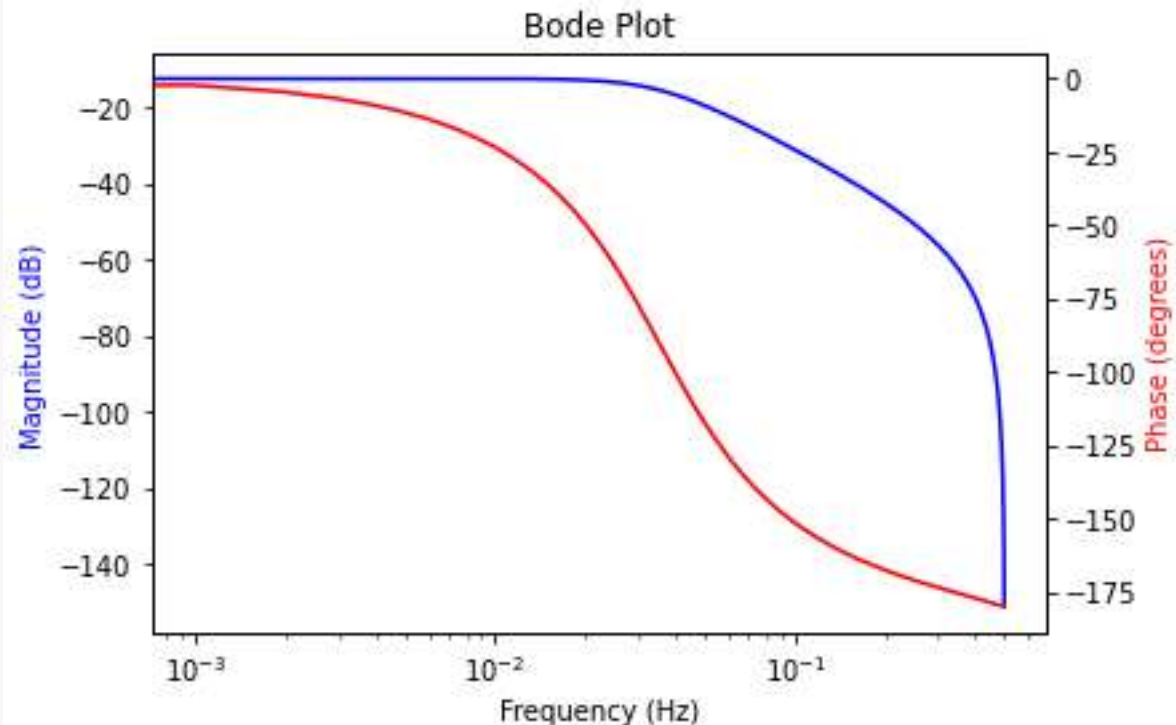
# Define the coefficients of the transfer function
b = [0.002484, 0.004968, 0.002484]
a = [1.0, -1.6907, 0.7328]

# Calculate the frequency response of the filter
w, h = freqz(b, a)

# Convert the angular frequency to Hz
f = w / (2*np.pi)

# Calculate the magnitude and phase of the frequency response
mag = 20*np.log10(np.abs(h))
phase = np.unwrap(np.angle(h)) * 180 / np.pi

# Plot the magnitude and phase responses
fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.semilogx(f, mag, 'b')
ax2.semilogx(f, phase, 'r')
ax1.set_xlabel('Frequency (Hz)')
ax1.set_ylabel('Magnitude (dB)', color='b')
ax2.set_ylabel('Phase (degrees)', color='r')
plt.title('Bode Plot')
plt.show()
```



scipy.signal.freqz

```
scipy.signal.freqz(b, a=1, worN=512, whole=False, plot=None, fs=6.283185307179586,  
include_nyquist=False)
```

Compute the frequency response of a digital filter.

Given the M-order numerator b and N-order denominator a of a digital filter, compute its frequency response:

$$H(e^{j\omega}) = \frac{B(e^{j\omega})}{A(e^{j\omega})} = \frac{b[0] + b[1]e^{-j\omega} + \dots + b[M]e^{-j\omega M}}{a[0] + a[1]e^{-j\omega} + \dots + a[N]e^{-j\omega N}}$$

Parameters: b : *array_like*

Numerator of a linear filter. If b has dimension greater than 1, it is assumed that the coefficients are stored in the first dimension, and $b.shape[1:]$, $a.shape[1:]$, and the shape of the frequencies array must be compatible for broadcasting.

a : *array_like*

Denominator of a linear filter. If b has dimension greater than 1, it is assumed that the coefficients are stored in the first dimension, and $b.shape[1:]$, $a.shape[1:]$, and the shape of the frequencies array must be compatible for broadcasting.

worN : {None, int, array_like}, optional

If a single integer, then compute at that many frequencies (default is N=512). This is a convenient alternative to:

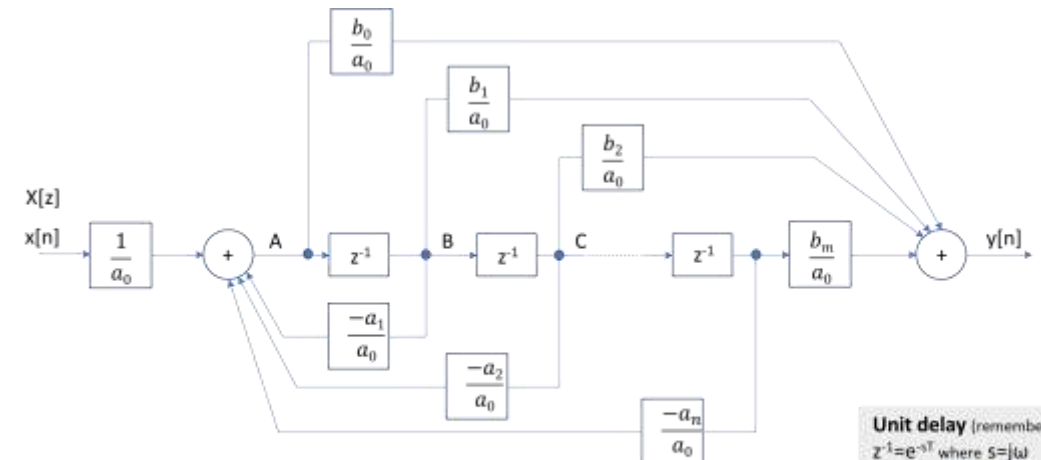
```
np.linspace(0, fs if whole else fs/2, N, endpoint=include_nyquist)
```

Using a number that is fast for FFT computations can result in faster computations (see Notes).

If an array_like, compute the response at the frequencies given. These are in the same units as fs .



Write the Pseudo code for real-time operation on an embedded system where the sampling period is 1000Hz



Unit delay (remember):
 $z^{-1} = e^{-sT}$ where $s = j\omega$
A $\xrightarrow{z^{-1}}$ B
 $B = A \cdot z^{-1} \Rightarrow B[k] = A[k-1]$



could you please write the pseudo code that performs the transfer function $H(z)$ you provided above



Sure, here's a pseudo code that performs the transfer function of a second order Butterworth low pass digital filter in z-domain with a cutoff frequency of 100Hz using the transfer function:

```

java
Copy code

// Initialize the filter coefficients
b0 = 0.002484
b1 = 0.004968
b2 = 0.002484
a1 = -1.6907
a2 = 0.7328

// Initialize the filter states
z1 = 0.0
z2 = 0.0

// Filter input samples
function butterworth_lpf(x):
    // Calculate output sample y
    y = b0*x + b1*z1 + b2*z2

    // Update filter states
    z2 = z1
    z1 = x - a1*z1 - a2*z2

    // Return output sample y
    return y

// Test the filter with a sine wave input
fs = 1000 // Sample rate
f = 50    // Input signal frequency
N = 1000  // Number of samples
t = linspace(0, 1, N)
x = sin(2*pi*f*t)

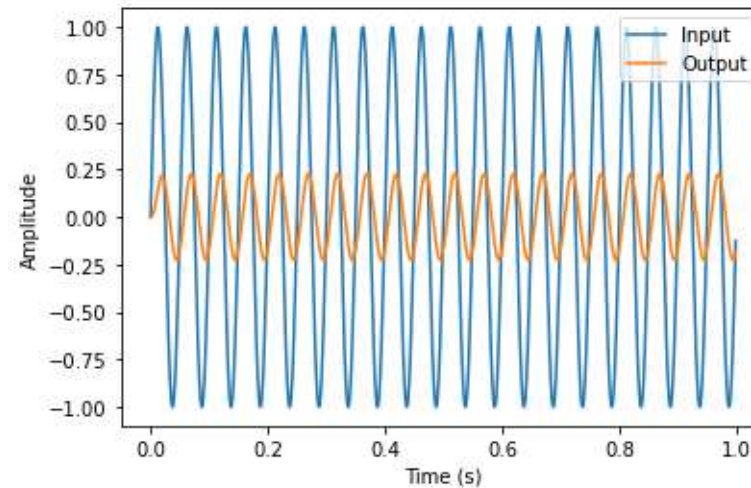
// Filter the input signal
y = zeros(N)
for i = 0 to N-1:
    y[i] = butterworth_lpf(x[i])

```

Python code



What is the inconsistency ?



How can it be corrected ?

```

import numpy as np
# Define the filter coefficients
b = [0.002484, 0.004968, 0.002484]
a = [1.0, -1.6907, 0.7328]

# Initialize the filter states
z1 = 0.0
z2 = 0.0
y1 = 0.0
y2 = 0.0

# Filter input samples
def butterworth_lpf(x):
    global z1, z2, y1, y2
    y = b[0]*x + b[1]*z1 + b[2]*z2 - a[1]*y1 - a[2]*y2
    z2 = z1
    z1 = x
    y2 = y1
    y1 = y
    return y

# Test the filter with a sine wave input
fs = 1000 # Sample rate
f = 20    # Input signal frequency
t = np.arange(0, 1, 1/fs)
x = np.sin(2*np.pi*f*t)

# Filter the input signal
y = np.zeros_like(x)
for i in range(len(x)):
    y[i] = butterworth_lpf(x[i])

# Plot the input and output signals
import matplotlib.pyplot as plt
plt.plot(t, x, label='Input')
plt.plot(t, y, label='Output')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.legend()
plt.show()

```