EE 387 – Signal Processing
Lab 2: Discrete Time Signals
E/20/420 – WANASINGHE J.K.

1. Understanding properties of Discrete Time Sinusoidal signals.

a. Plot the discrete time real sinusoidal signal $x[n] = 10\beta n$ for positive C when,

i. ß <- 1
ii. -1<ß < 0
iii. 0<ß < 1
iv. ß > 1

```
n = 0:15; % Define range of n
beta_values = [-1.5 -0.5 0.9 2.5]; % Define values for beta selected from the given ranges

% Loop through different beta values
for i = 1:length(beta_values)
    beta = beta_values(i);
    % Calculate x(n) for current beta
    x = 10 * beta.^n;
    % Plot the signal
    subplot(2,2,i);
    stem(n, x,'g');
    xlabel('n');
    ylabel('x(n)');
    title(sprintf('x(n) = 10(%g)^n', beta));
    grid on;
end
```

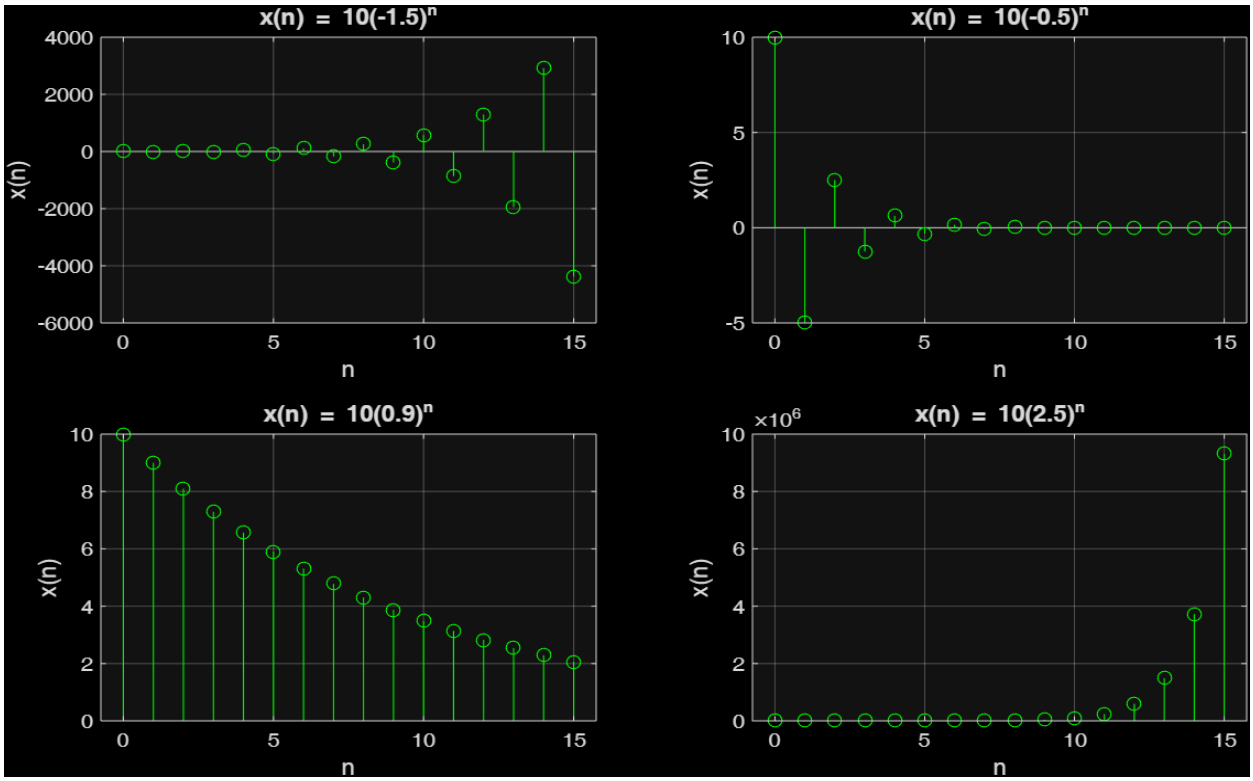

Figure 1: *Subplots of discrete time real sinusoidal signal $x[n]=10\beta^n$ for various ranges of β*

b. Plot x[n] and x (t) in the same plot for the following sinusoidal signals. Let n = kT where T = 5s and k E Z. That is x[n] is obtained by sampling x[t] at every 5 seconds. Determine the theoretical fundamental period of each signal. Is the observed period of the signal from the plot always equal to the theoretical period?

i. $x[n] = \cos\left(\frac{2\pi n}{12}\right), x(t) = \cos\left(\frac{2\pi t}{12}\right)$

ii. $x[n] = \cos\left(\frac{8\pi n}{31}\right), x(t) = \cos\left(\frac{8\pi t}{31}\right)$

```
% --------- First Signal: cos(2*pi*t/12) ---------
T = 5;                  % Sampling period in seconds for discrete signals
n = 0:T:200;            % Discrete time indices
x_n = cos(2 * pi * n / 12); % Discrete signal

fs_c = 1000;            % Sampling frequency for continuous signal (Hz)
t_c = 0:1/fs_c:200;     % Continuous time vector
x_c = cos(2 * pi * t_c / 12); % Continuous signal

figure(1);
plot(t_c, x_c, '-w');   % Plot continuous signal in black
hold on;
stem(n, x_n, 'filled'); % Overlay discrete samples
hold off;
title('cos(2\pi t/12)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
xlim([0 200]);          % Set x-axis limits
ylim([-1.2 1.2]);       % Set y-axis limits

% --------- Second Signal: cos(8*pi*t/31) ---------
T = 5;                  % Sampling period in seconds for discrete signals
n = 0:T:200;            % Discrete time indices
x_n = cos(8 * pi * n / 31); % Discrete signal

fs_c = 1000;            % Sampling frequency for continuous signal (Hz)
t_c = 0:1/fs_c:200;     % Continuous time vector
x_c = cos(8 * pi * t_c / 31); % Continuous signal

figure(2);
plot(t_c, x_c, '-w');   % Plot continuous signal in black
hold on;
stem(n, x_n, 'filled'); % Overlay discrete samples
hold off;
title('cos(8\pi t/31)');
xlabel('Time (s)');
ylabel('Amplitude');
grid on;
xlim([0 200]);          % Set x-axis limits
ylim([-1.2 1.2]);       % Set y-axis limits
```

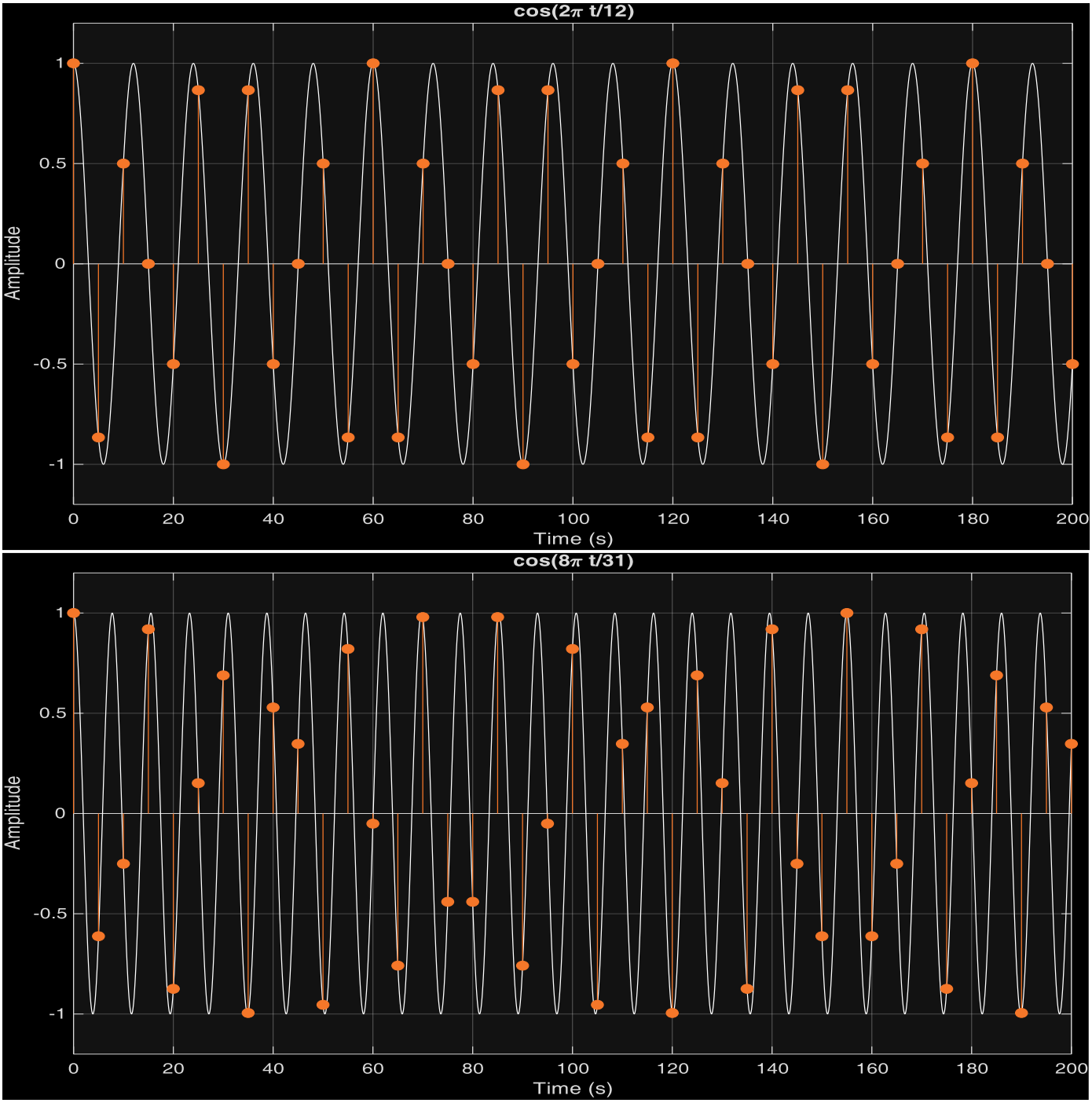

Figure 2: Plots of x[t] and x[n] on the same plot for given sinusoidal signals

- Theoretical Fundamental Periods of the two signals are 12 and 31/4 respectively
    i. $T = {}^{2\pi}/_{\pi/6} = 12$ time units
    ii. $T = {}^{2\pi}/_{8\pi/31} = {}^{31}/_{4} = 7.75$ time units

- The observed period of the signal from the plot is **not** always equal to the theoretical period. A discrete-time sinusoidal signal exhibits periodicity only when its fundamental frequency can be expressed as $\omega_0 = {}^{2\pi m}/_{N}$, where both N and m are positive integers. This means that for a discrete sinusoidal signal to be periodic, its normalized digital frequency ω_0 must be a rational fraction of 2π. The period N corresponds to the denominator of this rational fraction, while m represents the numerator.

- When the normalized digital frequency ω_0 fails to meet this rational multiple requirement, the discrete-time sinusoidal signal becomes aperiodic. In such cases, the period observed when plotting the signal will differ from what would be expected theoretically.

c. Plot the following nine discrete time signals in the same graph (use subplot command)

i. $x[n] = \cos(0.n)$
ii. $x[n] = \cos\left(\frac{\pi n}{8}\right)$
iii. $x[n] = \cos(\frac{\pi n}{4})$
iv. $x[n] = \cos(\frac{\pi n}{2})$
v. $x[n] = \cos(\pi n)$
vi. $x[n] = \cos(\frac{3\pi n}{2})$
vii. $x[n] = \cos(\frac{7\pi n}{4})$
viii. $x[n] = \cos(\frac{15\pi n}{8})$
ix. $x[n] = \cos(2\pi n)$

```
n = -10:10;                          % Discrete time index
axesRange = [-10 10 -1.5 1.5];       % Axis limits for all subplots

figure(1);

% 1. cos(0 * n)
x = cos(0 * n);                      % Zero frequency (DC)
subplot(3,3,1);
stem(n, x, 'filled');
axis(axesRange);
xlabel("n");
ylabel("Amplitude");
title("x = cos(0 \cdot n)");
grid on;

% 2. cos(pi * n / 8)
x = cos(pi * n / 8);                 % Low frequency
subplot(3,3,2);
stem(n, x, 'filled');
axis(axesRange);
xlabel("n");
ylabel("Amplitude");
title("x = cos(\pi/8)");
grid on;

% 3. cos(pi * n / 4)
x = cos(pi * n / 4);                 % Higher frequency
subplot(3,3,3);
stem(n, x, 'filled');
axis(axesRange);
xlabel("n");
ylabel("Amplitude");
title("x = cos(\pi/4)");
grid on;

% 4. cos(pi * n / 2)
x = cos(pi * n / 2);                 % Even higher frequency
subplot(3,3,4);
stem(n, x, 'filled');
axis(axesRange);
xlabel("n");
ylabel("Amplitude");
title("x = cos(\pi/2)");
grid on;

% 5. cos(pi * n)
x = cos(pi * n);                     % Alternating sequence
subplot(3,3,5);
stem(n, x, 'filled');
axis(axesRange);
xlabel("n");
```

```
ylabel("Amplitude");
title("x = cos(\pin)");
grid on;

% 6. cos(3*pi*n/2)
x = cos(3 * pi * n / 2);              % Higher frequency
subplot(3,3,6);
stem(n, x, 'filled');
axis(axesRange);
xlabel("n");
ylabel("Amplitude");
title("x = cos(3\pin/2)");
grid on;

% 7. cos(7*pi*n/4)
x = cos(7 * pi * n / 4);              % Near Nyquist frequency
subplot(3,3,7);
stem(n, x, 'filled');
axis(axesRange);
xlabel("n");
ylabel("Amplitude");
title("x = cos(7\pin/4)");
grid on;

% 8. cos(15*pi*n/8)
x = cos(15 * pi * n / 8);             % Very high frequency
subplot(3,3,8);
stem(n, x, 'filled');
axis(axesRange);
xlabel("n");
ylabel("Amplitude");
title("x = cos(15\pin/8)");
grid on;

% 9. cos(2*pi*n)
x = cos(2 * pi * n);                  % Periodic with period 1
subplot(3,3,9);
stem(n, x, 'filled');
axis(axesRange);
xlabel("n");
ylabel("Amplitude");
title("x = cos(2\pin)");
grid on;
```

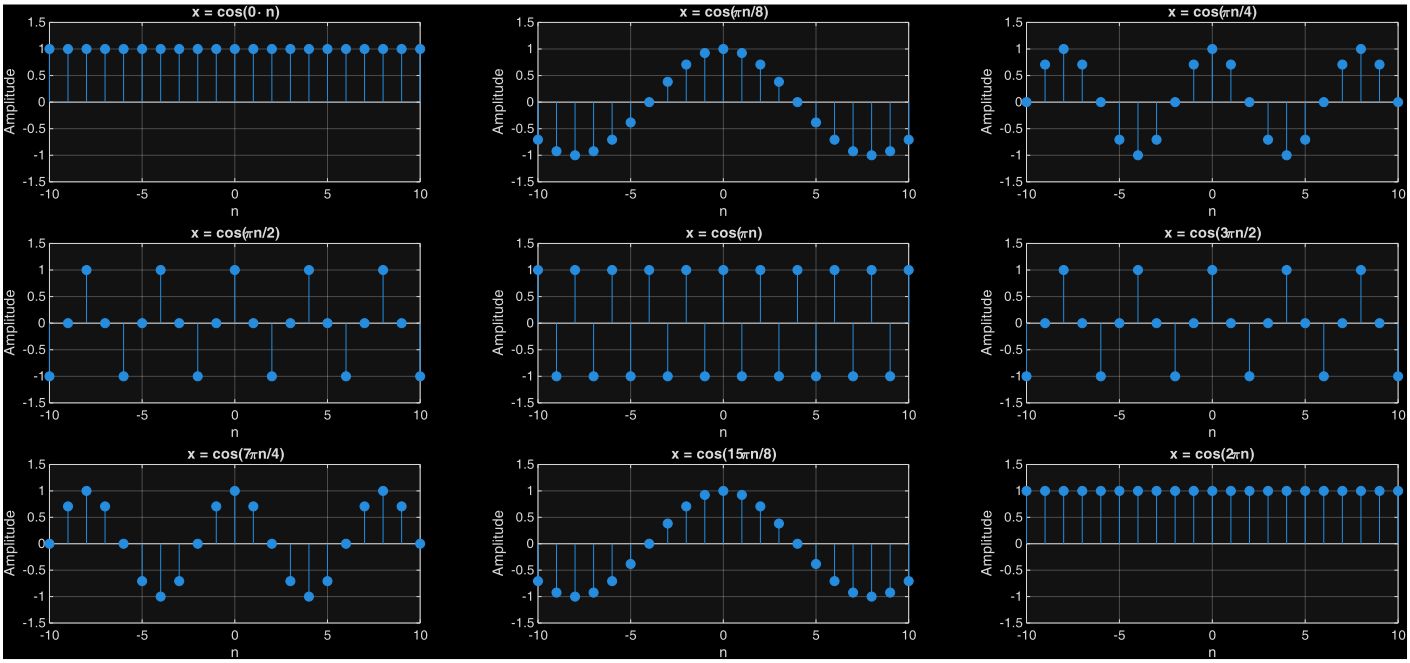

Figure 3: Subplots for Discrete Time Sinusoidal Plots for different frequencies

d) By observing the plots you have obtained in question 1.c, what can you tell about the shape of the signal as discrete frequency is varied?

When examining plots of cosine functions at various discrete frequencies, the waveform characteristics change significantly based on their relationship to the Nyquist frequency threshold of 0.5 Hz. Frequencies exceeding this limit experience aliasing distortion.

Frequencies Below the Nyquist Threshold

Zero Frequency Signal:

- The function cos(0 * n) produces a constant horizontal line at unity amplitude, as expected for a DC component.

Sub-Nyquist Frequencies:

- Functions like cos(π * n / 8) and cos(π * n / 4) display proper sinusoidal characteristics since their frequencies remain under the Nyquist boundary. These signals maintain their intended periodic behavior with clearly identifiable peaks that reflect their respective frequencies.

At the Nyquist Boundary

Critical Frequency Response:

- The signal cos(π * n / 2), operating exactly at the Nyquist frequency, exhibits minimal sampling with only two sample points per complete cycle.

Frequencies Above the Nyquist Threshold

Aliasing Effects:

- cos(π * n) operates at the Nyquist boundary, creating an alternating binary pattern oscillating between +1 and -1 values.
- Higher frequency signals including cos(3π * n / 2), cos(7π * n / 4), cos(15π * n / 8), and cos(2π * n) all exceed the Nyquist limit, causing their plots to deviate from true sinusoidal representation due to aliasing artifacts.

Specific Aliasing Manifestations:

- cos(3π * n / 2) undergoes frequency folding to π/2, creating a phase-shifted sinusoid
- cos(7π * n / 4) aliases down to π/4, producing altered frequency and phase characteristics
- cos(15π * n / 8) folds to π/8, similarly distorting both frequency and phase properties
- cos(2π * n), being a harmonic of the sampling frequency, collapses to a constant value of either +1 or -1 across all sample points

2. Discrete Convolution

a. Write a matlab function to implement discrete convolution for n > 0. Note that

$y[n] = x[n] * h[n]$ is given by the convolution summation

$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$

```
function y = discrete_conv(x, h)
% Discrete convolution: y[n] = sum_k x[k] * h[n-k+1]
nx = length(x);
nh = length(h);
N = nx + nh - 1; % Length of the convolution result
y = zeros(1, N); % Initialize output vector

for n = 1:N
    for k = 1:nx
        if (n - k + 1 > 0) && (n - k + 1 <= nh)
            y(n) = y(n) + x(k) * h(n - k + 1);
        end
    end
end
end
```

b. Using the function written in section a, convolve $x[n] = 0.5^n u(n)$ with $h[n] = u[n]$. Plot the output signal along with the two input signals.

```
n = 0:20;
% x[n] = 0.5^n * u[n]
x_n = 0.5.^n .* unit_step(n);
% h[n] = unit step
h_n = unit_step(n);
% y[n] = x[n] * h[n] (convolution)
y = a(x_n, h_n);

figure(1);
subplot(3, 1, 1);
stem(n, x_n);
xlabel('n');
ylabel('x[n]');
title('x[n] = 0.5^n \times u[n] (Exponential Sequence)');
xlim([0 20]);
ylim([0 1.5]);
grid on;

subplot(3, 1, 2);
stem(n, h_n);
xlabel('n');
ylabel('h[n]');
title('h[n] = u[n] (Unit Step Sequence)');
xlim([0 20]);
ylim([-0.1 1.5]);
grid on;

subplot(3, 1, 3);
stem(0:40, y(1:41));
xlabel('n');
ylabel('y[n]');
title('y[n] = x[n] * h[n] (Convolution Output)');
xlim([0 40]);
ylim([0 max(y(1:21))*1.5]);
grid on;

function y = unit_step(n)
y = double(n >= 0);
end
```

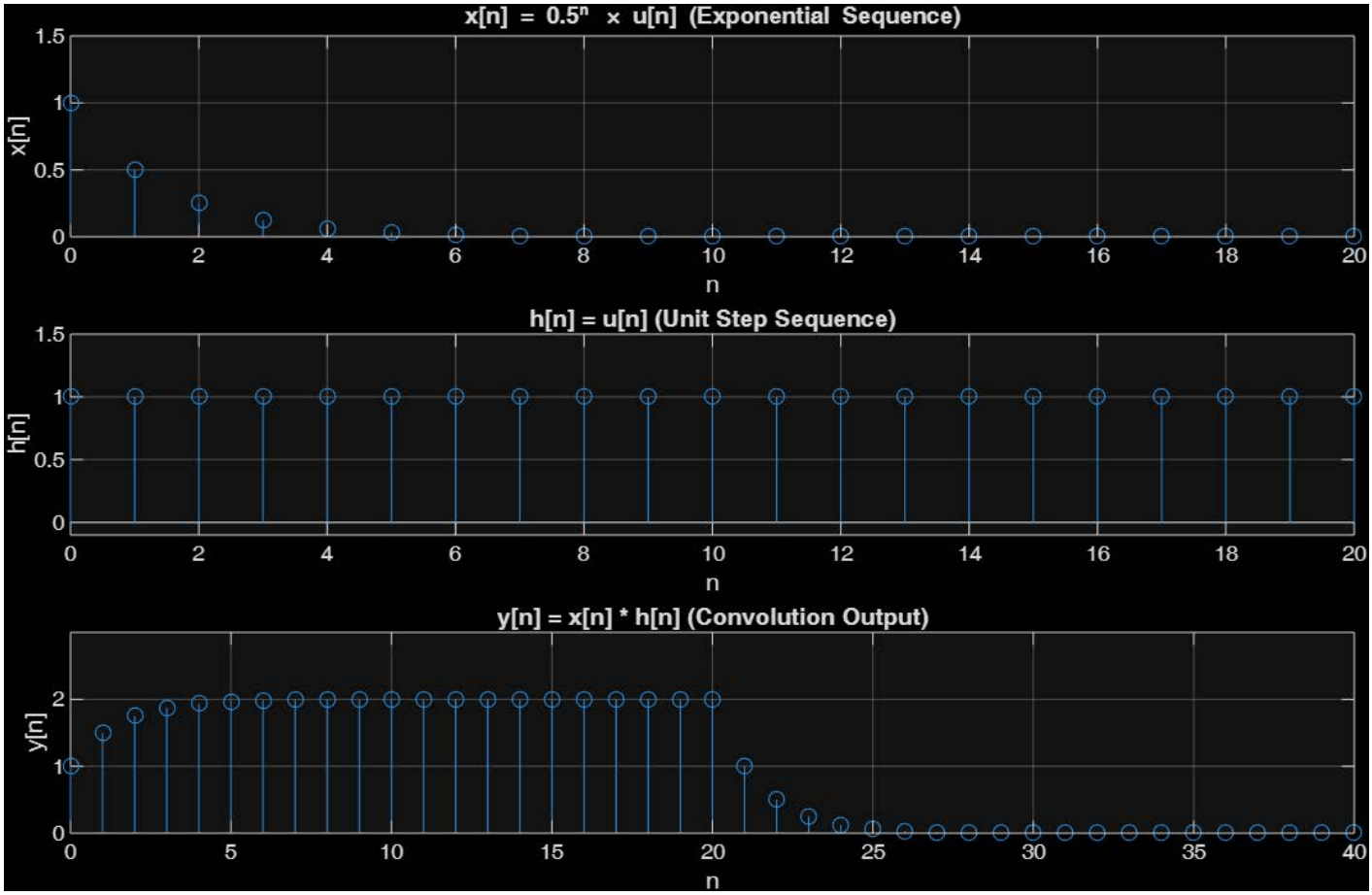

Figure 4: Convolution output plot along with the two inputs derived using the implemented convolution code

c. Consider the following two signals

i. X[n]=[1 1 1 1 1 0 0 0 0 0 0 0 0 0 0]

ii. h[n]=[2 4 8 16 32 64 0 0 0 0 0 0 0 0 0]

iii. Convolve the two signals using the function written in part a. Use matlab conv command to verify your answer.

```
h_n = [2 4 8 16 32 64 0 0 0 0 0 0 0 0 0];
x_n = [1 1 1 1 1 0 0 0 0 0 0 0 0 0 0];
% Calculate convolution
y = a(x_n, h_n);
y_c = conv(x_n, h_n);

% Plot the generated sequences and convolution results
figure(1);

subplot(4, 1, 1);
stem(0:length(x_n)-1, x_n);
xlabel('n');
ylabel('x[n]');
title('x[n]');
ylim([0 max(x_n)*1.2]);
grid on;

subplot(4, 1, 2);
stem(0:length(h_n)-1, h_n);
xlabel('n');
ylabel('h[n]');
title('h[n]');
ylim([0 max(h_n)*1.2]);
grid on;

subplot(4, 1, 3);
```

```
stem(0:length(y)-1, y);
xlabel('n');
ylabel('y[n]');
title('y[n] = x[n] * h[n] (custom function)');
ylim([0 max(y)*1.2]);
grid on;

subplot(4, 1, 4);
stem(0:length(y_c)-1, y_c);
xlabel('n');
ylabel('y[n]');
title('y[n] = x[n] * h[n] (MATLAB conv)');
ylim([0 max(y_c)*1.2]);
grid
```

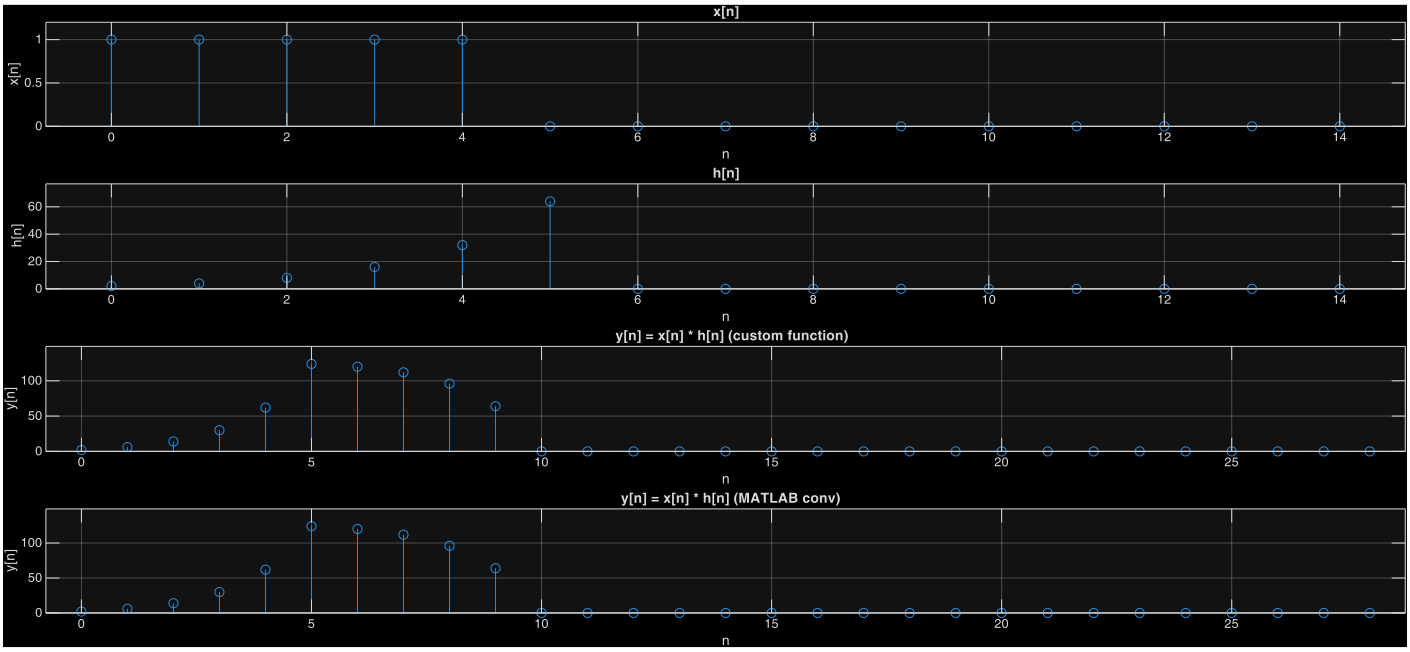

Figure 5: Convolution Plots using both conv function and the custom function implemented

iv. Considering the shape of the signal h[n] and the output signal, what sort of a transformation has been applied through the convolution operation?

The transformation seems like a “sliding window summation”, or a moving average without the normalization. The code below implements a window summation of window length 5

```
h_n = [2 4 8 16 32 64 0 0 0 0 0 0 0 0 0];

% Sliding window summation (window length 5) applied to h[n]
y_sum = get_previous_sum(h_n);

% Plot the result
figure;
stem(0:length(y_sum)-1, y_sum, 'filled');
xlabel('n');
ylabel('y_{sum}[n]');
title('Sliding Window Summation of h[n] (window length 5)');
grid on;

function y_n = get_previous_sum(x_n)
    N = length(x_n);
    y_n = zeros(1, N);
    for n = 1:N
        start_index = max(1, n - 4);
        y_n(n) = sum(x_n(start_index:n));
    end
end
```

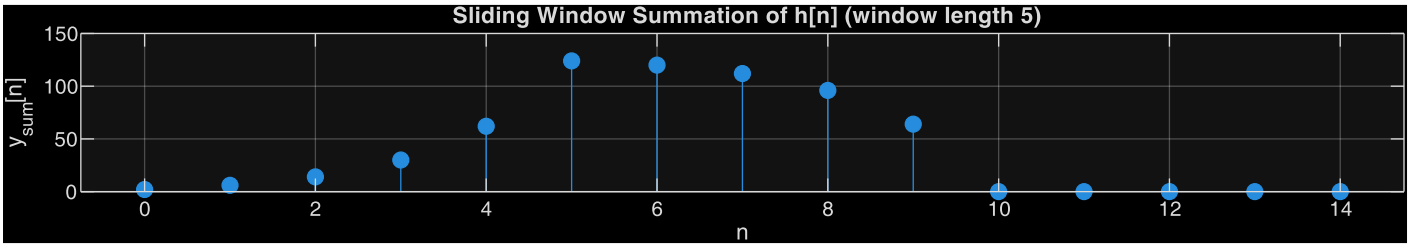

Figure 6: Sliding window summation

By, comparing the Figure 5 and Figure 6, it can be seen that the answer of the convolution is equal to the answer generated by the sliding windows summation.

3) LTI Systems

a. Consider the following processes. Identify input x[n] and the output y[n] for each case. Implement a matlab function to implement the given system.

i. An investor is maintaining a bank account. The bank pays him a monthly interest of 1%. It is given that the net savings he makes is P. Write a function to calculate his current bank balance B in terms of B and P.

let; new saving be $x[n]$ and bank balance at that month be $y[n]$ where the month of saving is n;

$$y[n] = 1.01y[n-1] + x[n]$$

Matlab code with the given parameters;

```
% Calculate the current bank balance using the formula:
% B_current = B_previous * 1.01 + P
% y[n] = y[n-1] * 1.01 + x[n]
function B_current = a_i(B_previous, P)
    B_current = B_previous * 1.01 + P;
end
```

ii. A merchant earns M amount of money monthly. He spends half of it and retains the rest of it as savings. Write a function to calculate the amount of money he has as savings.

let; the current saving be y[n], and the amount of monthly earning bee x[n] at the month n.

$$y[n] = y[n-1] + 0.5x[n]$$

Matlab code with the given parameters;

```
% Calculate the amount of savings
% y[n] = y[n-1] + m[n]/2;
function savings = a_ii(previous_savings, M)
    savings = previous_savings + M / 2;
end
```

b. Find the impulse response of the above two LTI systems. Hint: you may use convolution function to obtain the impulse response.

Analytical Derivation

(i) $y[n] = 1.01\, y[n-1] + x[n]$

checking with the impulse response;

$$x[n] = \delta[n]$$
$$y[n] = h[n]$$
$$\therefore h[n] = 1.01\, h[n-1] + \delta[n]$$

assuming causality $h[-1] = 0$

$$h[0] = 1.01 \times h[-1] + \delta[0]$$
$$h[0] = 1$$

$$h[1] = 1.01\, h[0] + \delta[1]$$
$$= 1.01 \times 1$$
$$= 1.01$$

$$h[2] = 1.01 \cdot h[1] + \delta[2]$$
$$= 1.01^2$$

$$h[3] = 1.01^3$$

impulse response $h[n] = 1.01^n$; $\forall\ n \in \mathbb{Z}^+$

Matlab Implementation:

```
% Define the impulse input
n = 0:40;
delta = zeros(size(n));
delta(1) = 1; % δ[n] = 1 for n = 0

% Initial balance
B_previous = 0;

% Output array
y_n = zeros(1, length(n));

% Simulate the system for each time step
for i = 1:length(n)
    % Apply the bank_balance function with the current impulse input
    y_n(i) = bank_balance(B_previous, delta(i));
    % Update the previous balance for the next iteration
    B_previous = y_n(i);
end

% Plot the output
stem(n, y_n);
xlabel('Time Index (n)');
ylabel('Output (y_n)');
title('Output for Impulse Input (Initial Balance = 0)');

% Calculate the current bank balance using the formula:
% B_current = B_previous * 1.01 + P
% y[n] = y[n-1] * 1.01 + x[n]
function B_current = bank_balance(B_previous, P)
    B_current = B_previous * 1.01 + P;
end
```

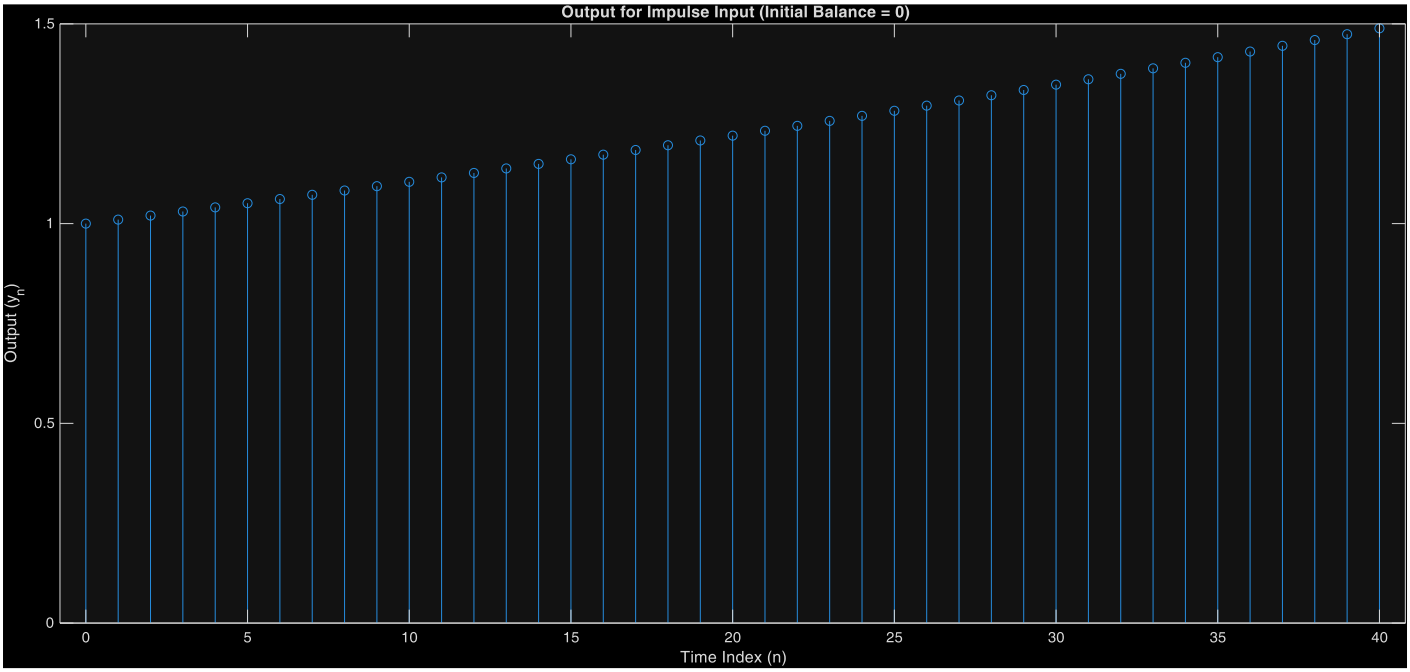

Figure 7: Impulse response of the bank balance LTI System

Analytical Deriving;

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

Checking with the impulse response;

$x[n] = \delta[n]$

$y[n] = h[n]$

assuming causality $h[-1] = 0$

$$h[0] = h[-1] + \frac{1}{2}\delta[0]$$
$$= \frac{1}{2}$$

$$h[1] = h[0] + \frac{1}{2}\delta[1]$$
$$= \frac{1}{2}$$

$$h[2] = \frac{1}{2}$$

$$h[3] = \frac{1}{2}$$

impulse response; $h[n] = \frac{1}{2}$ ~~if~~ $n \in \mathbb{Z}^+$

Matlab Implementation:

```
% Define the impulse input δ[n]
n = 0:40;
delta = zeros(size(n));
delta(1) = 1; % δ[n] = 1 for n = 0

previous_savings = 0;

% Output array
y_n = zeros(1, length(n));

% Simulate the system for each time step
for i = 1:length(n)
    % Apply the merchant_savings function with the current impulse input
    y_n(i) = merchant_savings(previous_savings, delta(i));
    % Update the previous balance for the next iteration
    previous_savings = y_n(i);
end

% Plot the output
stem(n, y_n);
xlabel('Month Index (n)');
ylim([0 1]);
ylabel('Output (y_n)');
title('Output for Impulse Input (Initial Savings = 0)');

% Calculate the amount of savings
% y[n] = y[n-1] + m[n]/2;
function savings = merchant_savings(previous_savings, M)
    savings = previous_savings + M / 2;
end
```

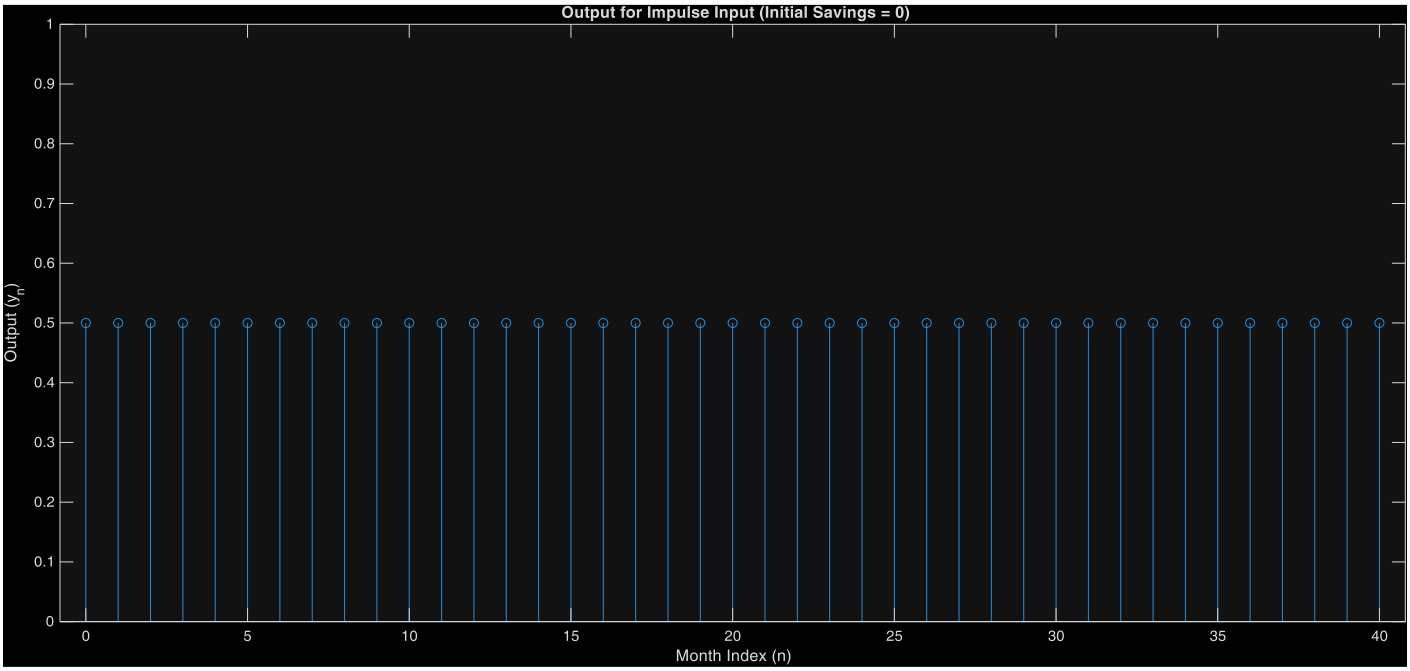

Figure 8: Impulse response of the merchant's savings LTI System

c. Based on the results obtained at part b, classify the two LTI systems into IIR or FIR.

i) bank_balance system – IIR System
The impulse response for this LTI system is given by $h[n] = 1.01^n$, which is an exponential sequence. This means the system's output depends on an infinite series of past input values, each multiplied by a corresponding exponential factor. Since the exponential function never actually reaches zero for any n, the impulse response contains infinitely many non-zero terms. Therefore, this system is classified as an Infinite Impulse Response (IIR) system.

i) merchant_savings system – FIR System
For this LTI system, the impulse response is $h[n] = 1/2$, a constant value. This results in the output being simply a scaled version of the input, with the scaling factor of 0.5. Here, the impulse response has only one non-zero value, meaning only a finite number of input samples (just one) affect the output. Thus, this system is a Finite Impulse Response (FIR) system.