National Institute of Technology
Calicut

Department of Electronics and Communication Engineering

Digital Signal Processing LAB

Week 1 Report

Date: 24–07–2025
Instructor: Dr. Sakthivel V

Submitted by:

Mummana Jagadeesh B231113EC
Muhammed Nihal MP B230437EC

Aim

To generate and visualize basic discrete-time signals using MATLAB such as periodic signals, exponentially growing and decaying signals, and alternating sign sequences.

(a) Discrete impulse signal

(b) Discrete unit step signal

(c) Discrete unit ramp signal

Experiment 1.a: Basic Signal Generation

(a) Discrete Impulse Signal

Theory

The **discrete impulse signal**, also known as the *unit sample function*, is a fundamental building block in discrete-time signal processing. It is denoted as $\delta[n]$ and defined mathematically as:

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

This is the discrete-time counterpart of the continuous-time Dirac delta function. Unlike the continuous delta — which is a mathematical distribution — $\delta[n]$ is an actual function with a clear graphical interpretation.

Properties of the Discrete Impulse Signal:

- $\delta[n]$ is non-zero only at $n = 0$
- Acts as the identity element under convolution:

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

- Can be used to reconstruct any discrete signal:

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

Discrete-Time Domain Overview

In discrete-time systems, signals are defined only at integer values of time, i.e., $n \in Z$. These signals are often represented as sequences and visualized as stem plots. MATLAB naturally supports such signals using vectors and indexing, making it a powerful tool for signal analysis and simulation.

MATLAB Implementation

To implement the discrete impulse signal in MATLAB:

- Define a range for the time index (e.g., $n = -10 : 10$)
- Create a zero-valued vector
- Set the center element (corresponding to $n = 0$) to 1
- Use `stem` to plot the signal

MATLAB Code Listing

Code:

MATLAB Code for Discrete Impulse Signal

```
% Discrete Impulse Signal Generation
n = -10:10;              % Time index from -10 to 10
delta = (n == 0);        % Logical condition: 1 at n = 0, else 0

stem(n, delta, 'filled'); % Discrete stem plot
xlabel('n');
ylabel('\delta[n]');
title('Discrete Impulse Signal');
grid on;
```

Explanation of the Code

- `n = -10:10;` defines the discrete-time index from $n = -10$ to $n = 10$.
- `delta = (n == 0);` creates a logical array where the value is 1 only at $n = 0$.
- `stem` generates a discrete-time stem plot of the signal.
- Labels and grid enhance readability of the signal plot.

Output

The plot consists of a single spike (value = 1) located at $n = 0$, and zero values elsewhere. This spike represents the discrete-time impulse.

Output Plot:

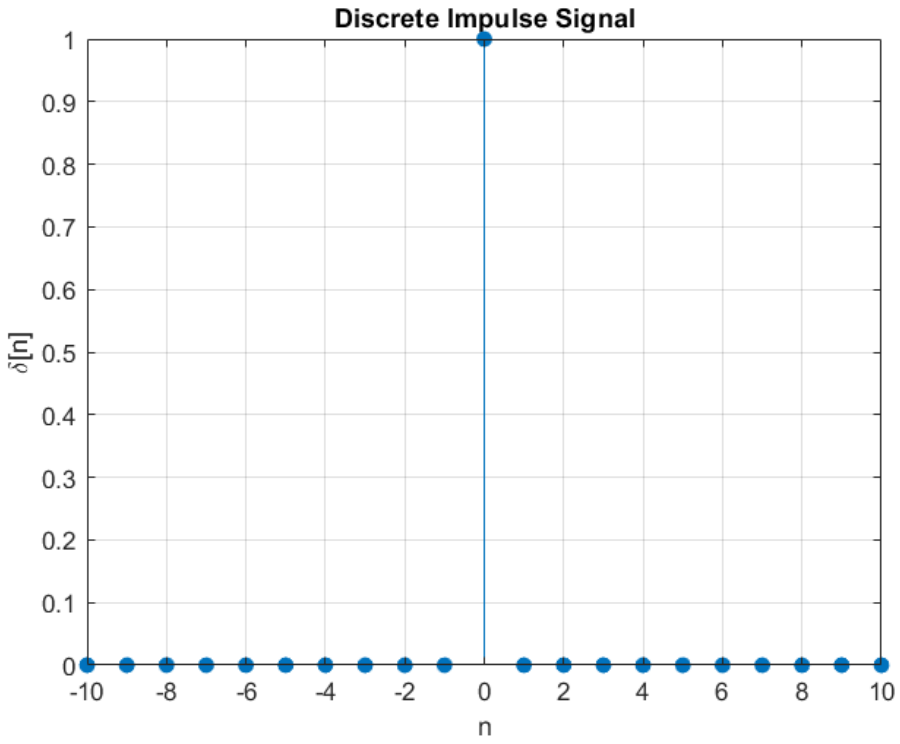

Figure 1: Discrete-time impulse signal with a spike at $n = 0$

(b.1) Discrete Unit Step Signal

Theory

The **discrete unit step signal**, denoted by $u[n]$, is a fundamental signal used to describe systems that begin operating at $n = 0$. It is defined as:

$$u[n] = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

The unit step signal is widely used in digital signal processing for modeling system responses and initial conditions. It represents a sudden transition from an "off" state to an "on" state and is used as a building block to create more complex signals.

Properties of the Discrete Unit Step Signal:

- $u[n]$ is causal: it is zero for all $n < 0$
- Cumulative sum of impulse:

$$u[n] = \sum_{k=-\infty}^{n} \delta[k]$$

- Used to control the start of signals and systems

MATLAB Implementation

To generate the discrete unit step signal:

- Define time index n (e.g., from -10 to 10)
- Assign value 1 for $n \geq 0$ using logical indexing
- Use 'stem' for discrete visualization

MATLAB Code Listing

Code:

MATLAB Code for Discrete Unit Step Signal

```
% Discrete Unit Step Signal Generation
n = -10:10;           % Time index
u = (n >= 0);         % Unit step: 1 for n >= 0, else 0

stem(n, u, 'filled');
xlabel('n');
ylabel('u[n]');
title('Discrete Unit Step Signal');
grid on;
```

Explanation of the Code

- `u = (n >= 0);` sets all values at and after $n = 0$ to 1.
- `stem` creates a discrete-time stem plot.
- The resulting plot is a sequence of ones for $n \geq 0$.

Output

This plot starts at $n = 0$ with a value of 1 and continues indefinitely (in simulation, up to $n = 10$). It visually represents a step turning "on" at $n = 0$.

Output Plot:

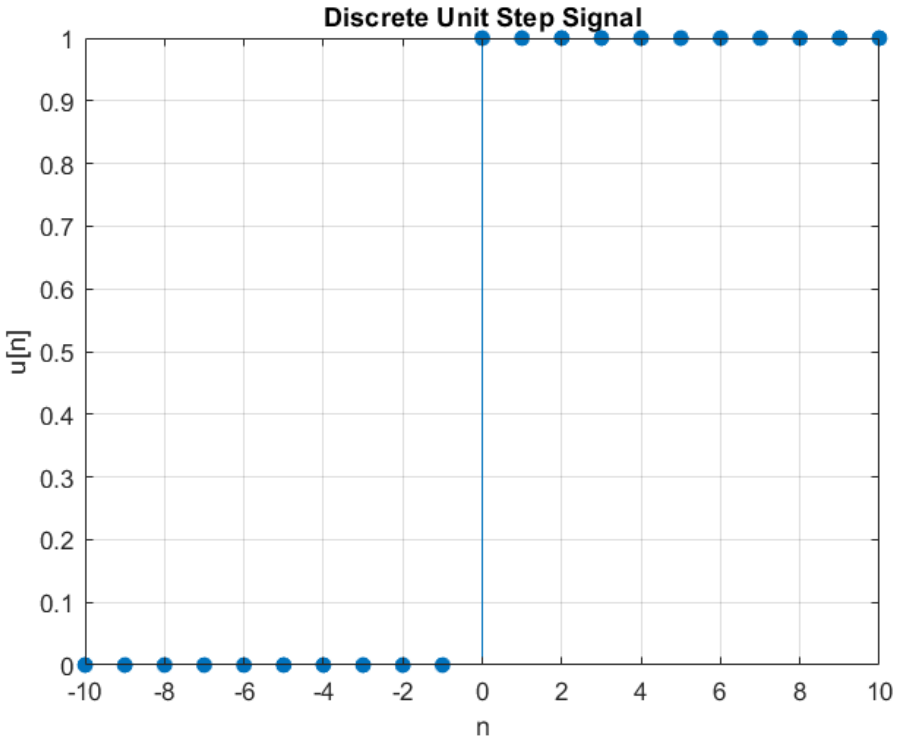

Figure 2: Discrete-time unit step signal starting from $n = 0$

(b.2) Continuous-Time Unit Step Signal

Theory

The **continuous-time unit step signal**, denoted by $u(t)$, is defined as:

$$u(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

It is the continuous analogue of the discrete unit step. It is not defined at $t = 0$ in classical terms, but is often assigned a value of 0.5 at $t = 0$ in Laplace and system theory (depending on convention).

Properties:

- Discontinuous at $t = 0$
- Often used to model switches, system inputs, and start-up behavior
- Integral of Dirac delta:

$$u(t) = \int_{-\infty}^{t} \delta(\tau)\, d\tau$$

MATLAB Implementation

To generate the continuous unit step signal:

- Define a time vector t, e.g., from -2 to 2
- Set values to 1 for $t \geq 0$
- Use 'plot' for continuous appearance

MATLAB Code Listing

Code:

MATLAB Code for Continuous-Time Unit Step Signal

```
% Continuous-Time Unit Step Signal Generation
t = -2:0.01:2;          % Time vector with small step size
u = (t >= 0);           % Unit step definition

plot(t, u, 'LineWidth', 2);
xlabel('t');
ylabel('u(t)');
title('Continuous-Time Unit Step Signal');
grid on;
axis([-2 2 -0.2 1.2]);
```

Explanation of the Code

- `t = -2:0.01:2;` defines a dense time vector for a smooth plot.
- `u = (t >= 0);` implements the condition for the unit step.
- The 'plot' command gives a piecewise jump at $t = 0$.

Output

The output is a step function rising from 0 to 1 at $t = 0$, which continues as a flat line for $t > 0$.

Output Plot:

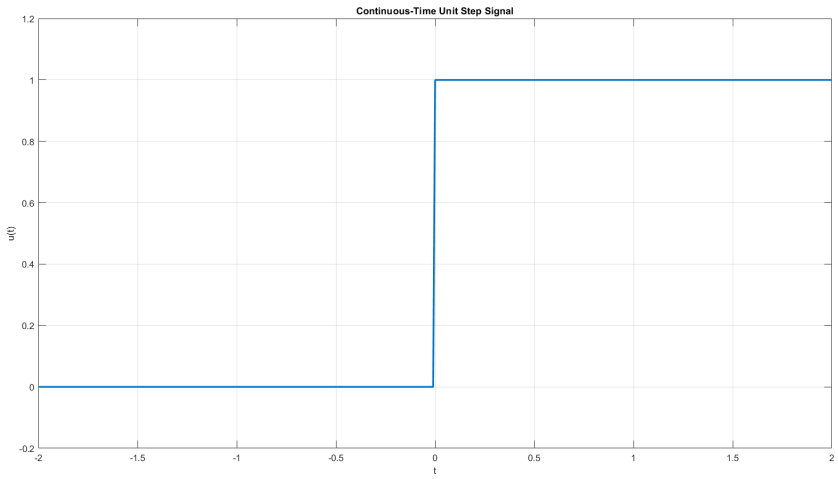

Figure 3: Continuous-time unit step signal with a jump at $t = 0$

(c.1) Discrete Unit Ramp Signal

Theory

The **discrete unit ramp signal**, denoted by $r[n]$, is defined as a linearly increasing sequence starting at $n = 0$. It is given by:

$$r[n] = \begin{cases} n, & n \geq 0 \\ 0, & n < 0 \end{cases}$$

This signal increases linearly with time in discrete steps and is useful for analyzing systems with growing responses.

Properties of Discrete Ramp Signal:

- $r[n] = n \cdot u[n]$ (product of ramp and step)
- First difference of ramp gives step:

$$r[n] - r[n-1] = u[n]$$

- Often used to test system responses to increasing inputs

MATLAB Implementation

To generate the discrete ramp signal:

- Define a time vector n
- Use `ramp = n .* (n >= 0)` to keep only $n \geq 0$
- Use 'stem' to plot it

MATLAB Code Listing

Code:

MATLAB Code for Discrete Ramp Signal

```
% Discrete Unit Ramp Signal
n = -10:10;                    % Discrete time index
ramp = n .* (n >= 0);          % Ramp defined only for n >= 0

stem(n, ramp, 'filled');
xlabel('n');
ylabel('r[n]');
title('Discrete Unit Ramp Signal');
grid on;
```

Explanation of the Code

- `ramp = n .* (n >= 0);` ensures zero for $n < 0$, and $r[n] = n$ for $n \geq 0$.
- `stem` provides a discrete-time visualization.
- The output shows increasing values for non-negative indices.

Output

The output is a stair-step sequence starting at $n = 0$ with value 0, increasing linearly as n increases.

Output Plot:

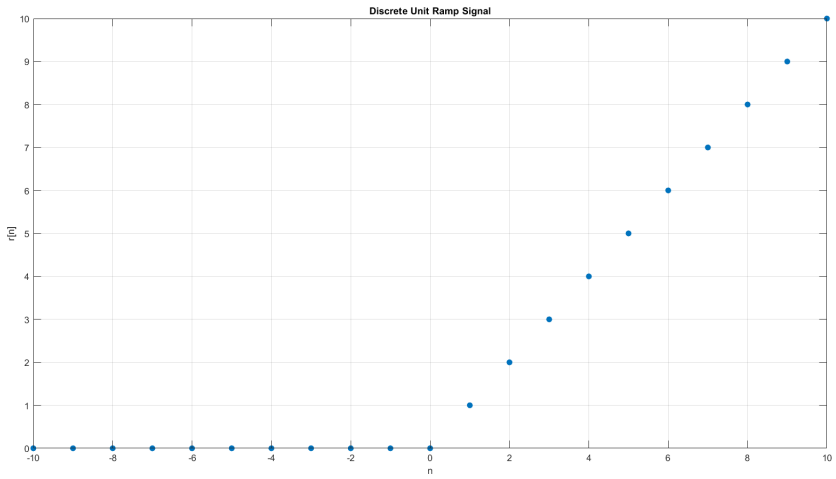

Figure 4: Discrete-time unit ramp signal $r[n] = n$ for $n \geq 0$

(c.2) Continuous-Time Unit Ramp Signal

Theory

The **continuous-time unit ramp signal**, denoted as $r(t)$, is defined as:

$$r(t) = \begin{cases} t, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

It is a linearly increasing function for $t \geq 0$ and zero before. It is used to analyze system responses to inputs that grow over time.

Properties:

- $r(t) = t \cdot u(t)$
- Derivative of ramp gives step:
$$\frac{d}{dt} r(t) = u(t)$$
- Integral of unit step:
$$r(t) = \int_{-\infty}^{t} u(\tau)\, d\tau$$

MATLAB Implementation

To implement the continuous ramp:

- Define time vector t (e.g., from -2 to 2)
- Set $r(t) = t$ for $t \geq 0$, zero otherwise
- Use 'plot' for smooth graph

MATLAB Code Listing

Code:

MATLAB Code for Continuous Ramp Signal

```
% Continuous-Time Unit Ramp Signal
t = -2:0.01:2;                % Time vector
ramp = t .* (t >= 0);         % Ramp: t for t >= 0, 0 otherwise

plot(t, ramp, 'LineWidth', 2);
xlabel('t');
ylabel('r(t)');
title('Continuous-Time Unit Ramp Signal');
grid on;
axis([-2 2 -0.5 2.5]);
```

Explanation of the Code

- `ramp = t .* (t >= 0);` ensures $r(t) = t$ only for $t \geq 0$.
- A smooth linear rise from zero is shown in the output.
- The 'plot' command gives a continuous curve.

Output

The signal starts from zero at $t = 0$ and increases linearly. It remains zero for $t < 0$.

Output Plot:

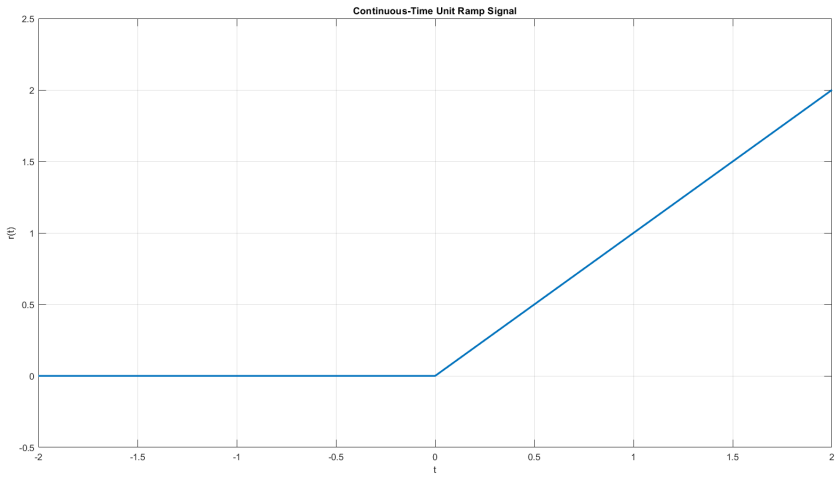

Figure 5: Continuous-time unit ramp signal $r(t) = t$ for $t \geq 0$

(1.b) Impulse Train Signal

Theory

An **impulse train** is a periodic sequence of discrete impulses spaced at regular intervals. It is mathematically expressed as:

$$x[n] = \sum_{k=-\infty}^{\infty} \delta[n - kN]$$

Where: - $\delta[n]$ is the unit impulse, - N is the spacing between impulses, - The signal is periodic with period N.

Applications:

- Used in sampling theory to model ideal sampling
- Acts as a frequency comb in the frequency domain

MATLAB Implementation

- Choose n range (e.g., -20 to 20)
- Define spacing N (e.g., $N = 4$)
- Use modulus operation to generate impulses at $n = kN$

MATLAB Code Listing

MATLAB Code for Impulse Train

```
% Impulse Train Signal
n = -20:20;
N = 4;                          % Period of impulse train
x = mod(n, N) == 0;             % Impulses at multiples of N

stem(n, x, 'filled');
xlabel('n');
ylabel('x[n]');
title('Impulse Train with Period N = 4');
grid on;
```

Explanation

- `mod(n, N) == 0` returns 1 at every n that is a multiple of N
- The plot shows impulses every N steps

Output Plot

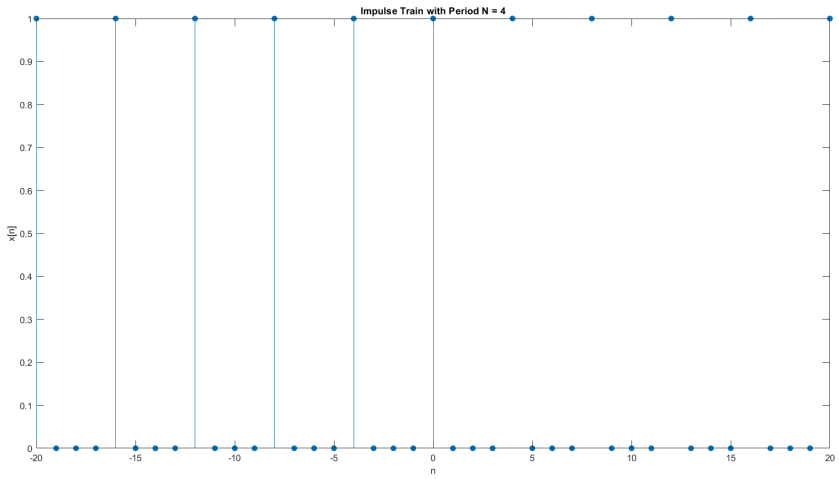

Figure 6: Impulse train with impulses every $N = 4$ samples

(1.c) Pulse Train Signal

Theory

A **pulse train** is a periodic signal composed of rectangular pulses. Each pulse is typically of fixed width W and separated by period T. It is defined as:

$$x[n] = \begin{cases} 1, & 0 \leq n \bmod T < W \\ 0, & \text{otherwise} \end{cases}$$

Applications:

- Used in digital timing, clocks, control signals
- Approximates square waves and sampling gates

MATLAB Implementation

- Define n, period T, and pulse width W
- Use modulus to generate periodic rectangular pulses

MATLAB Code Listing

MATLAB Code for Pulse Train

```
% Pulse Train Signal
n = 0:50;
T = 10;                          % Period
W = 4;                           % Pulse width
x = mod(n, T) < W;               % Logic for pulse window

stem(n, x, 'filled');
xlabel('n');
ylabel('x[n]');
title('Pulse Train (T=10, W=4)');
grid on;
```

Explanation

- `mod(n, T) < W` creates 1s for the pulse width within each period
- The result is a repeating rectangular wave

Output Plot

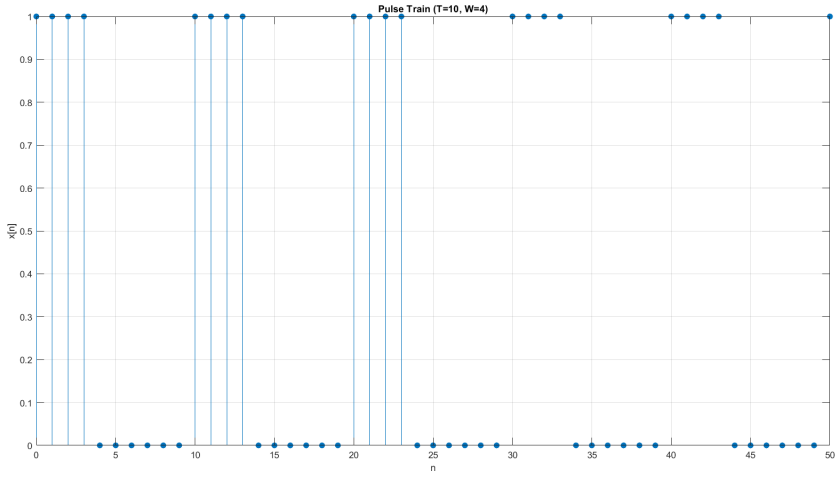

Figure 7: Pulse train with period $T = 10$ and width $W = 4$

(1.d) Sine and Cosine Signals (Continuous-Time)

Theory

Sine and cosine are fundamental continuous-time periodic signals, widely used in communication, modulation, and Fourier analysis. The general forms are:

$$x(t) = \sin(2\pi ft), \quad y(t) = \cos(2\pi ft)$$

Where:

- t is the continuous time variable,
- f is the frequency in Hz (cycles per second),
- The signals are periodic with period $T = \frac{1}{f}$.

Properties:

- Amplitude is bounded: $-1 \leq x(t), y(t) \leq 1$
- Periodic: $\sin(2\pi ft) = \sin(2\pi f(t + T))$
- Orthogonality: $\int_0^T \sin(2\pi ft)\cos(2\pi ft)\, dt = 0$

MATLAB Implementation

- Define a continuous time vector using small steps (e.g., `t = 0:0.01:2`)
- Choose a frequency (e.g., $f = 2$ Hz)
- Use `sin(2*pi*f*t)` and `cos(2*pi*f*t)` for the signals
- Use `plot` for smooth curves

MATLAB Code Listing

MATLAB Code for Continuous-Time Sine and Cosine

```
% Continuous-Time Sine and Cosine Signals
t = 0:0.01:2;               % Time vector (0 to 2 seconds)
f = 2;                      % Frequency in Hz

x = sin(2*pi*f*t);          % Sine wave
y = cos(2*pi*f*t);          % Cosine wave

subplot(2,1,1);
plot(t, x, 'b', 'LineWidth', 1.5);
title('Continuous-Time Sine Wave');
xlabel('Time (t)'); ylabel('sin(2\pi f t)'); grid on;

subplot(2,1,2);
plot(t, y, 'r', 'LineWidth', 1.5);
title('Continuous-Time Cosine Wave');
xlabel('Time (t)'); ylabel('cos(2\pi f t)'); grid on;
```

Explanation

- `t = 0:0.01:2;` gives a dense time sampling for smooth curves.
- `f = 2;` sets the signal frequency to 2 Hz (period = 0.5 s).
- The sine and cosine signals are smooth and continuous.
- `subplot` is used to display both signals in one figure.

Output Plot

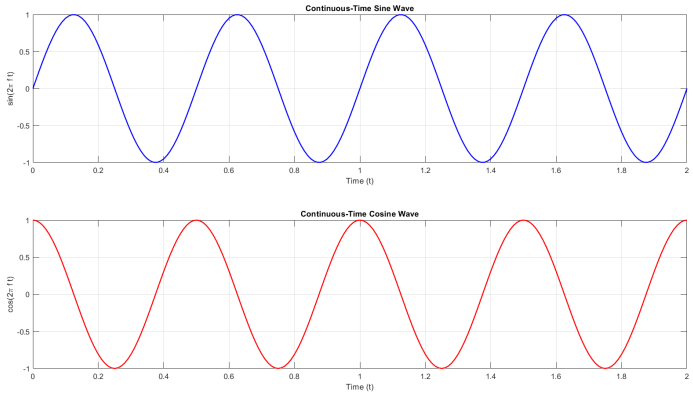

Figure 8: Continuous-time sine and cosine waves ($f = 2$ Hz)

(1.e) Miscellaneous Discrete-Time Signals

Mathematical Expressions

1. Periodic Sequence: $x_1[n] = \{1, 2, 3, 4, 1, 2, 3, 4, \dots\}$
2. Exponential Decay: $x_2[n] = 20 \cdot (0.9)^n \cdot u[n]$
3. Exponential Growth: $x_3[n] = 0.2 \cdot (1.2)^n \cdot u[n]$
4. Alternating Decay: $x_4[n] = (-0.8)^n \cdot u[n]$
5. Negative Exponential Decay: $x_5[n] = -4 \cdot (0.8)^n \cdot u[n]$

MATLAB Code Listing

MATLAB Code for Miscellaneous Discrete-Time Signals

```
% Define the range of n
n = 0:20;

% Unit step function
u = ones(size(n));  % u[n] = 1 for n >= 0

% 1. Periodic signal: x[n] = ...,1,2,3,4,...
x1 = repmat([1 2 3 4], 1, ceil(length(n)/4));
x1 = x1(1:length(n));

% 2. Exponential decay: x[n] = 20*(0.9)^n * u[n]
x2 = 20 * (0.9).^n .* u;

% 3. Exponential growth: x[n] = 0.2*(1.2)^n * u[n]
x3 = 0.2 * (1.2).^n .* u;

% 4. Alternating decay: x[n] = (-0.8)^n * u[n]
x4 = (-0.8).^n .* u;

% 5. Negative exponential decay: x[n] = -4*(0.8)^n * u[n]
x5 = -4 * (0.8).^n .* u;

% Plotting
figure;

subplot(3,2,1);
stem(n, x1, 'filled');
title('x[n] = periodic \{1,2,3,4\}');
xlabel('n'); ylabel('x[n]');
grid on;
```

```
subplot(3,2,2);
stem(n, x2, 'filled');
title('x[n] = 20(0.9)^n u[n]');
xlabel('n'); ylabel('x[n]');
grid on;

subplot(3,2,3);
stem(n, x3, 'filled');
title('x[n] = 0.2(1.2)^n u[n]');
xlabel('n'); ylabel('x[n]');
grid on;

subplot(3,2,4);
stem(n, x4, 'filled');
title('x[n] = (-0.8)^n u[n]');
xlabel('n'); ylabel('x[n]');
grid on;

subplot(3,2,5);
stem(n, x5, 'filled');
title('x[n] = -4(0.8)^n u[n]');
xlabel('n'); ylabel('x[n]');
grid on;
```

Output Plot

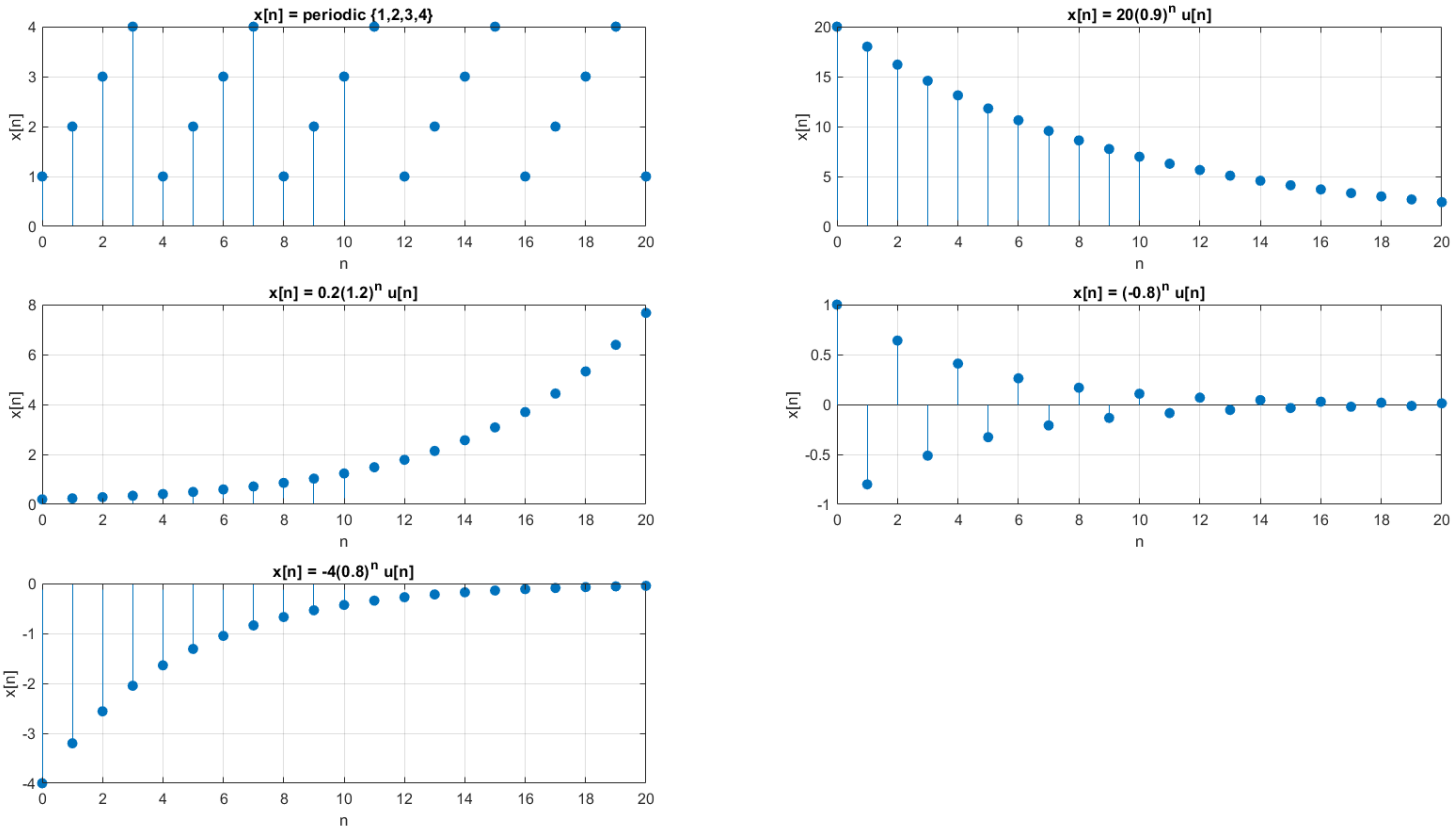

Figure 9: Various discrete-time signals plotted using MATLAB