1.2 Output

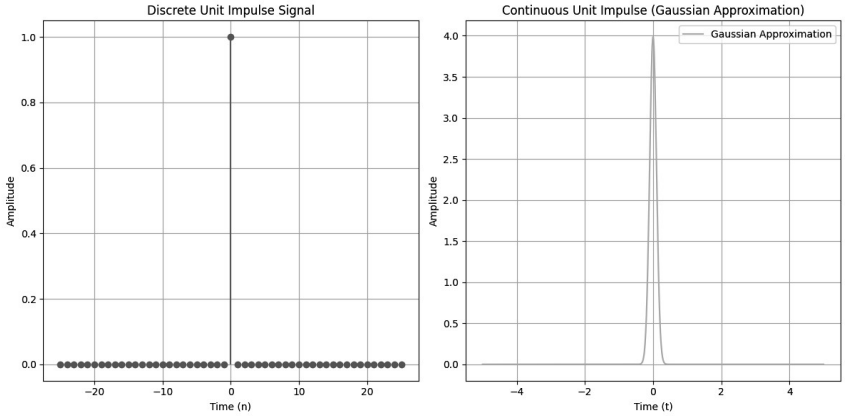

Figure 1: Distribution-plot

1.3 Discussion

- **Range for n:** The range for n is given by the expression:

  $$n = \texttt{np.arange(-25, 26)}$$

  This creates a range of n from -25 to 25 (a total of 51 samples), with $n = 0$ being the middle of the range.

- **Impulse at $n = 0$:** The impulse signal is initialized as all zeros using:

  $$\texttt{impulse = np.zeros\_like(n)}$$

  This creates an array of zeros, and then the value at $n = 0$ is set to 1 using:

  $$\texttt{impulse[n == 0] = 1}$$

- **Plot:** The signal is visualized using a stem plot:

  $$\texttt{plt.stem(n, impulse)}$$

  This generates a stem plot where each "stem" represents a value of the impulse signal.

- **Unit Impulse:** The signal has a value of 1 at $n = 0$ and 0 everywhere else. It is a discrete-time signal with a non-zero value only at a single point (in this case, $n = 0$).

- **Centered around $n = 0$:** The impulse occurs exactly at $n = 0$. The range from -25 to 25 is chosen to provide a broader context for the visualization of the signal.

2 Generate a Unit Step Sequence $[u(n) - u(n - N)]$

2.1 Code: Python

```python
import matplotlib.pyplot as plt
import numpy as np

```

```
# Function to create a unit step sequence
def do_unit_step_sequence(range_n, shift_N):
    step_u_n = np.where(range_n >= 0, 1, 0)  # Step sequence u(n)
    step_u_n_minus_N = np.where(range_n >= shift_N, 1, 0)  # Step sequence u(n-N)
    return step_u_n - step_u_n_minus_N  # Difference: [u(n) - u(n-N)]

# Parameters for the sequence
shift_N = 5  # Shift length
range_n = np.arange(-5, 10)  # Values of n

# Generate the unit step signal
step_sequence = do_unit_step_sequence(range_n, shift_N)

# Plotting the generated unit step sequence
plt.figure(figsize=(10, 5))
plt.stem(range_n, step_sequence, basefmt=" ")
plt.title("Unit Step Sequence: [u(n) - u(n - N)]")
plt.xlabel("Index (n)")
plt.ylabel("Amplitude")
plt.grid(True)
plt.show()
```

Listing 2: Code

2.2 Output

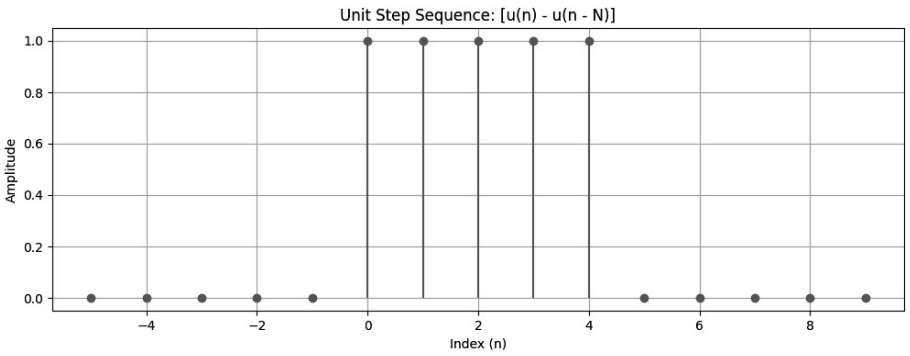

Figure 2: Distribution-plot

2.3 Discussion

- **Function Type:**It is a discrete impulse function in terms of Unit step function
- **Movement:** As per signal, it is moving in positive direction.
- **Plotting:** If we check the plot, the amplitude of u(n-N) is 1 for $n \geq N$, otherwise 0.
- **Effect of Sigma:** The value of σ affects how sharp or broad the peak is. A smaller σ results in a narrower peak, while a larger σ spreads it out.
- **Conditions:**
  - u(n) is a Unit step function if u(n) = 1 for n=0, otherwise u(n)=0
  - $u(n - N) = 1$ for $n \geq N$, otherwise 0

3 Generate a ramp sequence.

3.1 Code: Python

```
import numpy as np
import matplotlib.pyplot as plt

# discrete ramp sequence
def do_discrete_ramp_sequence(start, stop, step):
    n = np.arange(start, stop + 1, step)  # Discrete range for n
    ramp = n
    return n, ramp

# continuous ramp signal
def do_continuous_ramp_sequence(t_start, t_stop, t_step):
    t = np.arange(t_start, t_stop, t_step)  # Continuous time range
    r_continuous = t
    return t, r_continuous

# Parameters for the ramp sequences
start = 0
stop = 10
step = 2

# discrete ramp sequence
n_values, ramp_discrete = do_discrete_ramp_sequence(start, stop, step)

# continuous ramp sequence with smaller steps
t_values, ramp_continuous = do_continuous_ramp_sequence(0, stop+1, 0.1)

# Create side-by-side subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 3))

# Plot discrete ramp sequence
axes[0].stem(n_values, ramp_discrete, basefmt=" ", linefmt="red", markerfmt="o",
             label="Discrete Ramp Sequence")
axes[0].set_title("Discrete Ramp Sequence")
axes[0].set_xlabel("Index (n)")
axes[0].set_ylabel("Amplitude")
axes[0].grid(True)
axes[0].legend()

# Plot continuous ramp sequence
axes[1].plot(t_values, ramp_continuous, color='blue',
             label="Continuous Ramp Sequence", linewidth=2)
axes[1].set_title("Continuous Ramp Sequence")
axes[1].set_xlabel("Time (t)")
axes[1].set_ylabel("Amplitude")
axes[1].grid(True)
axes[1].legend()

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```

Listing 3: Code

3.2 Output

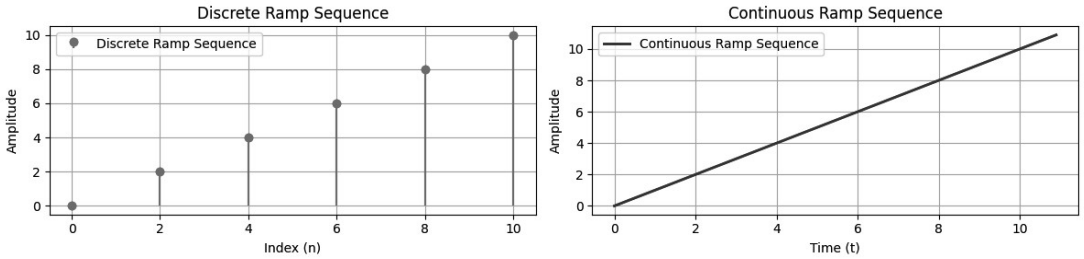

Figure 3: Distribution-plot

3.3 Discussion

Continuous Ramp Signal:

- The continuous ramp signal is given by $r(t) = t$ for $t \geq 0$, and $r(t) = 0$ for $t < 0$.
- When $t > 0$, the signal's amplitude increases linearly with time, meaning the value of the signal grows as t increases.
- For $t < 0$, the amplitude stays at 0, and the signal is effectively inactive during this time.

4 Generate an exponential sequence.

4.1 Code: Python

```python
import numpy as np
import matplotlib.pyplot as plt

# discrete exponential sequence
def do_exponential_sequence_discrete(base, length):
    return [base ** n for n in range(length)]

# continuous exponential sequence
def do_exponential_sequence_continuous(base, t_start, t_stop, t_step):
    t = np.arange(t_start, t_stop, t_step)  # Continuous time range
    exp_continuous = base ** t  # Exponential function
    return t, exp_continuous

# Parameters for the exponential sequences
exp_base = 2
length = 5
t_start = 0
t_stop = 5
t_step = 0.1

# discrete exponential sequence
exp_sequence_discrete = do_exponential_sequence_discrete(exp_base, length)

# continuous exponential sequence
t_values_continuous, exp_sequence_continuous = (
    do_exponential_sequence_continuous(exp_base, t_start, t_stop, t_step))

# Create side-by-side subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 3))

```

```
# Plot the discrete exponential sequence
axes[0].stem(range(length), exp_sequence_discrete, basefmt=" ", linefmt="red",
             markerfmt="o", label="Discrete Exponential")
axes[0].set_title("Discrete Exponential Sequence")
axes[0].set_xlabel("Discrete Time")
axes[0].set_ylabel("Amplitude")
axes[0].grid(True)
axes[0].legend()

# Plot the continuous exponential sequence
axes[1].plot(t_values_continuous, exp_sequence_continuous, color='blue',
             label="Continuous Exponential", linewidth=2)
axes[1].set_title("Continuous Exponential Sequence")
axes[1].set_xlabel("Time (t)")
axes[1].set_ylabel("Amplitude")
axes[1].grid(True)
axes[1].legend()

# Adjust layout for better spacing
plt.tight_layout()
plt.show()
```

Listing 4: Code

4.2 Output

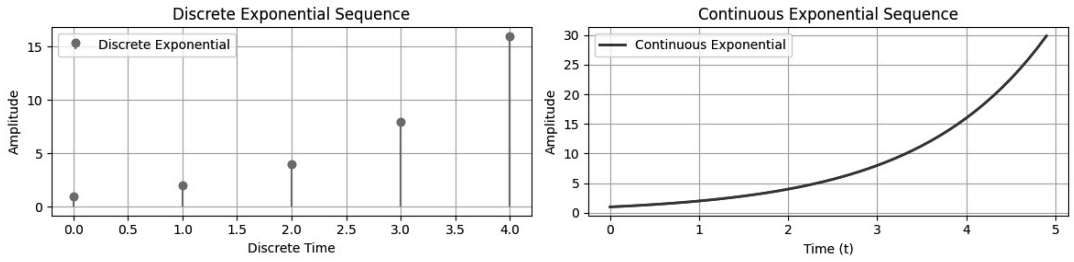

Figure 4: Distribution-plot

4.3 Discussion

Continuous Ramp Signal:

- The continuous ramp signal is given by $r(t) = t$ for $t \geq 0$, and $r(t) = 0$ for $t < 0$.
- When $t > 0$, the signal's amplitude increases linearly with time, meaning the value of the signal grows as t increases.
- For $t < 0$, the amplitude stays at 0, and the signal is effectively inactive during this time.

5 Generate a sine sequence.

5.1 Code: Python

```
import matplotlib.pyplot as plt
import numpy as np

# continuous sine sequence
def do_sine_sequence_continuous(length, amplitude=1, frequency=1, phase=0):
    x = np.linspace(0, 2 * np.pi, length)  # Generate evenly spaced points
```

```
    y = amplitude * np.sin(frequency * x + phase)
    return x, y

# discrete sine sequence
def do_sine_sequence_discrete(length, amplitude=1, frequency=1, phase=0):
    n = np.arange(length)  # Discrete time samples
    y = amplitude * np.sin(frequency * n + phase)
    return n, y

# Parameters for the sine sequences
length = 10
amplitude = 1
frequency = 1
phase = 0

# continuous sine sequence
x_cont, sine_cont = do_sine_sequence_continuous(length, amplitude, frequency, phase)

# discrete sine sequence
n_discrete, sine_discrete = do_sine_sequence_discrete(length, amplitude, frequency, phase)

plt.figure(figsize=(12, 3))

# Continuous sine wave plot (left)
plt.subplot(1, 2, 1)
plt.plot(x_cont, sine_cont, label='Continuous Sine Sequence', color='blue')
plt.title("Continuous Sine Wave")
plt.xlabel("Time (t)")
plt.ylabel("Amplitude")
plt.grid(True)
plt.legend()

# Discrete sine wave plot (right)
plt.subplot(1, 2, 2)
plt.stem(n_discrete, sine_discrete, label='Discrete Sine Sequence',
         basefmt=" ", linefmt='r-', markerfmt='ro')
plt.title("Discrete Sine Wave")
plt.xlabel("Sample Index (n)")
plt.ylabel("Amplitude")
plt.grid(True)
plt.legend()

# Show the plots
plt.tight_layout()
plt.show()
```

Listing 5: Code

5.2 Output

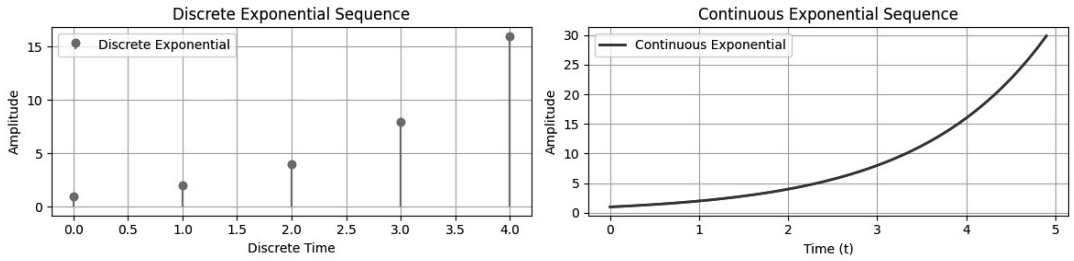

Figure 5: Distribution-plot

6 Generate a cosine sequence.

6.1 Code: Python

```
import matplotlib.pyplot as plt
import numpy as np

# continuous cosine sequence
def do_cosine_sequence_continuous(length, amplitude=1, frequency=1, phase=0):
    x = np.linspace(0, 2 * np.pi, length)  # Generate evenly spaced points
    y = amplitude * np.cos(frequency * x + phase)
    return x, y

# discrete cosine sequence
def do_cosine_sequence_discrete(length, amplitude=1, frequency=1, phase=0):
    n = np.arange(length)  # Discrete time samples
    y = amplitude * np.cos(frequency * n + phase)
    return n, y

# Parameters
length = 10
amplitude = 1
frequency = 1
phase = 0

# continuous cosine sequence
x_cont, cosine_cont = do_cosine_sequence_continuous(length, amplitude, frequency, phase)

# discrete cosine sequence
n_discrete, cosine_discrete = do_cosine_sequence_discrete(length, amplitude, frequency, phase)

plt.figure(figsize=(12, 3))

# Continuous cosine wave plot (left)
plt.subplot(1, 2, 1)
plt.plot(x_cont, cosine_cont, label='Continuous Cosine Sequence', color='blue')
plt.title("Continuous Cosine Wave")
plt.xlabel("Time (t)")
plt.ylabel("Amplitude")
plt.grid(True)
plt.legend()

# Discrete cosine wave plot (right)
plt.subplot(1, 2, 2)
plt.stem(n_discrete, cosine_discrete, label='Discrete Cosine Sequence', basefmt=" ",
         linefmt='r-', markerfmt='ro')
plt.title("Discrete Cosine Wave")
plt.xlabel("Sample Index (n)")
plt.ylabel("Amplitude")
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```

Listing 6: Code

6.1.1 Output

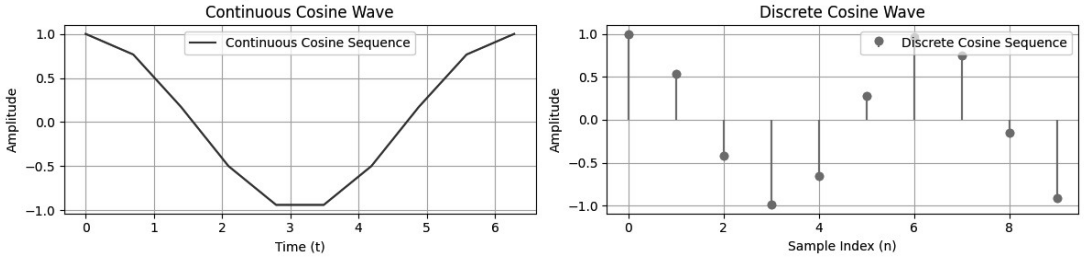

Figure 6: Distribution-plot

7 Generate a sine sequence with three frequencies, three amplitudes, and three phases

7.1 Code: Python

```python
import matplotlib.pyplot as plt
import numpy as np

# combined sine sequence
def do_combined_sine_sequence(length, f1, f2, f3, A1, A2, A3, phi_1, phi_2, phi_3):
    x = np.linspace(0, 2 * np.pi, length)  # do evenly spaced points
    y1 = A1 * np.sin(f1 * x + phi_1)
    y2 = A2 * np.sin(f2 * x + phi_2)
    y3 = A3 * np.sin(f3 * x + phi_3)

    return x, y1, y2, y3

# Parameters
length = 500
f1, f2, f3 = 1, 2, 3
A1, A2, A3 = 1, 0.5, 0.8
phi_1, phi_2, phi_3 = 0, np.pi/4, np.pi/2

# combined sine sequence
x, y1, y2, y3 = do_combined_sine_sequence(length, f1, f2, f3, A1, A2, A3, phi_1, phi_2, phi_3)

# Plot
plt.figure(figsize=(10, 6))
plt.plot(x, y1, label='Sine Wave 1 (f1, A1, phi_1)', linestyle='dotted')
plt.plot(x, y2, label='Sine Wave 2 (f2, A2, phi_2)', linestyle='dashed')
plt.plot(x, y3, label='Sine Wave 3 (f3, A3, phi_3)', linestyle='dotted')


plt.xlabel("X")
plt.ylabel("Amplitude")
plt.grid(True)
plt.legend()
plt.show()
```

Listing 7: Code

7.2 Output

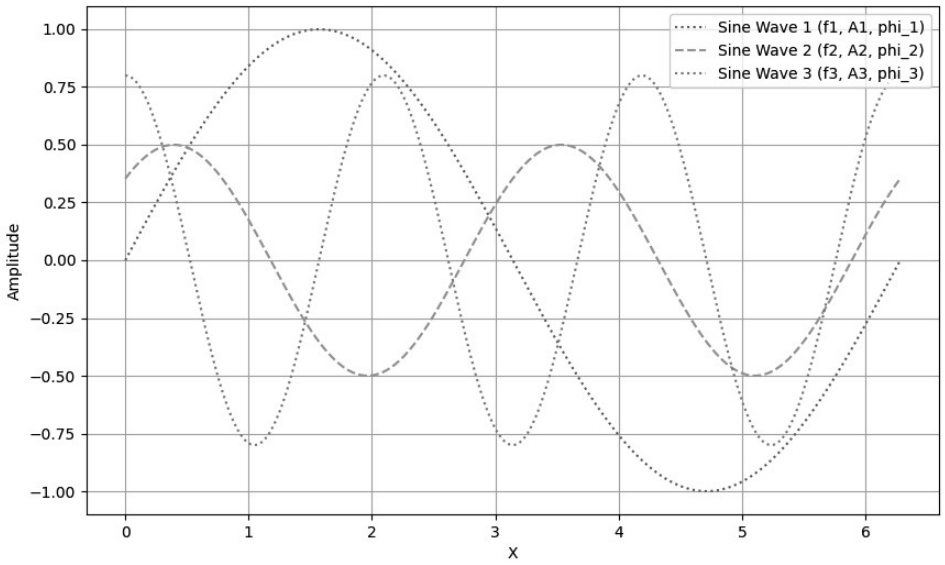

Figure 7: Distribution-plot