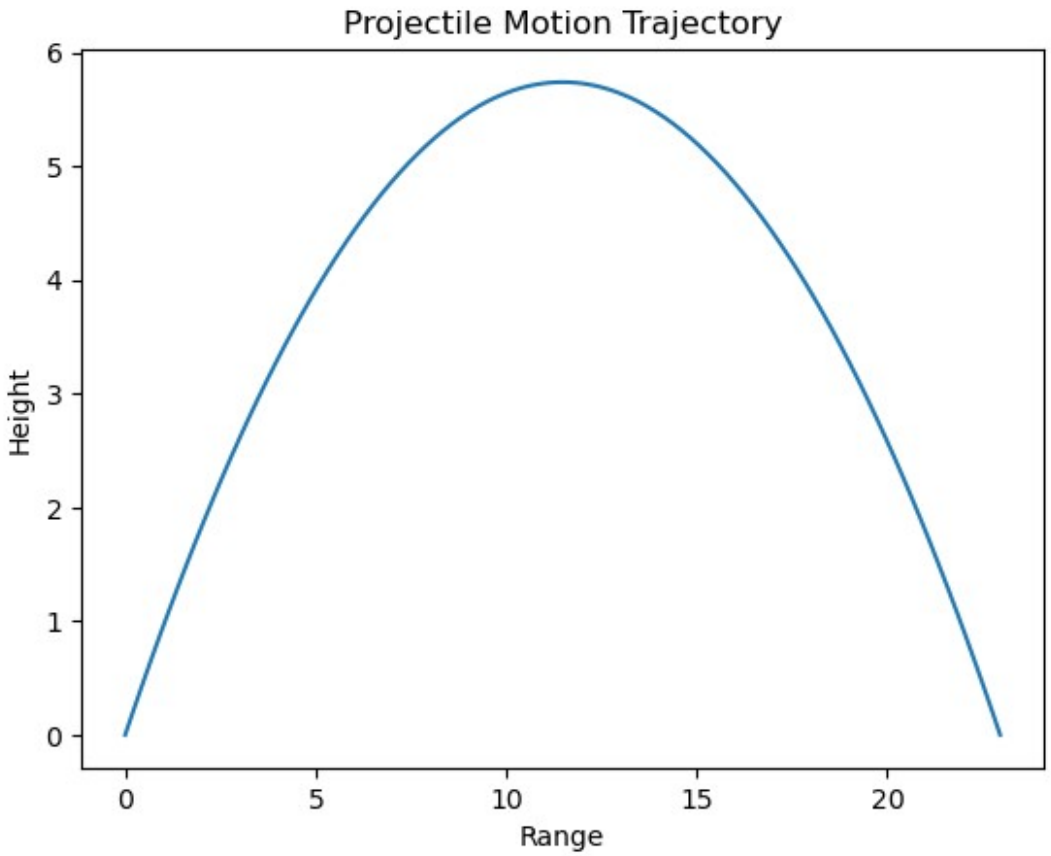
PROGRAM 7: Generate data for coordinates of a projectile and plot the trajectory. Determine the range, maximum height and time of flight for a projectile motion.

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

def projectile_motion(v0, theta, g=9.8):
    # Time of flight
    t_flight = (2 * v0 * np.sin(theta)) / g

    # Range
    range_ = (v0**2) * np.sin(2 * theta) / g

    # Maximum height
    max_height = (v0**2) * (np.sin(theta)**2) / (2 * g)

    # Time and displacement arrays
    t = np.linspace(0, t_flight, 100)
    x = v0 * np.cos(theta) * t
    y = v0 * np.sin(theta) * t - 0.5 * g * t**2

    return t, x, y, range_, max_height, t_flight

# Example usage
v0 = 15  # Initial velocity
theta = np.pi / 4  # Launch angle in radians

# Call the function
t, x, y, range_, max_height, t_flight = projectile_motion(v0, theta)

# Plotting the trajectory
plt.plot(x, y)
plt.xlabel('Range')
plt.ylabel('Height')
plt.title('Projectile Motion Trajectory')
plt.show()

# Printing the results.
print('Range: ', range_)
print('Maximum Height: ', max_height)
print('Time of Flight: ', t_flight)
plt.show()
```

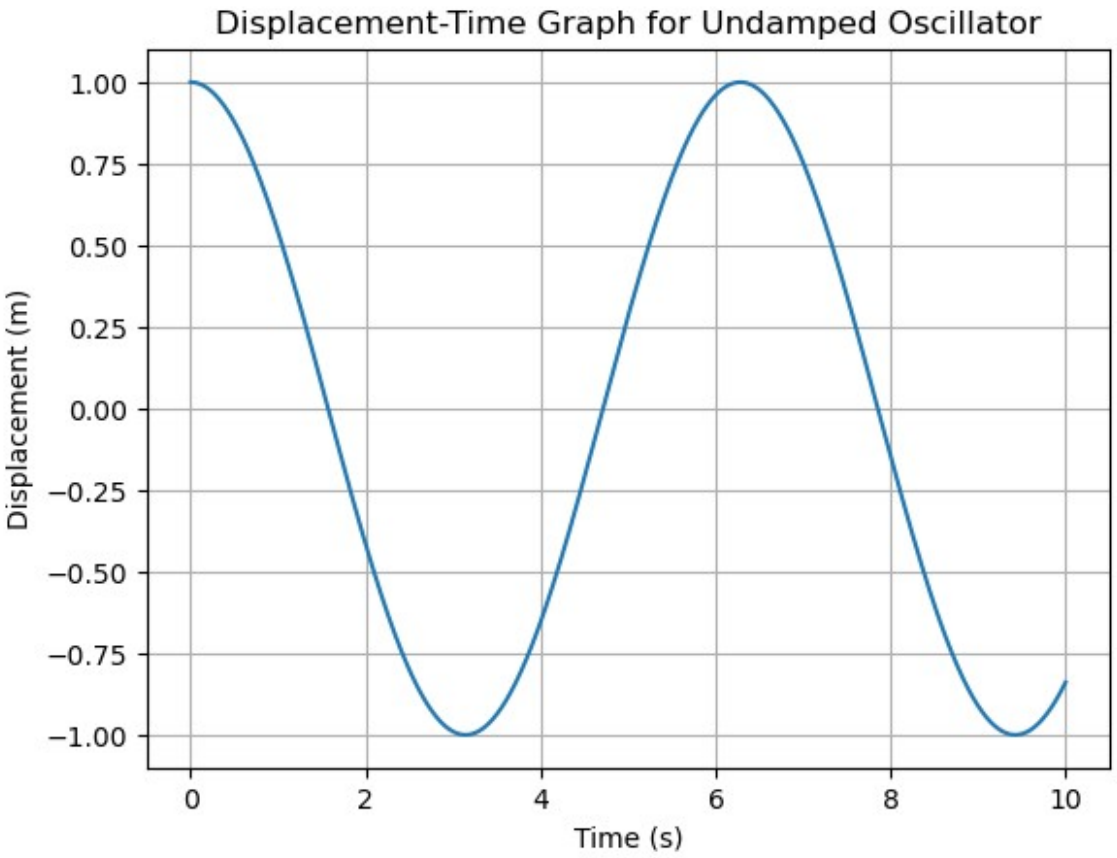


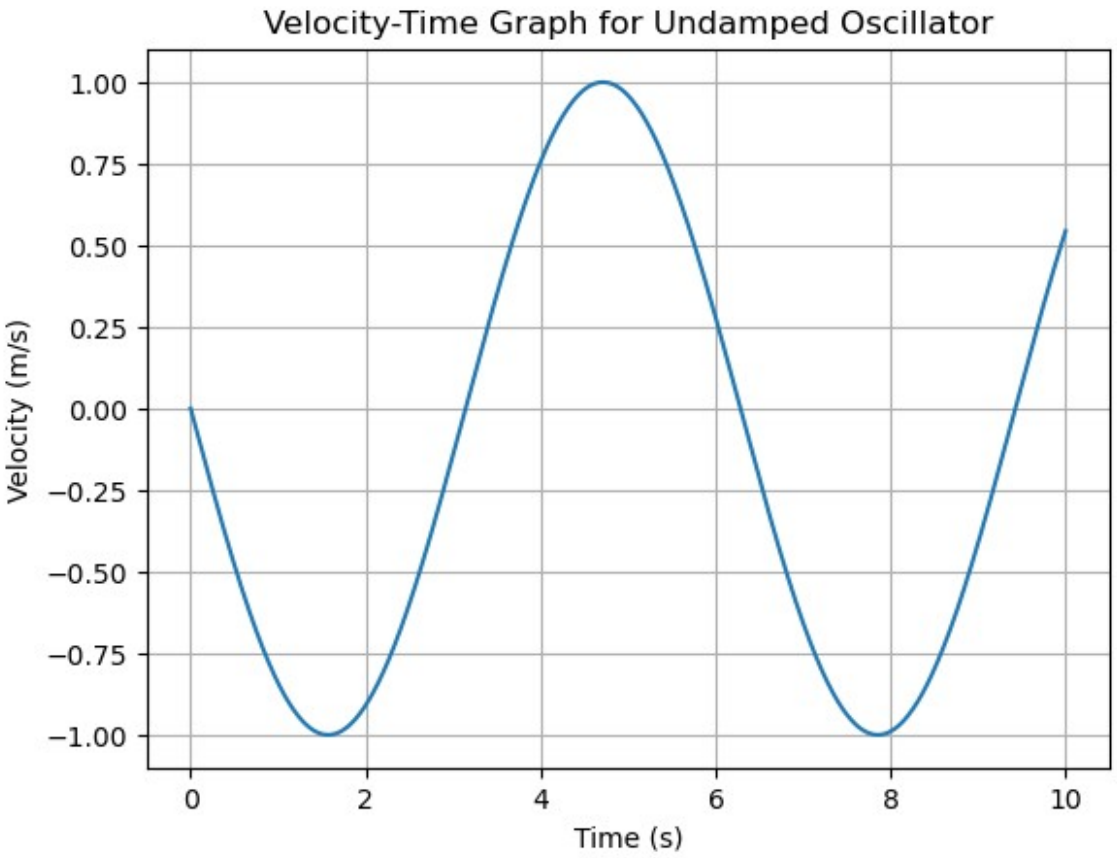

```
Range:  22.959183673469386
Maximum Height:  5.739795918367348
Time of Flight:  2.164612595469023
```

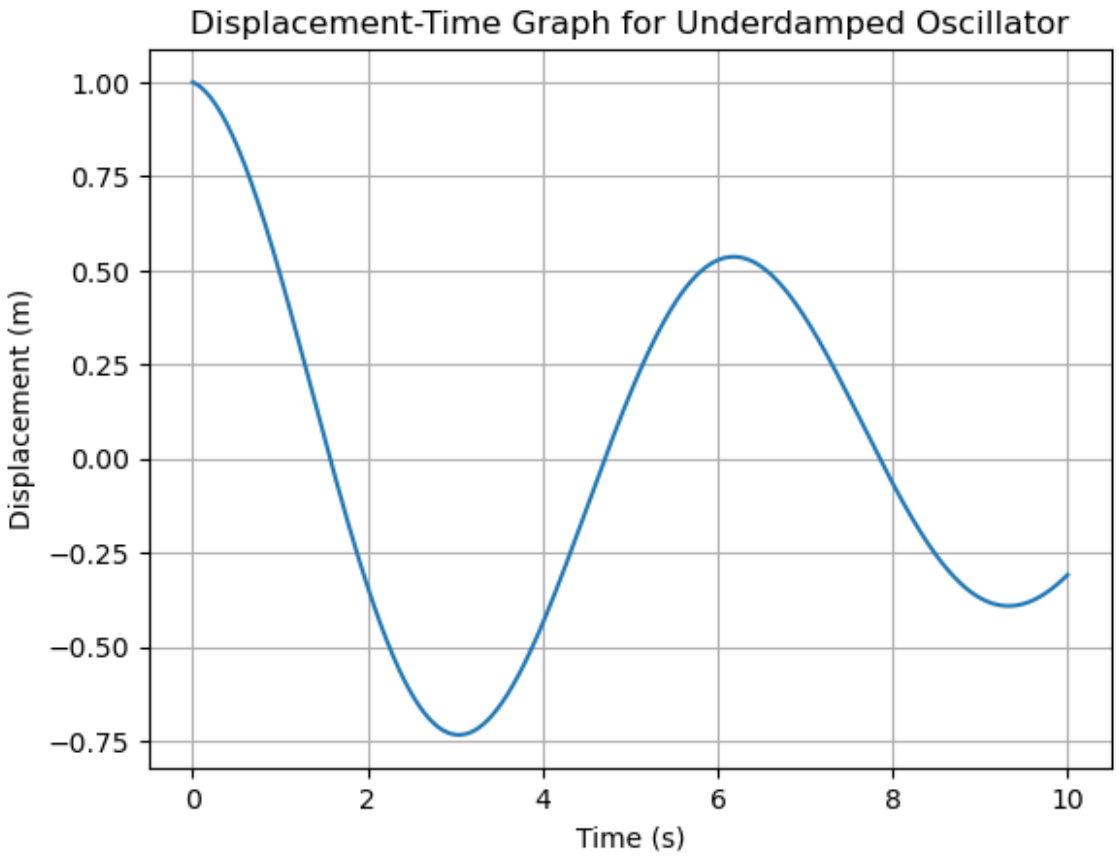
PROGRAM 8: To plot the displacement-time and velocity-time graph for an un-damped , underdamped , critically damped and over damped oscillator using matplotlib using given formulae.

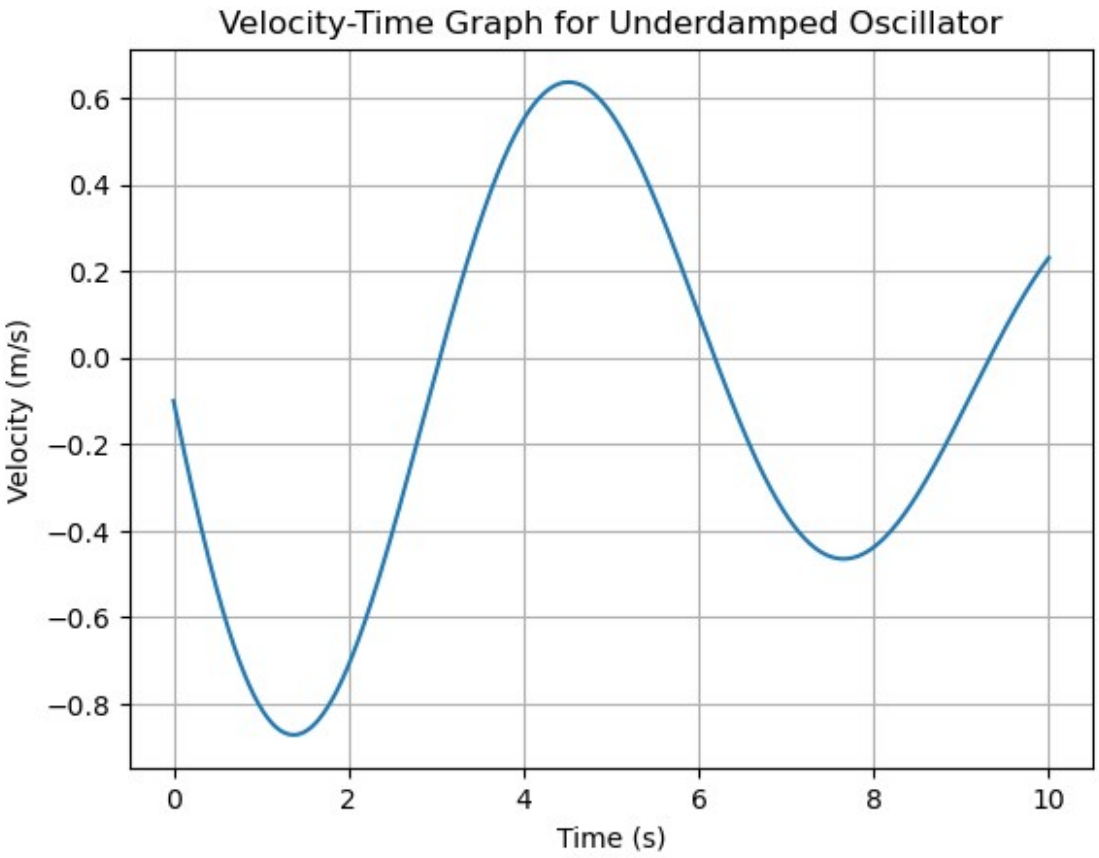
```python
import matplotlib.pyplot as plt
import numpy as np

# Function to plot displacement-time graph for a damped oscillator
def plot_displacement(omega, alpha, t_start, t_end, title):
    # Generate time array
    t = np.linspace(t_start, t_end, 1000)
    # Calculate displacement
    x = np.exp(-alpha * t) * np.cos(omega * t)
    # Plot the graph
    plt.plot(t, x)
    plt.title(title)
    plt.xlabel("Time (s)")
    plt.ylabel("Displacement (m)")
    plt.grid()
```

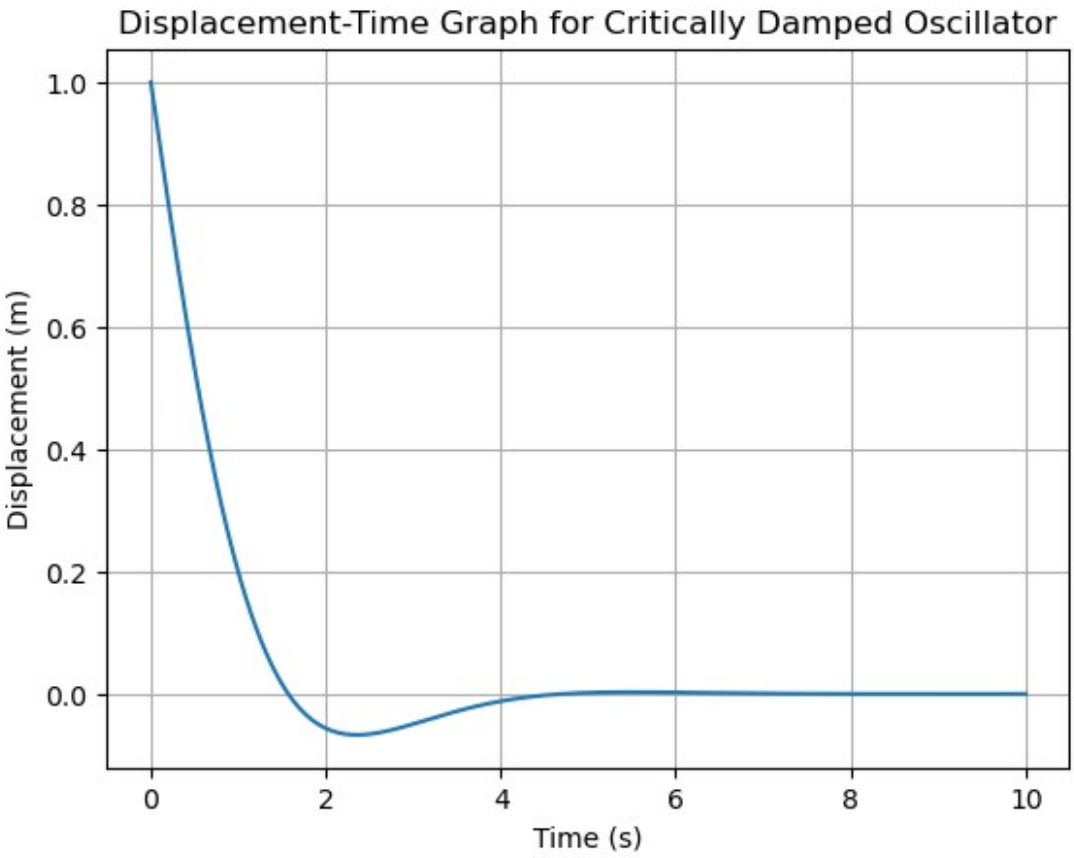
```
    plt.show()

# Function to plot velocity-time graph for a damped oscillator
def plot_velocity(omega, alpha, t_start, t_end, title):
    # Generate time array
    t = np.linspace(t_start, t_end, 1000)
    # Calculate velocity
    v = -omega * np.exp(-alpha * t) * np.sin(omega * t) - alpha *
np.exp(-alpha * t) * np.cos(omega * t)
    # Plot the graph
    plt.plot(t, v)
    plt.title(title)
    plt.xlabel("Time (s)")
    plt.ylabel("Velocity (m/s)")
    plt.grid()
    plt.show()

# Plot displacement-time graph for undamped oscillator
plot_displacement(1, 0, 0, 10, "Displacement-Time Graph for Undamped
Oscillator")

# Plot velocity-time graph for undamped oscillator
plot_velocity(1, 0, 0, 10, "Velocity-Time Graph for Undamped
Oscillator")

# Plot displacement-time graph for underdamped oscillator
plot_displacement(1, 0.1, 0, 10, "Displacement-Time Graph for
Underdamped Oscillator")

# Plot velocity-time graph for underdamped oscillator
plot_velocity(1, 0.1, 0, 10, "Velocity-Time Graph for Underdamped
Oscillator")

# Plot displacement-time graph for critically damped oscillator
plot_displacement(1, 1, 0, 10, "Displacement-Time Graph for Critically
Damped Oscillator")

# Plot velocity-time graph for critically damped oscillator
plot_velocity(1, 1, 0, 10, "Velocity-Time Graph for Critically Damped
Oscillator")

# Plot displacement-time graph for overdamped oscillator
plot_displacement(1, 2, 0, 10, "Displacement-Time Graph for Overdamped
Oscillator")

# Plot velocity-time graph for overdamped oscillator
plot_velocity(1, 2, 0, 10, "Velocity-Time Graph for Overdamped
Oscillator")
```

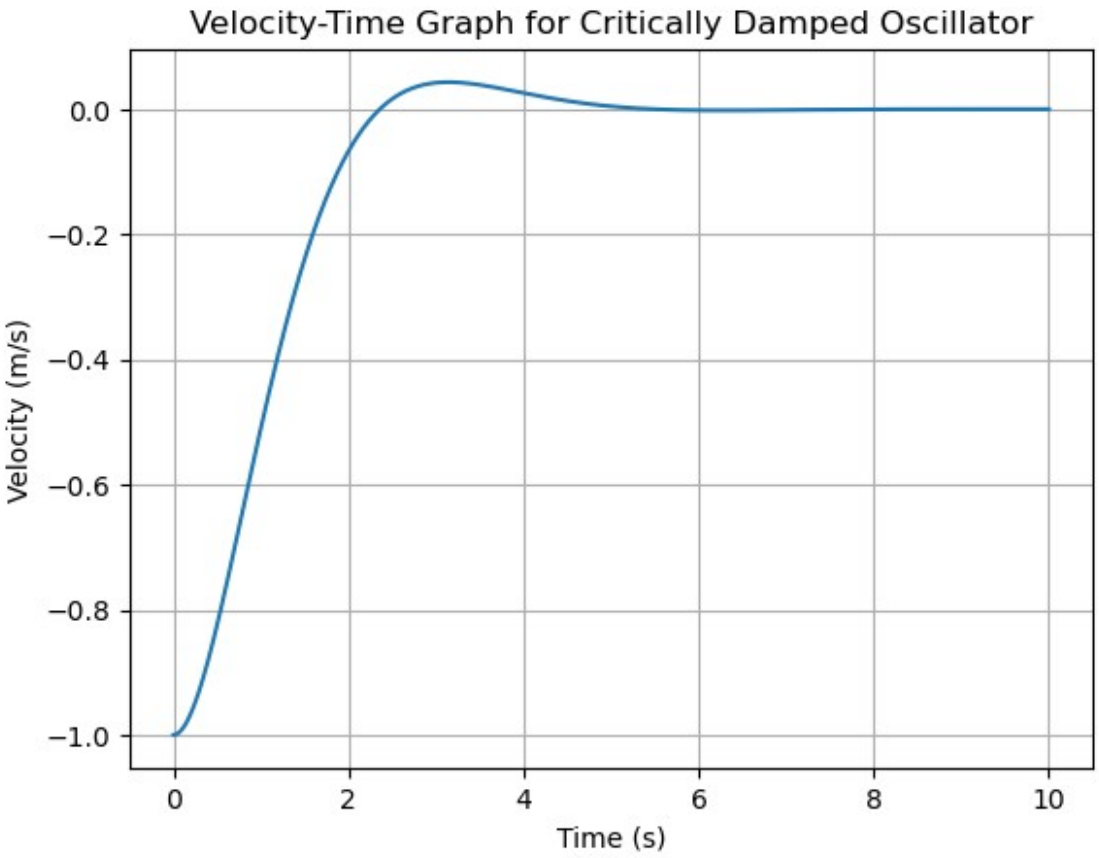

Displacement-Time Graph for Undamped Oscillator
Displacement (m)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2
4
6
8
10
Time (s)

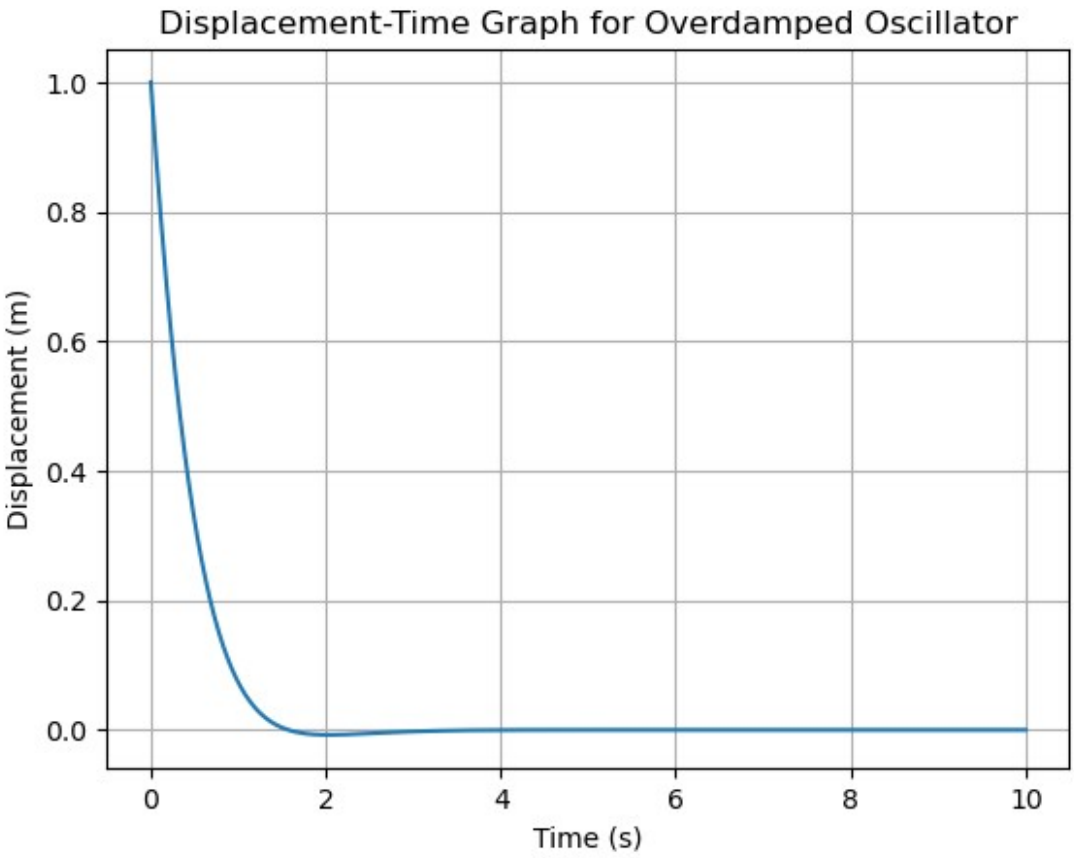

Velocity-Time Graph for Undamped Oscillator
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Velocity (m/s)
0
2
4
6
8
10
Time (s)

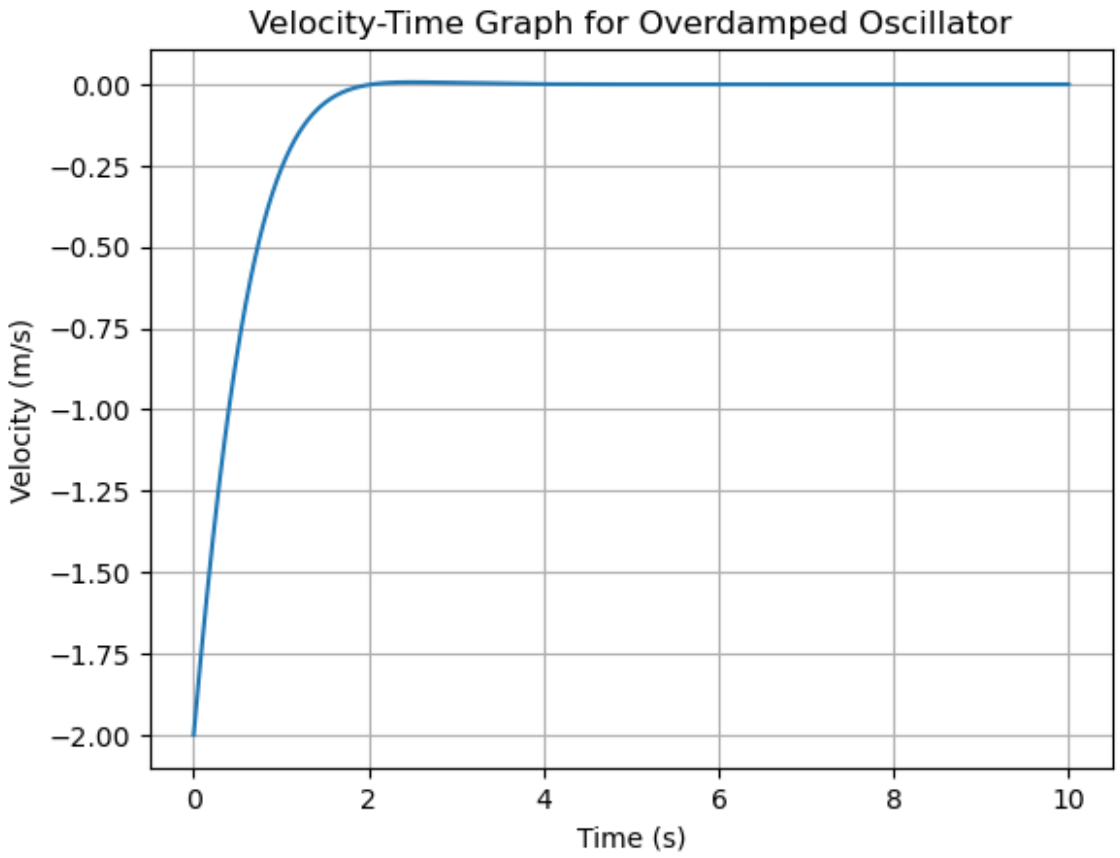

Displacement-Time Graph for Underdamped Oscillator
Displacement (m)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2
4
6
8
10
Time (s)


Velocity-Time Graph for Underdamped Oscillator
Velocity (m/s)
Time (s)
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0
2
4
6
8
10


Displacement-Time Graph for Critically Damped Oscillator
Displacement (m)
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
Time (s)


Velocity-Time Graph for Critically Damped Oscillator
Velocity (m/s)
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
0
2
4
6
8
10
Time (s)


Displacement-Time Graph for Overdamped Oscillator
Displacement (m)
Time (s)
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10

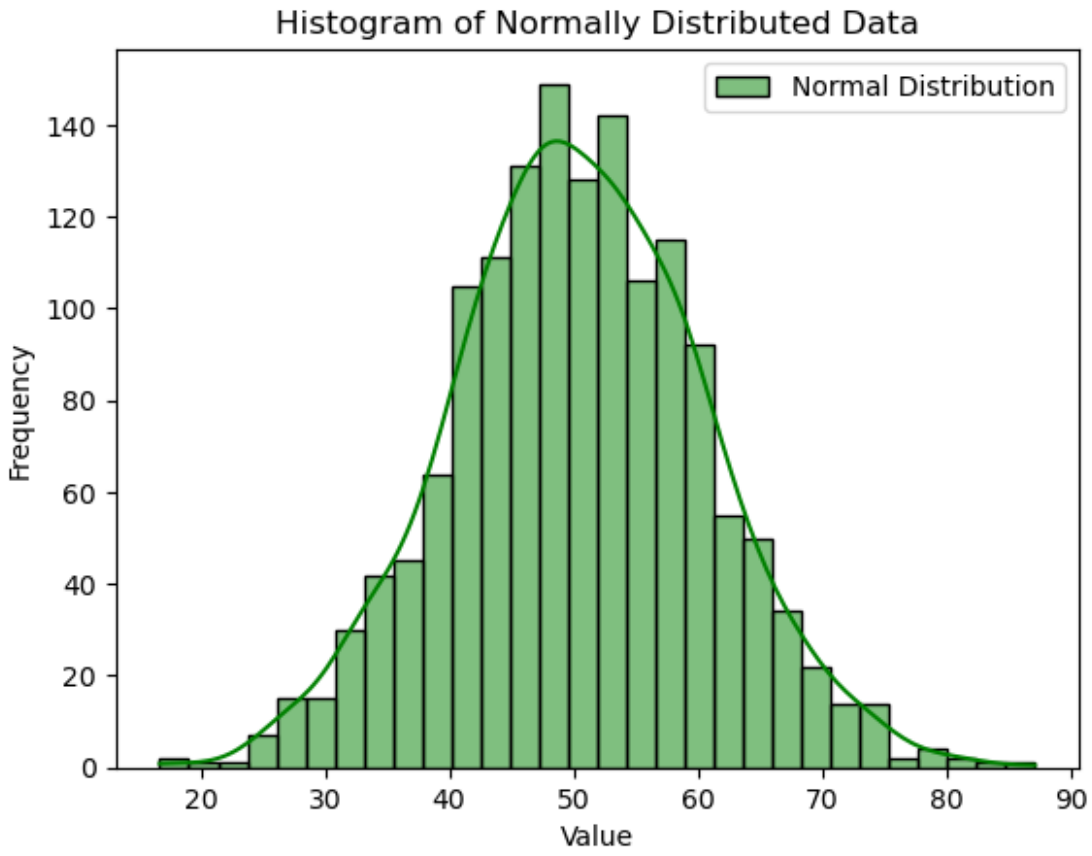


PROGRAM 9: To generate array of N random numbers drawn from NORMAL(GAUSSIAN) distribution and plot them using matplotlib for increasing N to verify the distribution.

```
import numpy as np  # Importing numpy
import matplotlib.pyplot as plt  # Importing matplotlib for plotting
import seaborn as sns  # Importing seaborn for advanced plotting

# Generate random numbers from a normal distribution
x = np.random.normal(loc=50, scale=10, size=1500)

# Print the generated random numbers
print(x)

# Plot the histogram of the random numbers
sns.histplot(x, bins=30, kde=True, label="Normal Distribution",
color="green")

# Add labels and title
plt.title("Histogram of Normally Distributed Data")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.legend()
```

```
# Show the plot
plt.show()
```

```
[45.13921693 59.63976151 32.22131938 ... 61.10546539 42.81088375
 24.74528129]
```

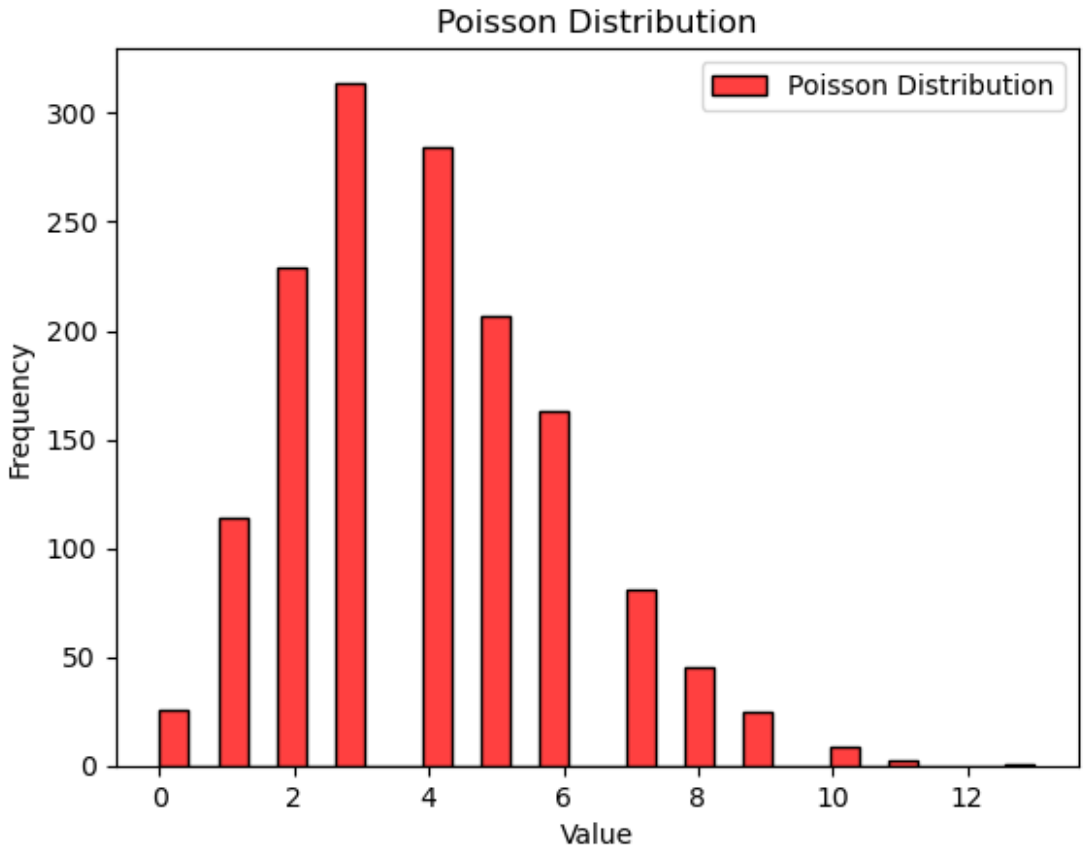


PROGRAM 10: To generate array of N random numbers drawn from POISSON distribution and plot them using matplotlib for increasing N to verify the distribution.

```
import numpy as np  # Importing numpy for numerical computations
import matplotlib.pyplot as plt  # Importing matplotlib for plotting
import seaborn as sns  # Importing seaborn for advanced visualizations

# Generate random numbers from a Poisson distribution
x = np.random.poisson(lam=4, size=1500)

# Print the generated numbers
print(x)

# Plot the distribution of the Poisson random numbers
sns.histplot(x, kde=False, bins=30, color="red", label="Poisson
Distribution")
```

```
# Add titles and labels
plt.title("Poisson Distribution")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.legend()

# Display the plot
plt.show()
```

```
[3 2 5 ... 4 2 8]
```



PROGRAM 11: To generate array of N random numbers drawn from BINOMIAL distribution and plot them using matplotlib for increasing N to verify the distribution.

```
import numpy as np
import matplotlib.pyplot as plt
from math import comb

# Define the Binomial PMF function
def binomial_pmf(k, n, p):
    return comb(n, k) * (p*k) * ((1 - p)*(n - k))

# Set the parameters for the Binomial distribution
```

```
n = 10  # number of trials
p = 0.5  # probability of success

# Generate possible values of k (number of successes)
k_values = np.arange(0, n+1)

# Calculate the binomial probabilities for each k
binomial_probs = [binomial_pmf(k, n, p) for k in k_values]

# Plot the Binomial distribution as a bar chart
plt.bar(k_values, binomial_probs, alpha=0.6, color='b',
label='Binomial PMF')

# Add labels and title
plt.title(f'Binomial Distribution with n = {n}, p = {p}')
plt.xlabel('Number of successes (k)')
plt.ylabel('Probability')

# Show the plot
plt.show()
```