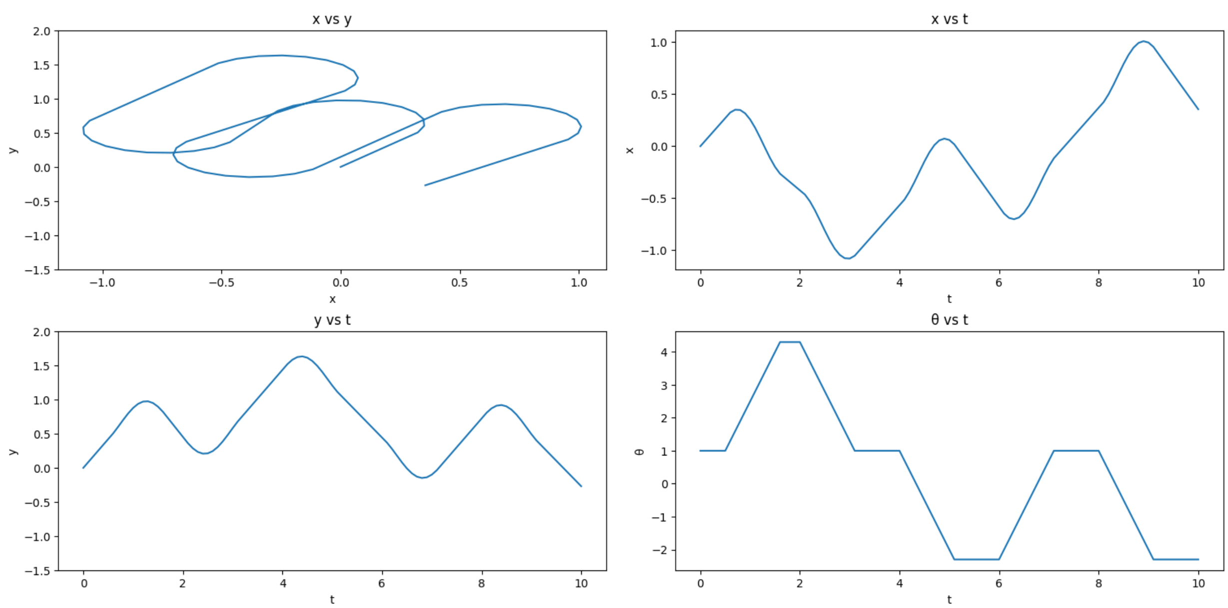
Answer to question no 2

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

# Define time-related parameters
T = 10.0
dt = 0.1
num_timesteps = int(T / dt) + 1

# Set the initial conditions x(0) = 0, y(0) = 0, q(0) = 1.)
x0, y0, theta0 = 0.0, 0.0, 1.0

# Initializing arrays for storing state variables
x, y, theta, t = np.zeros(num_timesteps), np.zeros(num_timesteps),
np.zeros(num_timesteps), np.linspace(0, T, num_timesteps)

# Define Control input functions
def input_control_v(t):
    return 1.0

def input_control_omega(t):
    intervals = [(0.5, 1.5, 3.0), (2.0, 3.0, -3.0), (4.0, 5.0, -3.0),
(6.0, 7.0, 3.0), (8.0, 9.0, -3.0)]
    for start, end, value in intervals:
        if start <= t <= end:
            return value
    return 0.0

# Simulating the model using Euler's method
x[0] = x0
y[0] = y0
theta[0] = theta0

for i in range(1, num_timesteps):
        v, omega = input_control_v(t[i - 1]), input_control_omega(t[i
- 1])
        x_dot, y_dot, theta_dot = v * np.cos(theta[i - 1]), v *
np.sin(theta[i - 1]), omega
        x[i], y[i], theta[i] = x[i - 1] + dt * x_dot, y[i - 1] + dt *
y_dot, theta[i - 1] + dt * theta_dot



def create_subplot(data, title, x_label, y_label, ylim=None):
    plt.subplot(2, 2, len(plt.gcf().get_axes()) + 1)
    plt.plot(*data)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
```

```
    plt.title(title)
    if ylim is not None:
        plt.ylim(ylim)

# Define subplot data
subplot_data = [
    [(x, y), 'x vs y', 'x', 'y', (-1.5, 2)],
    [(t, x), 'x vs t', 't', 'x', None],
    [(t, y), 'y vs t', 't', 'y', (-1.5, 2)],
    [(t, theta), 'θ vs t', 't', 'θ', None]
]

plt.figure(figsize=(14, 8))

for data, title, x_label, y_label, ylim in subplot_data:
    create_subplot(data, title, x_label, y_label, ylim)

# Showing the plot
plt.tight_layout()
plt.show()
```

Figure:

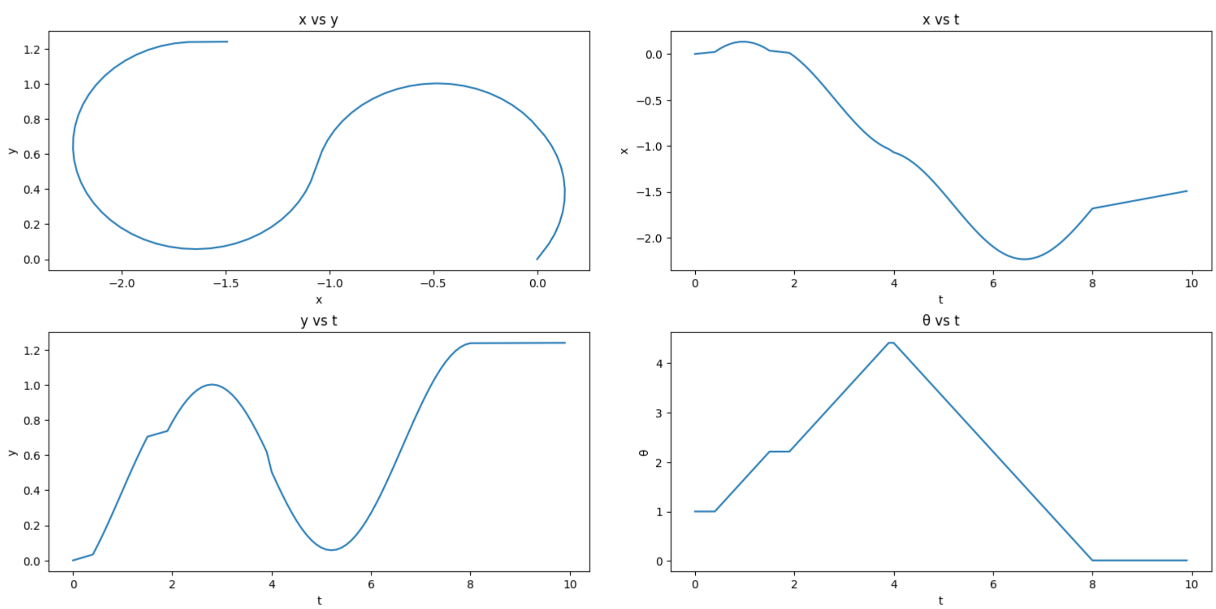


Answer to question no 3

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

# Robot parameters
r = 0.1
L = 1.0
del_t = 0.1
total_t = 10.0

# Initial conditions
x0 = [0]
y0 = [0]
theta0 = [1]

t_steps = np.arange(0, total_t, del_t)

# Control inputs
wl_value = [12 if 4 <= t <= 6 else 12 if 6 <= t <= 8 else 1 for t in
t_steps]
wr_value = [12 if 0.5 <= t <= 1.5 else 12 if 2 <= t <= 4 else 1 for t
in t_steps]


#Euler's method
for i in range(1, len(t_steps)):
    wl = wl_value[i]
    wr = wr_value[i]
    dx = (r / 2) * (wl + wr) * np.cos(theta0[-1]) * del_t
    dy = (r / 2) * (wl + wr) * np.sin(theta0[-1]) * del_t
    d_theta = (r / L) * (wr - wl) * del_t
    x0.append(x0[-1] + dx)
    y0.append(y0[-1] + dy)
    theta0.append(theta0[-1] + d_theta)

# Plotting
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.plot(x0, y0)
plt.title('x vs y')
plt.xlabel('x')
plt.ylabel('y')

plt.subplot(2, 2, 2)
plt.plot(t_steps, x0)
plt.title('x vs t')
plt.xlabel('t')
plt.ylabel('x')
```

```
plt.subplot(2, 2, 3)
plt.plot(t_steps, y0)
plt.title('y vs t')
plt.xlabel('t')
plt.ylabel('y')

plt.subplot(2, 2, 4)
plt.plot(t_steps, theta0)
plt.title('θ vs t')
plt.xlabel('t')
plt.ylabel('θ')

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

Figure:



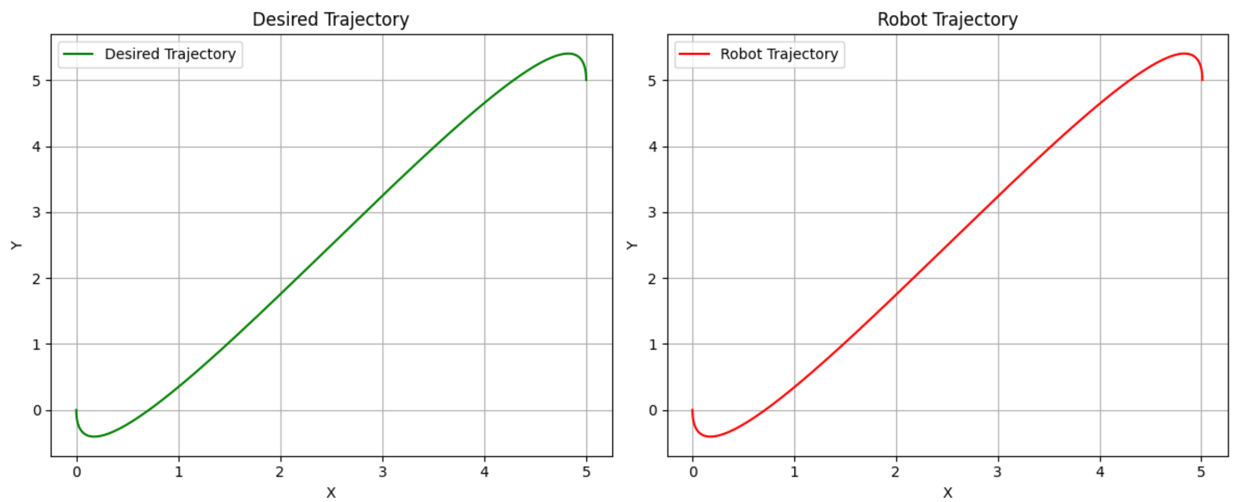
<u>Answer to question no 5</u>

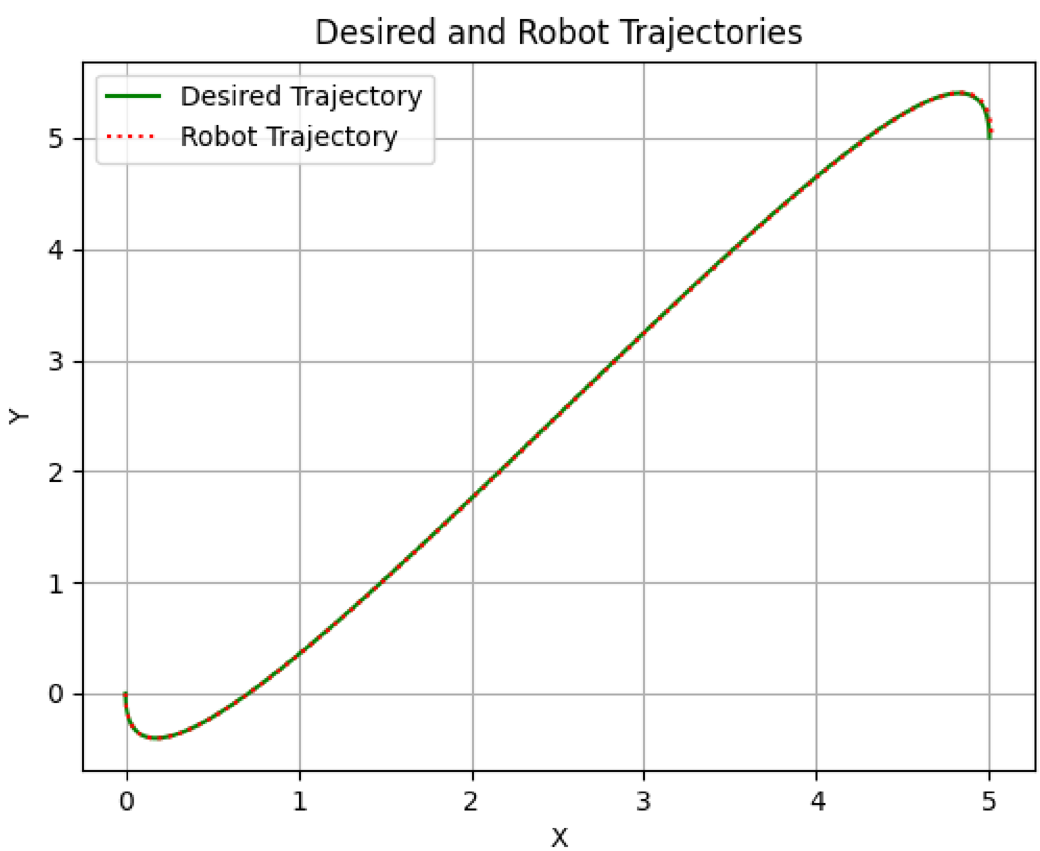
```
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0, 15, 0.01)
len(t)

#Final time T
T = 15
Tsq = np.power(T,2)
Tcb = np.power(T,3)

#initialized A
A = np.array([[1, 0, 0, 0,          0, 0, 0, 0],
              [0, 1, 0, 0,          0, 0, 0, 0],
              [0, 0, 0, 0,          1, 0, 0, 0],
              [0, 0, 0, 0,          0, 1, 0, 0],
              [1, T, Tsq, Tcb,      0, 0, 0, 0],
              [0, 1, 2*T, 3*Tsq,    0, 0, 0, 0],
              [0, 0, 0, 0,          1, T, Tsq, Tcb,],
              [0, 0, 0, 0,          0, 1, 2*T, 3*Tsq]
              ]
            )


#Pesudo inverse
A_inv = np.linalg.pinv(A)

#initialized b
b = np.array([[0],
               [0],
               [0],
               [-0.5],
               [5],
               [0],
               [5],
               [-0.5]
              ]
            )

#x = A_inv * b
x = np.matmul(A_inv, b)

a11 = x[0]
a12 = x[1]
a13 = x[2]
a14 = x[3]

a21 = x[4]
```

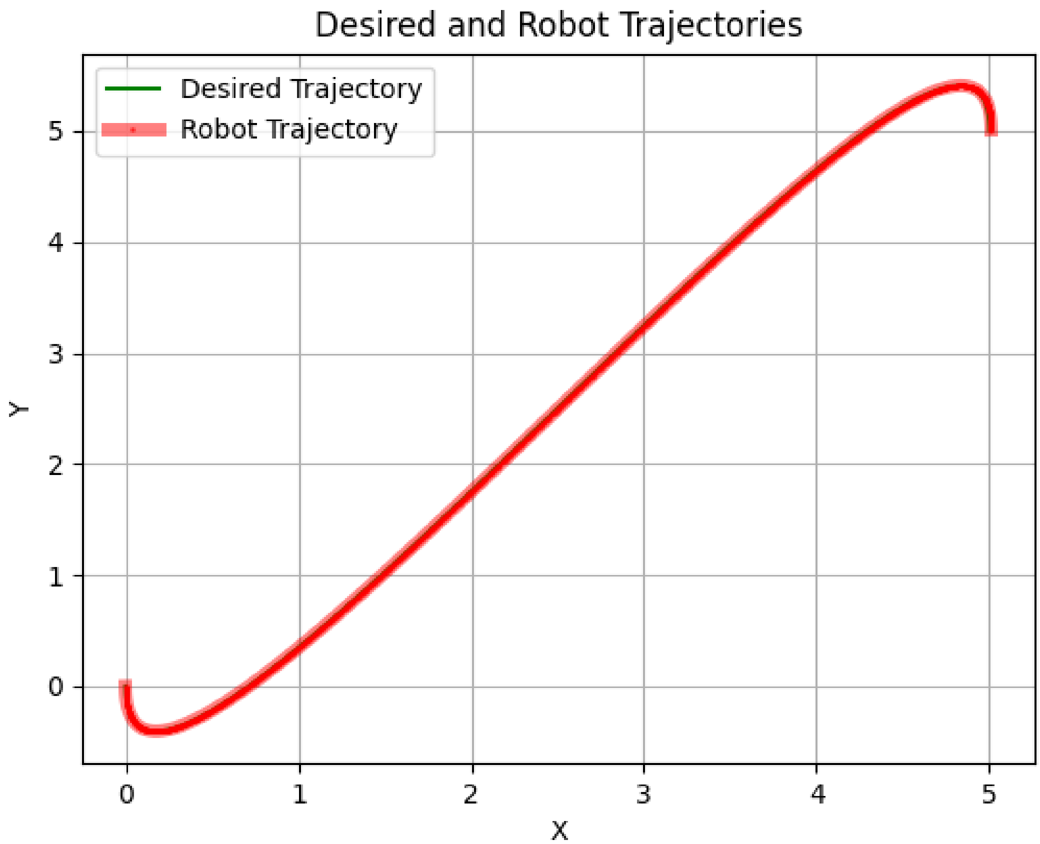
```
a22 = x[5]
a23 = x[6]
a24 = x[7]

for i in t:
  X = a11 + (a12* t) + (a13 * np.power(t,2)) + (a14 * np.power(t,3))
  Y = a21 + (a22* t) + (a23 * np.power(t,2)) + (a24 * np.power(t,3))


plt.plot(X,Y)
plt.title('X vs Y')
plt.xlabel('X')
plt.ylabel('Y')

X_new = a11 + a12 * t + a13 * t**2 + a14 * t**3
Y_new = a21 + a22 * t + a23 * t**2 + a24 * t**3

Xdd = np.gradient(np.gradient(X_new, t), t)
Ydd = np.gradient(np.gradient(Y_new, t), t)

theta = np.arctan2(np.gradient(Y_new, t), np.gradient(X_new, t))
V = np.sqrt(np.gradient(X_new, t)**2 + np.gradient(Y_new, t)**2)
a = np.cos(theta) * Xdd + np.sin(theta) * Ydd

omega = (-np.sin(theta) * Xdd + np.cos(theta) * Ydd) / V

x_final = X_new[0]
y_final = Y_new[0]
theta_final = theta[0]
V_final = V[0]

x_states = [x_final]
y_states = [y_final]


for i in range(1, len(t)):
    x_final += V_final * np.cos(theta_final) * (t[i] - t[i-1])
    y_final += V_final * np.sin(theta_final) * (t[i] - t[i-1])
    theta_final += omega[i] * (t[i] - t[i-1])
    V_final += a[i] * (t[i] - t[i-1])

    x_states.append(x_final)
    y_states.append(y_final)


plt.figure()
plt.plot(X, Y, label='Desired Trajectory', color='green')
plt.plot(x_states, y_states, label='Robot Trajectory',
linestyle='dotted', color='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
```

```python
plt.title('Desired and Robot Trajectories')
plt.grid(True)
plt.show()


plt.figure()
plt.plot(X, Y, label='Desired Trajectory', color='green')
plt.plot(x_states, y_states, label='Robot Trajectory', linestyle='-',
linewidth=5, color='red', alpha=0.5, marker='o', markersize=1,
markeredgecolor='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('Desired and Robot Trajectories')
plt.grid(True)
plt.show()


fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

ax1.plot(X, Y, label='Desired Trajectory', color='green')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax1.legend()
ax1.set_title('Desired Trajectory')
ax1.grid(True)

ax2.plot(x_states, y_states, label='Robot Trajectory',color='red')
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.legend()
ax2.set_title('Robot Trajectory')
ax2.grid(True)


plt.tight_layout()

plt.show()
```

Figure:


Desired Trajectory
Desired Trajectory
Robot Trajectory
Robot Trajectory
X
Y


Desired and Robot Trajectories
Desired Trajectory
Robot Trajectory
X
Y



Desired and Robot Trajectories
Desired Trajectory
Robot Trajectory
Y
X
0
1
2
3
4
5
0
1
2
3
4
5

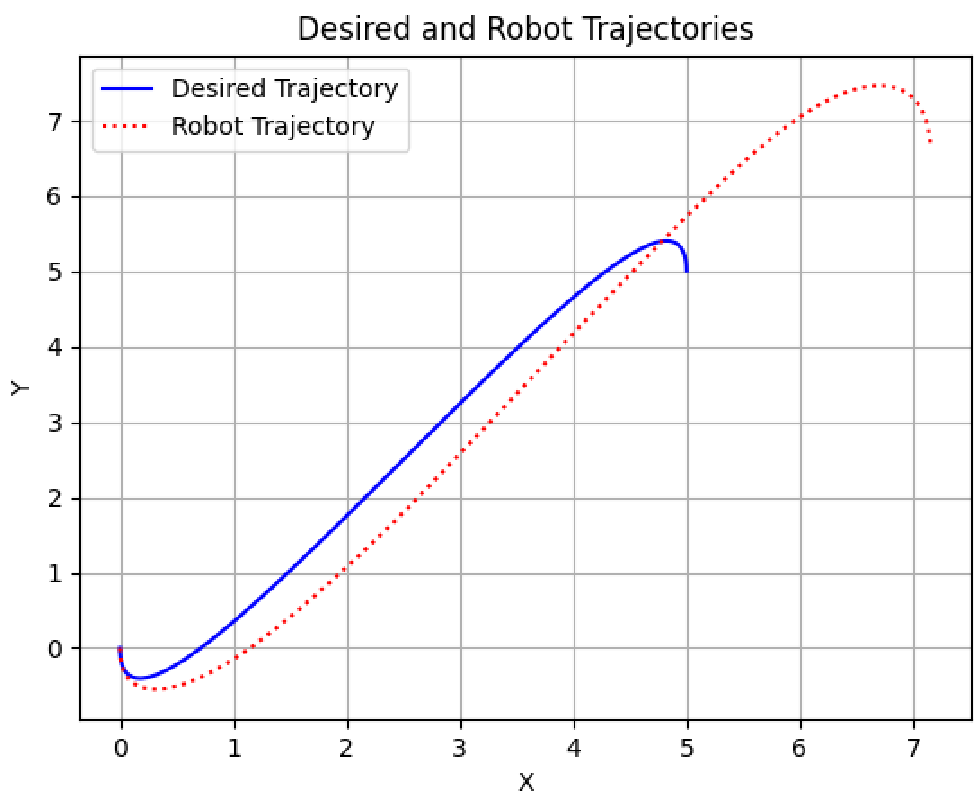
Answer to question no 6

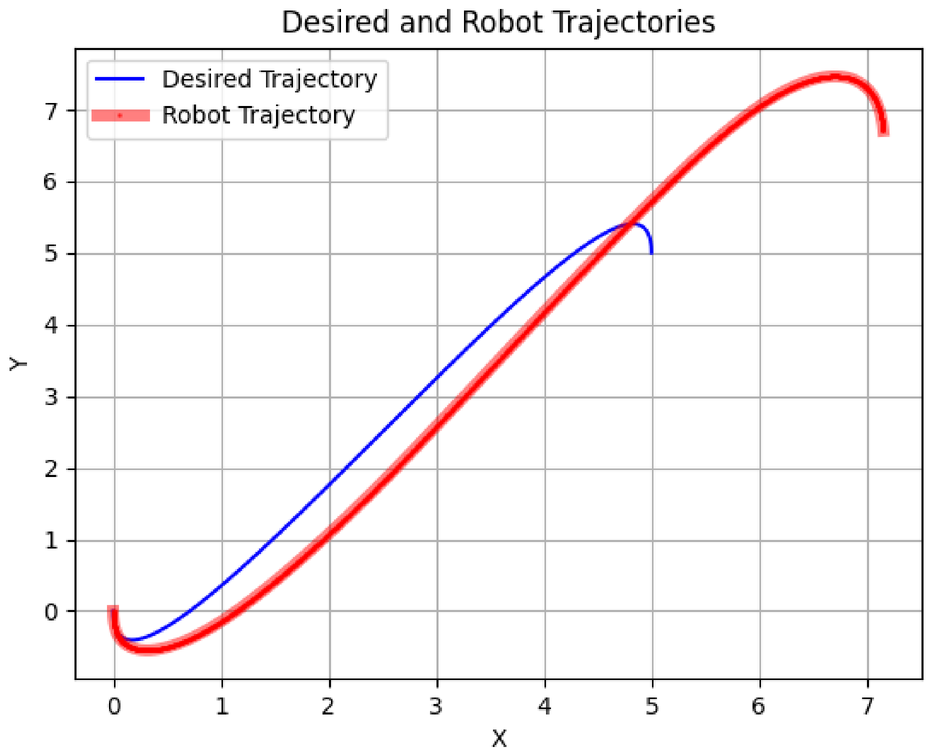
```
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0, 15, 0.01)
len(t)

#Final time T
T = 15
Tsq = np.power(T,2)
Tcb = np.power(T,3)

#nitialized A
A = np.array([[1, 0, 0, 0,          0, 0, 0, 0],
              [0, 1, 0, 0,          0, 0, 0, 0],
              [0, 0, 0, 0,          1, 0, 0, 0],
              [0, 0, 0, 0,          0, 1, 0, 0],
              [1, T, Tsq, Tcb,      0, 0, 0, 0],
              [0, 1, 2*T, 3*Tsq,    0, 0, 0, 0],
              [0, 0, 0, 0,          1, T, Tsq, Tcb,],
              [0, 0, 0, 0,          0, 1, 2*T, 3*Tsq]
              ]
            )

# Pesudo inverse
A_inv = np.linalg.pinv(A)

#Initialized b
b = np.array([[0],
               [0],
               [0],
               [-0.5],
               [5],
               [0],
               [5],
               [-0.5]
              ]
            )

#x = Ainv * b
x= np.matmul(A_inv, b)

a11 = x[0]
a12 = x[1]
a13 = x[2]
a14 = x[3]

a21 = x[4]
a22 = x[5]
```

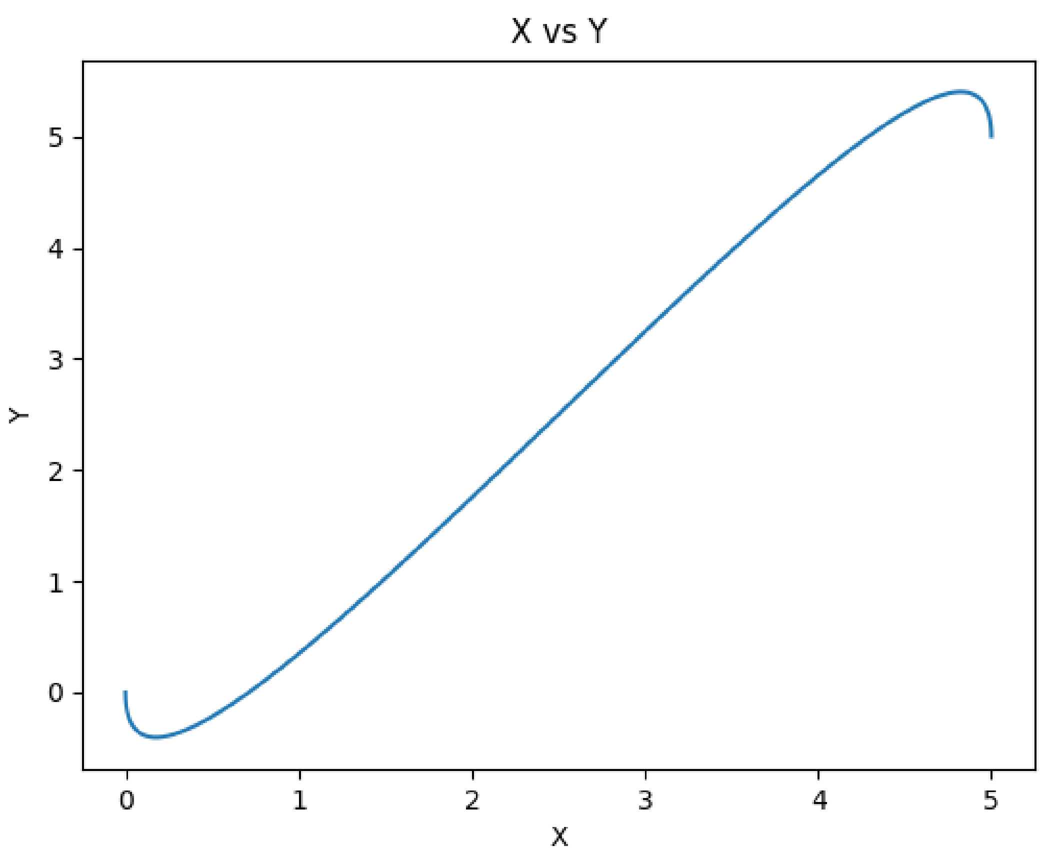
```
a23 = x[6]
a24 = x[7]

for i in t:
  X = a11 + (a12 * t) + (a13 * np.power(t,2)) + (a14 * np.power(t,3))
  Y = a21 + (a22 * t) + (a23 * np.power(t,2)) + (a24 * np.power(t,3))


plt.plot(X,Y)
plt.title('X vs Y')
plt.xlabel('X')
plt.ylabel('Y')

X_new = a11 + a12 * t + a13 * t**2 + a14 * t**3
Y_new = a21 + a22 * t + a23 * t**2 + a24 * t**3

Xdd = np.gradient(np.gradient(X_new, t), t)
Ydd = np.gradient(np.gradient(Y_new, t), t)

theta = np.arctan2(np.gradient(Y_new, t), np.gradient(X_new, t))
V = np.sqrt(np.gradient(X_new, t)**2 + np.gradient(Y_new, t)**2)
a = np.cos(theta) * Xdd + np.sin(theta) * Ydd

omega = (-np.sin(theta) * Xdd + np.cos(theta) * Ydd) / V

noise_std_v = 0.01
noise_std_theta = 0.001
noise_v = np.random.normal(0, noise_std_v, len(t))
noise_theta = np.random.normal(0, noise_std_theta, len(t))

# initialize
x_final = X_new[0]
y_final = Y_new[0]
theta_final = theta[0]
V_final = V[0]

x_states = [x_final]
y_states = [y_final]

for i in range(1, len(t)):
    x_final += V_final * np.cos(theta_final) * (t[i] - t[i-1])
    y_final += V_final * np.sin(theta_final) * (t[i] - t[i-1])

    theta_final += omega[i] * (t[i] - t[i-1]) + np.random.normal(0,
noise_std_theta)
    V_final += a[i] * (t[i] - t[i-1]) + np.random.normal(0,
noise_std_v)

    x_states.append(x_final)
    y_states.append(y_final)
```

```
plt.figure()
plt.plot(X, Y, label='Desired Trajectory', color='blue')
plt.plot(x_states, y_states, label='Robot Trajectory',
linestyle='dotted', color='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('Desired and Robot Trajectories')
plt.grid(True)
plt.show()

plt.figure()
plt.plot(X, Y, label='Desired Trajectory', color='blue')
plt.plot(x_states, y_states, label='Robot Trajectory', linestyle='-',
linewidth=5, color='red', alpha=0.5, marker='o', markersize=1,
markeredgecolor='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('Desired and Robot Trajectories')
plt.grid(True)
plt.show()
```

Figure:




Desired and Robot Trajectories
Desired Trajectory
Robot Trajectory
Y
X
0
1
2
3
4
5
6
7


X vs Y
Y
X
0
1
2
3
4
5