67678 - Introduction to Control with Learning **(Due: 02/06/24)**

Assignment 1 - LQR Control

Instructor: Oron Sabag *Name:* Hadar Tal

Instructions:

- The assignment is to be done individually.
- Submit your assignment as a single PDF file.
- Read all the Questions carefully before you start working on the assignment.
- This file contains extra materials for those who are interested in learning the tools used in the assignment's solution. Reading it is not required to complete the assignment.
- You can use any python library to solve the problems (e.g., *numpy, scipy, control, matplotlib,* etc.).

1 Continuous-time System

In the lecture, we focused on discrete linear systems. However, real-world systems often operate in continuous time, making it essential to understand the transition to continuous-time systems and the associated control strategies.

1.1 Motivation for Continuous-time Systems

Continuous-time systems are ubiquitous in engineering and natural processes. Examples include electrical circuits, mechanical systems, and biological systems. Unlike discrete systems, which are defined at specific time intervals, continuous systems evolve over time according to differential equations. This continuous evolution provides a more accurate representation of physical phenomena, allowing for precise modeling and control.

1.2 Equations of the Dynamics

The state-space representation of a continuous-time linear system is given by the following set of differential equations:

$$\dot{x}(t) = Ax(t) + Bu(t) \tag{1.1}$$

where:

- $x(t) \in \mathbb{R}^n$ is the state vector.
- $u(t) \in \mathbb{R}^m$ is the control input.
- $A \in \mathbb{R}^{n \times n}$ is the state matrix.
- $B \in \mathbb{R}^{n \times m}$ is the input matrix.

1.3 Linear Quadratic Regulator (LQR) Problem

The LQR problem for continuous-time systems involves finding a control law that minimizes a quadratic cost function. The objective is to regulate the state of the system to the origin while minimizing the control effort. The cost function is defined as:

$$J(t) = \int_0^t \left(x(\tau)^T Q x(\tau) + u(\tau)^T R u(\tau) \right) d\tau \tag{1.2}$$

where $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{m \times m}$ determine the relative importance of the state and control effort in the cost function.

1.4 Comparison to Discrete-time LQR

In discrete-time systems, the state-space representation is defined by difference equations rather than differential equations. The discrete-time LQR problem is similar to its continuous-time counterpart but with a cost function summed over discrete time steps. The key equations are:

$$x_{k+1} = A_d x_k + B_d u_k, \quad k = 0, 1, 2, \ldots \tag{1.3}$$

$$J_N(u^N) = \sum_{k=0}^{N} \left(x_k^T Q x_k + u_k^T R u_k \right) + x_{N+1}^T Q_f x_{N+1} \tag{1.4}$$

$$u_k = -K_d x_k, \tag{1.5}$$

where K_d is the optimal control gain matrix.

1.5 The Optimal Control

In the lecture, we discussed the determination of the control gain matrix K for discrete-time systems using an iterative approach. This method involved solving a finite-horizon cost function through backward iteration, essentially using dynamic programming techniques. However, for continuous-time systems, the process leverages the ***Continuous Algebraic Riccati equation*** for an infinite-horizon cost function, providing a more direct and analytical solution (3).

2 Inverted Pendulum on a Cart

The inverted pendulum on a cart is a classic problem in control theory and dynamics, often used to illustrate and test various control strategies. The system consists of a pendulum attached to a cart that can move horizontally. The goal is to design a controller that can stabilize the pendulum at the top position while regulating the position of the cart.

2.1 Formulation of the Problem

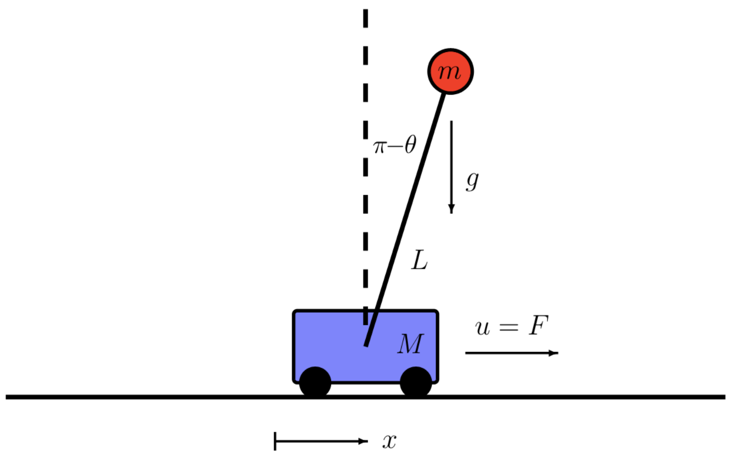

Figure 1: Schematic of inverted pendulum on a cart.

The dynamics of the system can be described by the following set of nonlinear differential equations:

$$\dot{x} = v \tag{2.1}$$

$$\ddot{x} = \frac{-m^2 L^2 g \cos(\theta)\sin(\theta) + mL^2 \left(mL\omega^2 \sin(\theta) - \delta v \right) + mL^2 u}{mL^2 \left(M + m(1 - \cos(\theta)^2) \right)} \tag{2.2}$$

$$\dot{\theta} = \omega \tag{2.3}$$

$$\ddot{\theta} = \frac{(m + M) mgL \sin(\theta) - mL\cos(\theta) \left(mL\omega^2 \sin(\theta) - \delta v \right) + mL\cos(\theta) u}{mL^2 \left(M + m(1 - \cos(\theta)^2) \right)} \tag{2.4}$$

where x is the cart position, v is the velocity, θ is the pendulum angle, ω is the angular velocity, m is the pendulum mass, M is the cart mass, L is the pendulum arm length, g is the gravitational acceleration, δ is a friction damping on the cart, and u is a control force applied to the cart.

Problem 1 - Modeling the Inverted Pendulum System

(a) Implement the function `dynamics` in the provided Python file `inverted_pendulum.py`.

```
def dynamics(y, m, M, L, g, delta, u):
    """
    Compute the state derivative.
    :param y: state vector
    :param m: pendulum mass
    :param M: cart mass
    :param L: pendulum length
    :param g: gravitational acceleration
    :param delta: friction damping
    :param u: control input
    :return dy: state derivative
    """
```

Solution:

```
def dynamics(y, m, M, L, g, delta, u):
    """
    Compute the state derivatives.
    :param y: state vector
    :param m: pendulum mass
    :param M: cart mass
    :param L: pendulum length
    :param g: gravitational acceleration
    :param delta: friction damping
    :param u: control input
    :return: dy: state derivative
    """
    theta = y[2]
    theta_dot = y[3]
    sin_theta = np.sin(theta)
    cos_theta = np.cos(theta)
    D = m * (L ** 2) * (M + m * (1 - cos_theta ** 2))
    dy = np.array([
        y[1],
        (1 / D) * (
            -(m ** 2) * (L ** 2) * g * cos_theta * sin_theta
            + m * (L ** 2) * (m * L * (theta_dot ** 2) * sin_theta - delta * y[1])
        ) + m * (L ** 2) * (1 / D) * u,
        y[3],
        (1 / D) * (
            (m + M) * m * g * L * sin_theta
            - m * L * cos_theta * (m * L * (theta_dot ** 2) * sin_theta
            - delta * y[1])
        ) - m * L * cos_theta * (1 / D) * u
    ])
    return dy
```

(b) Are there exist matrices A and B such that $\forall t \in \mathbb{R}_{\geq 0}$, $\dot{\mathbf{y}}_\mathbf{t} = A\mathbf{y}_\mathbf{t} + Bu_t$? Justify your answer.

The system dynamics are nonlinear, as shown in equations 2.1 - 2.4. Therefore, there are no matrices A and B that can linearize the system dynamics at all time points. Linearization is only possible around specific operating points, such as the upright position of the pendulum.

(c) Linearize the system dynamics around the upright position $\mathbf{y}_{goal} = [0, 0, \pi, 0]$ to obtain the linearized state-space representation:

$$\dot{\mathbf{x}} = A\mathbf{x} + Bu$$

where A and B are the state and input matrices, respectively.

The linearized system dynamics around the upright position ($\theta = \pi$) are given by:

$$\dot{\mathbf{y}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{\delta}{M} & \frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{\delta}{ML} & -\frac{(M+m)g}{ML} & 0 \end{bmatrix} \mathbf{y} + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ \frac{1}{ML} \end{bmatrix} u$$

(d) Implement the following functions.

```
def get_a_matrix(m, M, L, g, delta):
    """
    :return A: state matrix
    """

def get_b_matrix(m, M, L, g, delta):
    """
    :return B: input matrix
    """
```

Solution:

```
def get_a_matrix(m, M, L, g, delta):
    """
    :return A: state matrix
    """
    A = np.array([
        [0, 1, 0, 0],
        [0, -delta / M, m * g / M, 0],
        [0, 0, 0, 1],
        [0, -delta / (M * L), - (M + m) * g / (M * L), 0]
    ])
    return A

def get_b_matrix(m, M, L, g, delta):
    """
    :return B: input matrix
    """
    B = np.array([
        [0],
        [1 / M],
        [0],
        [1 / (M * L)]
    ])
    return B
```

Problem 2 - LQR Control for the Inverted Pendulum System

We will set the following parameters for the inverted pendulum system:

- Pendulum mass: $m = 0.1$ kg
- Cart mass: $M = 1.0$ kg
- Pendulum length: $L = 1$ m
- Gravitational acceleration: $g = 9.81$ m/s^2
- Friction damping: $\delta = 0.1$ Ns/m

(a) Calculate and print the State feedback gain matrix K for the LQR controller. Use the following cost matrices:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = 1$$

Bonus points will be given for using the continuous-time Riccati equation.

Solution:

```
Q = np.diag([1,1,1,1])
R = np.array([[1]])
K, S, E = control.lqr(A, B, Q, R)
print("State feedback gain matrix K:")
print(K)
print("Eigenvalues of the closed-loop system:")
print(E)
```

(b) Implement the following function.

```
    def dynamics_with_control(y, m, M, L, g, delta, K):
        """
        Compute the state derivative with control input.
        :param K: state feedback gain matrix
        :return dy: state derivative
        """
```

Solution:

```
def dynamics_with_control(y, t, y_goal, m, M, L, g, delta, K):
    """
    Compute the state derivative with control input.
    :param K: state feedback gain matrix
    :return dy: state derivative
    """
    u = np.dot(-K, y - y_goal )[0]
    return dynamics(y, m, M, L, g, delta, u)
```

(c) Simulate the system with the LQR controller for the initial state $\mathbf{y_0} = [2, 0, (1 + 0.1)\pi, 0]$ and a control input $u_t = -K \cdot (\mathbf{y_t} - \mathbf{y}_{goal})$. Plot the control input, cart position, and pendulum angle over time. Use the following simulation parameters:

- Simulation time: $t_f = 10$ s
- dt : $\frac{1}{30}$

Solution:

```
def degrees_to_radians(y):
    return (y % (2 * np.pi)) / np.pi


def compute_control_input(y, y_goal, K):
    """
    Compute the control input vector u based on the state vector y.
    :param y: state vector
    :param y_goal: goal state vector
    :param K: state feedback gain matrix
    :return: control input vector u
    """
    return np.array([np.dot(-K, y_i - y_goal) for y_i in y])


def plot_results(t, u, y):
    fig, axs = plt.subplots(3, 1, figsize=(10, 10))

    axs[0].plot(t, u, label='Control Input')
    axs[0].set_ylabel('Force (N)')
    axs[0].set_xlabel('Time (s)')
    axs[0].legend()
    axs[0].axhline(y=0, color='r', linestyle='--')

    axs[1].plot(t, y[:, 0], label='Cart Position')
    axs[1].set_ylabel('Position (m)')
    axs[1].set_xlabel('Time (s)')
    axs[1].legend()
    axs[1].axhline(y=0, color='r', linestyle='--')
```

```
axs[2].plot(t, degrees_to_radians(y[:, 2]), label='Pendulum Angle')
axs[2].set_ylabel('Angle (rad)')
axs[2].set_xlabel('Time (s)')
axs[2].legend()
axs[2].axhline(y=1, color='r', linestyle='--')
axs[2].invert_yaxis()
plt.tight_layout()
plt.show()
```

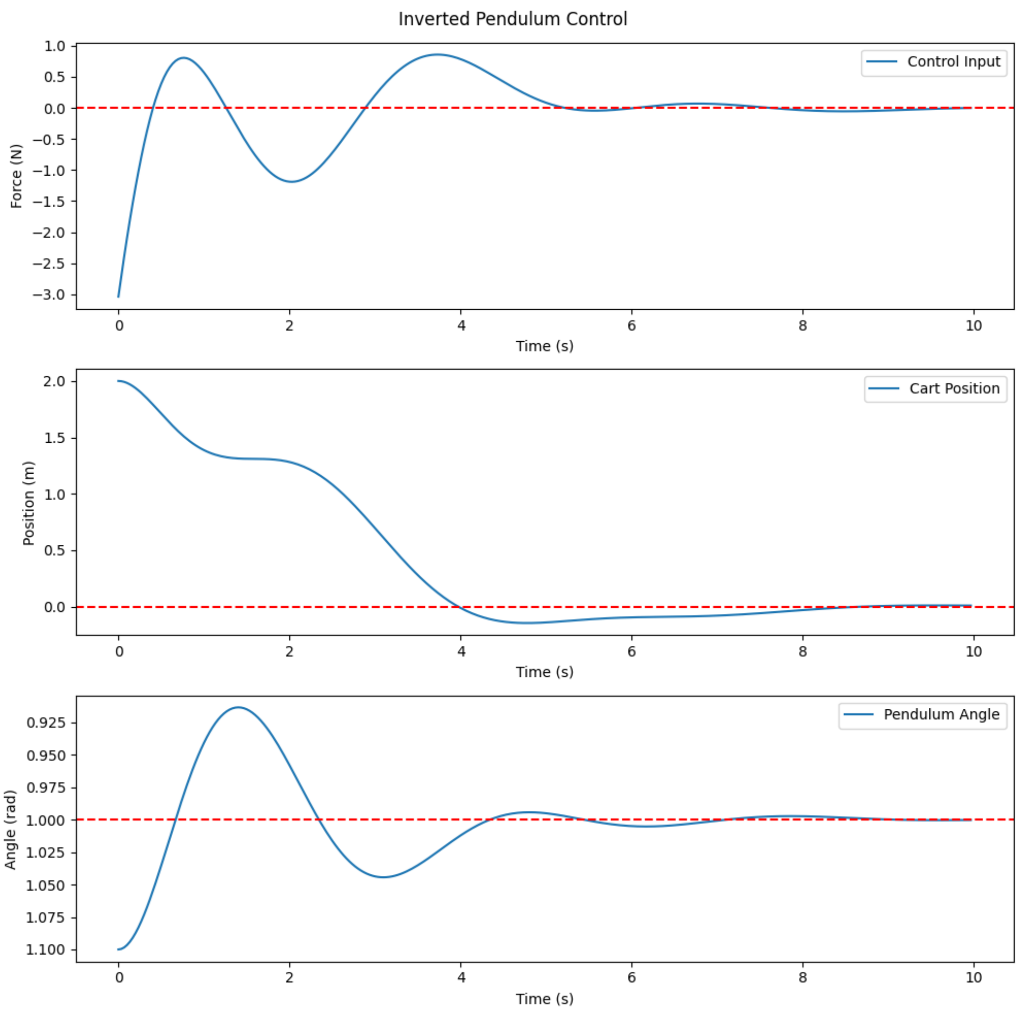

Figure 2: Simulation results for the inverted pendulum system with LQR control.

(d) Simulate the inverted pendulum system with the LQR controller for various initial states $\mathbf{y_0} = [x_0, 0, \pi, 0]$, where x_0 ranges from 0 to 10.5. Use a gradient color scheme to represent the initial position x_0, transitioning from blue to red as x_0 increases. Plot the control input, cart position, and pendulum angle over time. Explain what happens to the control input and the system states over time. Use the following simulation parameters:

- Simulation time: $t_f = 15$ s
- dt : $\frac{1}{30}$
- Number of initial positions: $N = 50$

Solution:

```
color1 = "blue"
color2 = "red"
N = 50
cmap = LinearSegmentedColormap.from_list("custom_cmap", [color1, color2], N=N)
colors = cmap(range(0, N+1))

tf = 15
dt = float(1 / 30)
t = np.arange(0, tf, dt)

A = get_a_matrix(m, M, L, g, delta)
B = get_b_matrix(m, M, L, g, delta)
Q = np.diag([1,1,1,1])
R = np.array([[1]])
y_goal = np.array([0, 0, np.pi * 1, 0])

K, S, E = control.lqr(A, B, Q, R)

fig, axs = plt.subplots(3, 1, figsize=(15, 25))
axs[0].set_ylabel('Force (N)')
axs[0].set_xlabel('Time (s)')
axs[0].axhline(y=0, color='r', linestyle='--')
axs[1].set_ylabel('Position (m)')
axs[1].set_xlabel('Time (s)')
axs[1].axhline(y=0, color='r', linestyle='--')
axs[2].set_ylabel('Angle (rad)')
axs[2].set_xlabel('Time (s)')
axs[2].axhline(y=1, color='r', linestyle='--')
axs[2].invert_yaxis()



for i, x0 in enumerate(np.linspace(0,10.5,N+1)):
    y0 = np.array([x0, 0, np.pi, 0])
    y = odeint(dynamics_with_control, y0, t, args=(y_goal, m, M, L, g, delta, K))
    u = compute_control_input(y, y_goal, K)
    axs[0].plot(t, u, color=colors[i])
    axs[1].plot(t, y[:, 0], color=colors[i])
    axs[2].plot(t, degrees_to_radians(y[:, 2]), color=colors[i])


fig.suptitle('System Response to Different Initial Positions', fontsize=20)
plt.show()
```

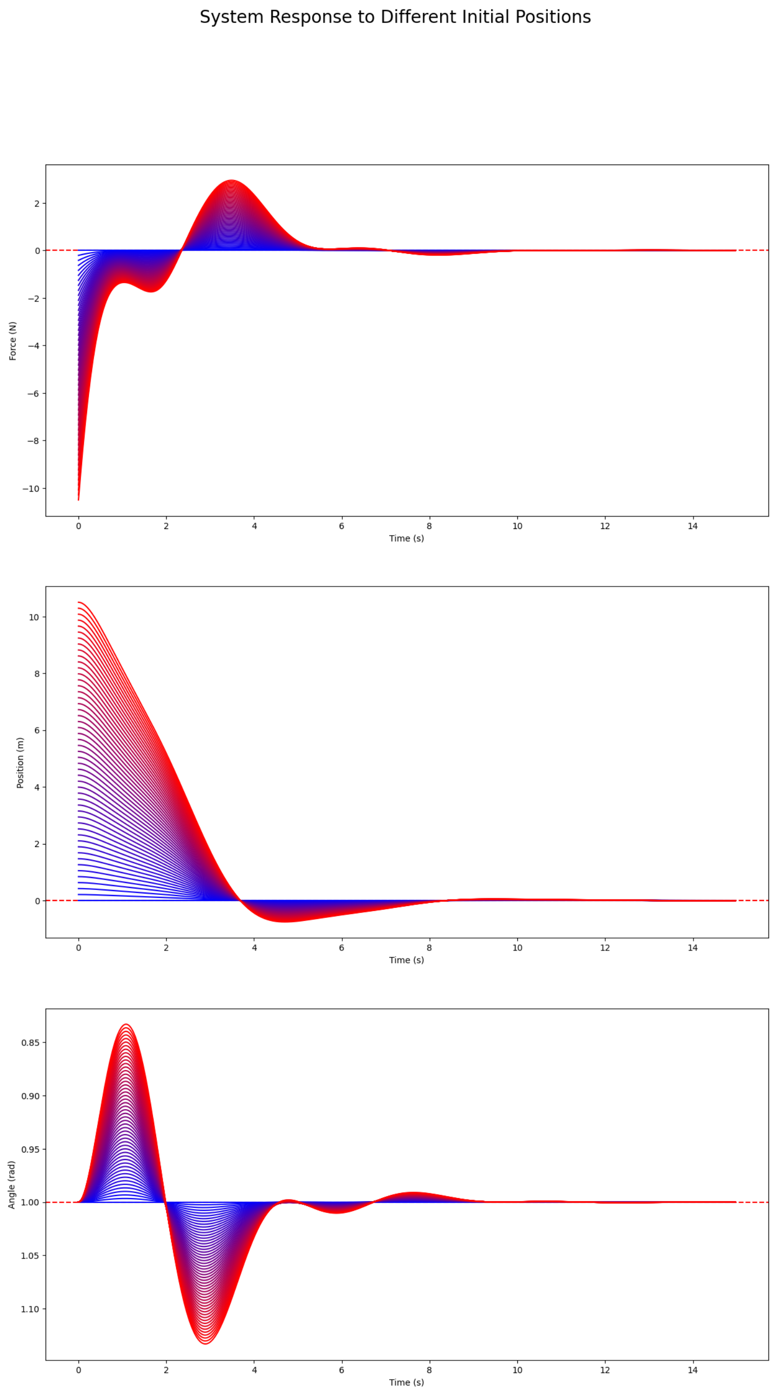

Figure 3: Simulation results for the inverted pendulum system with LQR control for various initial positions.

(e) Investigate the effect of varying the initial angle of the pendulum on the system's response. Simulate the inverted pendulum system for different initial angles θ_0 ranging from π to 0.3π, while keeping the initial position and velocities constant ($\mathbf{y_0} = [0, 0, \theta_0, 0]$). Use the LQR controller with $Q = \text{diag}(1, 1, 1, 1)$ and $R = 1$. Plot the control input, cart position, and pendulum angle over time. Use a gradient color scheme to represent different initial angles, transitioning from blue to red as the angle decreases. Explain how changing the initial angle affects the control input and the system states. Use the following simulation parameters:

- Simulation time: $t_f = 15$ s
- dt : $\frac{1}{30}$
- Number of initial angles: $N = 50$

Solution:

```
A = get_a_matrix(m, M, L, g, delta)
B = get_b_matrix(m, M, L, g, delta)
Q = np.diag([1,1,1,1])
R = np.array([[1]])
y_goal = np.array([0, 0, np.pi * 1, 0])

K, S, E = control.lqr(A, B, Q, R)

fig, axs = plt.subplots(3, 1, figsize=(15, 25))
axs[0].set_ylabel('Force (N)')
axs[0].set_xlabel('Time (s)')
axs[0].axhline(y=0, color='r', linestyle='--')
axs[1].set_ylabel('Position (m)')
axs[1].set_xlabel('Time (s)')
axs[1].axhline(y=0, color='r', linestyle='--')
axs[2].set_ylabel('Angle (rad)')
axs[2].set_xlabel('Time (s)')
axs[2].axhline(y=1, color='r', linestyle='--')
axs[2].invert_yaxis()


for i, radian in enumerate(np.linspace(0, 0.7, N+1)):
    y0 = np.array([0, 0, (1 - radian) * np.pi, 0])
    y = odeint(dynamics_with_control, y0, t, args=(y_goal, m, M, L, g, delta, K))
    u = compute_control_input(y, y_goal, K)
    axs[0].plot(t, u, color=colors[i])
    axs[1].plot(t, y[:, 0], color=colors[i])
    axs[2].plot(t, degrees_to_radians(y[:, 2]), color=colors[i])


fig.suptitle('System Response to Different Initial Angles', fontsize=20)
plt.show()
```

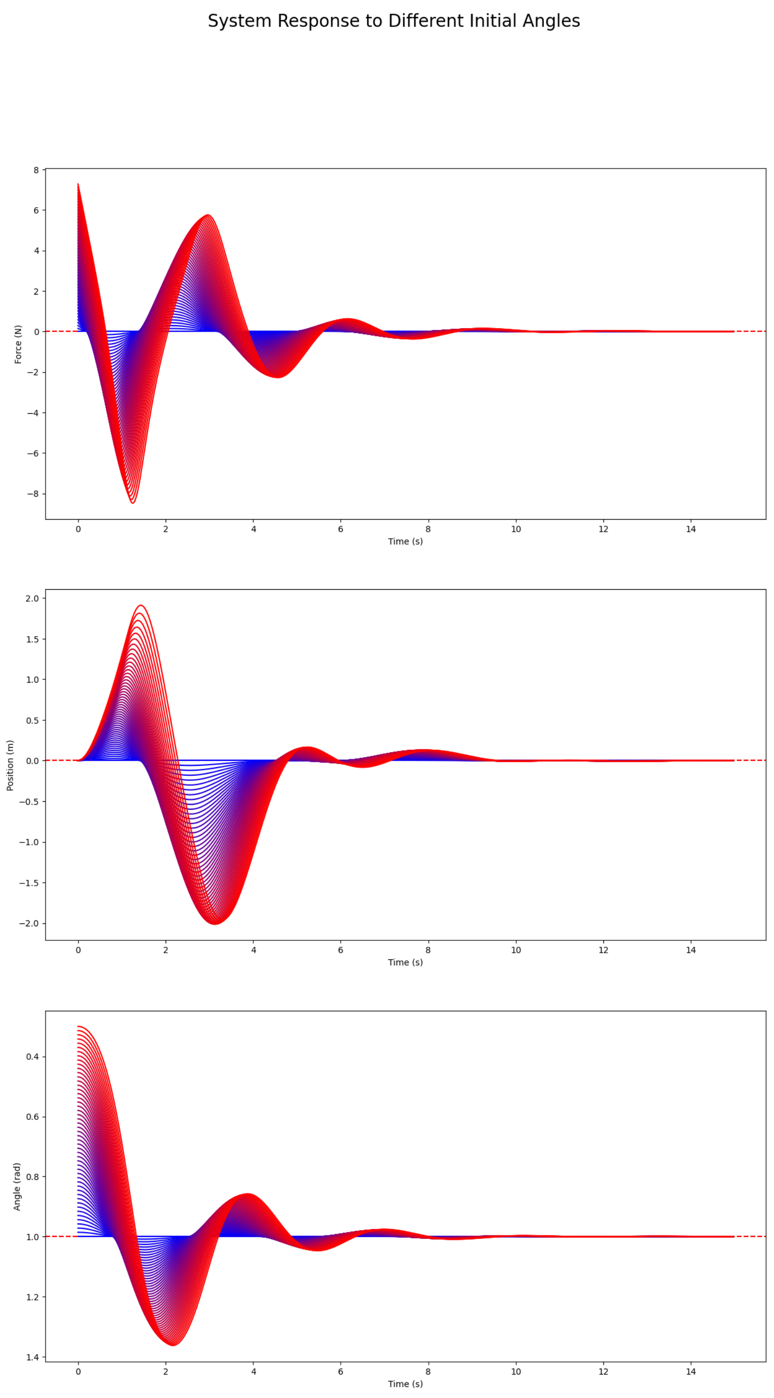

Figure 4: Simulation results for the inverted pendulum system with LQR control for various initial angles.

(f) Investigate the effect of varying the weighting matrix R in the LQR controller on the system's response. Simulate the inverted pendulum system for different values of R ranging from 1 to 50, while keeping the initial state $\mathbf{y_0} = [10, 0, \pi, 0]$ and the state weighting matrix $Q = \text{diag}(1, 1, 1, 1)$ constant. Plot the control input, cart position, and pendulum angle over time. Use a gradient color scheme to represent different values of R, transitioning from blue to red as R increases. Explain how changing R affects the control input and the system states. Use the following simulation parameters:

- Simulation time: $t_f = 15$ s
- dt : $\frac{1}{30}$
- Number of R values: $N = 50$

Solution:

```
A = get_a_matrix(m, M, L, g, delta)
B = get_b_matrix(m, M, L, g, delta)
Q = np.diag([1,1,1,1])
y_goal = np.array([0, 0, np.pi * 1, 0])
x0 = 10

fig, axs = plt.subplots(3, 1, figsize=(15, 25))
axs[0].set_ylabel('Force (N)')
axs[0].set_xlabel('Time (s)')
axs[0].axhline(y=0, color='r', linestyle='--')
axs[1].set_ylabel('Position (m)')
axs[1].set_xlabel('Time (s)')
axs[1].axhline(y=0, color='r', linestyle='--')
axs[2].set_ylabel('Angle (rad)')
axs[2].set_xlabel('Time (s)')
axs[2].axhline(y=1, color='r', linestyle='--')
axs[2].invert_yaxis()


for i, r in enumerate(np.linspace(1, 50, N+1)):
    R = np.array([[r]])
    K, S, E = control.lqr(A, B, Q, R)

    y0 = np.array([x0, 0, np.pi, 0])
    y = odeint(dynamics_with_control, y0, t, args=(y_goal, m, M, L, g, delta, K))
    u = compute_control_input(y, y_goal, K)
    axs[0].plot(t, u, color=colors[i])
    axs[1].plot(t, y[:, 0], color=colors[i])
    axs[2].plot(t, degrees_to_radians(y[:, 2]), color=colors[i])


fig.suptitle('System Response to Different Q values', fontsize=20)
plt.show()
```

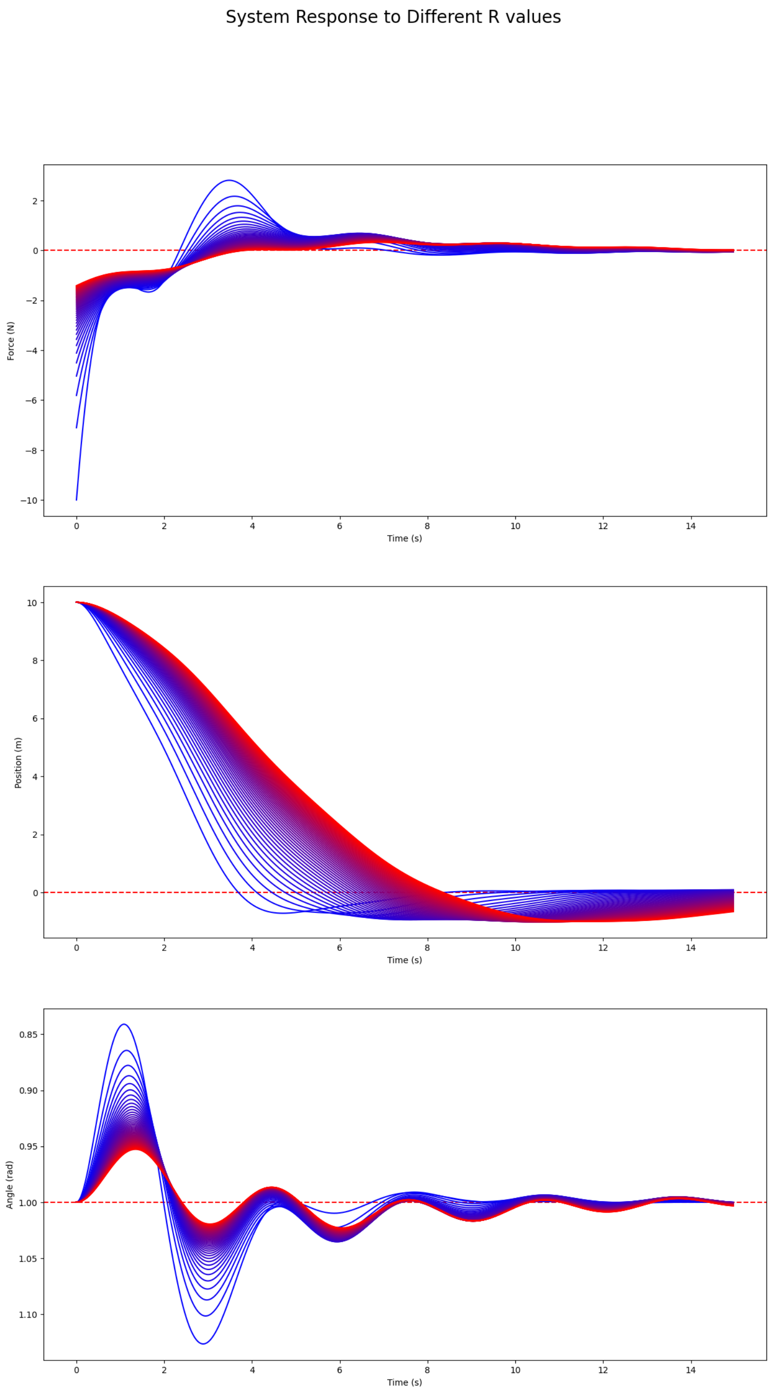

Figure 5: Simulation results for the inverted pendulum system with LQR control for different values of R.

(f) Investigate what happens when the initial angle of the pendulum is too small. Simulate the inverted pendulum system for initial angles θ_0 ranging from 0.2π to 0.1π, while keeping the initial position and velocities constant ($\mathbf{y_0} = [0, 0, \theta_0, 0]$). Use the LQR controller with $Q = \text{diag}(1, 1, 1, 1)$ and $R = 1$. Plot the control input, cart position, and pendulum angle over time. Explain why the system behaves differently for very small initial angles and write down what is the critical angle for the system to be unstable. Use the following simulation parameters:

- Simulation time: $t_f = 15$ s
- dt : $\frac{1}{30}$
- Number of initial angles: $N = 50$

Solution:

```
A = get_a_matrix(m, M, L, g, delta)
B = get_b_matrix(m, M, L, g, delta)
Q = np.diag([1,1,1,1])
R = np.array([[1]])
y_goal = np.array([0, 0, np.pi * 1, 0])

K, S, E = control.lqr(A, B, Q, R)

fig, axs = plt.subplots(3, 1, figsize=(15, 25))
axs[0].set_ylabel('Force (N)')
axs[0].set_xlabel('Time (s)')
axs[0].axhline(y=0, color='r', linestyle='--')
axs[1].set_ylabel('Position (m)')
axs[1].set_xlabel('Time (s)')
axs[1].axhline(y=0, color='r', linestyle='--')
axs[2].set_ylabel('Angle (rad)')
axs[2].set_xlabel('Time (s)')
axs[2].axhline(y=1, color='r', linestyle='--')
axs[2].invert_yaxis()


for i, radian in enumerate(np.linspace(0.877, 0.878, N+1)):
    y0 = np.array([0, 0, (1 - radian) * np.pi, 0])
    y = odeint(dynamics_with_control, y0, t, args=(y_goal, m, M, L, g, delta, K))
    u = compute_control_input(y, y_goal, K)
    axs[0].plot(t, u, color=colors[i])
    axs[1].plot(t, y[:, 0], color=colors[i])
    axs[2].plot(t, degrees_to_radians(y[:, 2]), color=colors[i])


fig.suptitle('System Response to Different Initial Angles - extream', fontsize=20)
plt.show()
```

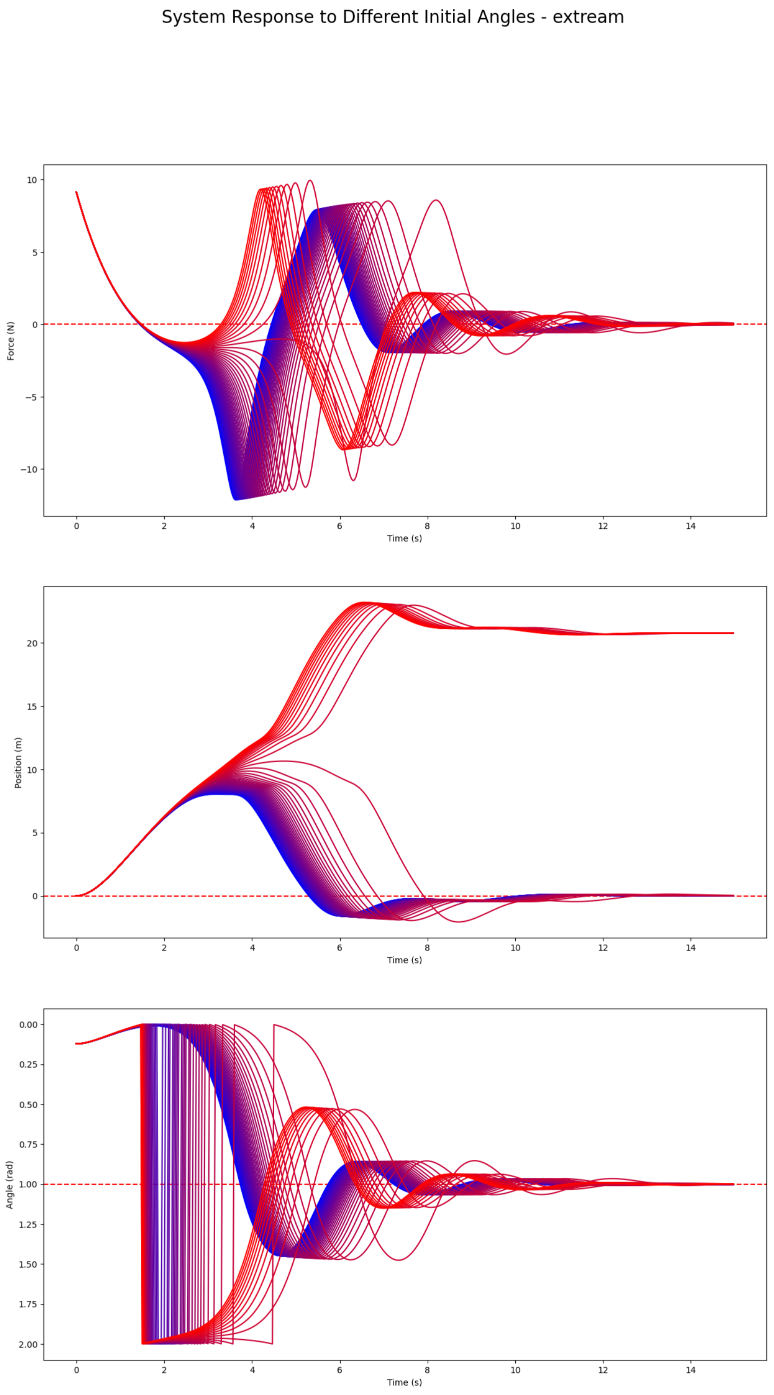

Figure 6: Simulation results for the inverted pendulum system with LQR control for various initial angles.

Extra material

3 Riccati Equation

This derivation of Riccati equation will provide an example of how to solve convex optimization problems using the ***calculus of variations***, and it will also provide a template for computing the optimal control solution for nonlinear systems.
First, we will add a terminal cost to our LQR cost function in 1.2, and also introduce a factor of 1/2 to simplify computations:

$$J = \int_0^{t_f} \underbrace{\frac{1}{2}\left(x^T Q x + u^T R u\right)}_{\text{Lagrangian } \mathcal{L}} d\tau + \underbrace{\frac{1}{2} x(t_f)^T Q_f x(t_f)}_{\text{Terminal cost}} \tag{3.1}$$

The goal is to minimize the quadratic cost function J subject to the dynamical constraint:

$$\dot{x} = Ax + Bu. \tag{3.2}$$

We may solve this using the calculus of variations by introducing the following augmented cost function:

$$J_{aug} = \int_0^{t_f} \left[\frac{1}{2}\left(x^T Q x + u^T R u\right) + \lambda^T (Ax + Bu - \dot{x})\right] dt + \frac{1}{2} x(t_f)^T Q_f x(t_f) \tag{3.3}$$

The variable λ is a ***Lagrange multiplier***, called the co-state, that enforces the dynamic constraints. λ may take any value and $J_{aug} = J$ will hold.

Taking the total variation of J_{aug} in 3.3 yields:

$$\delta J_{aug} = \int_0^{t_f} \left[\frac{\partial \mathcal{L}}{\partial x}\delta x + \frac{\partial \mathcal{L}}{\partial u}\delta u + \lambda^T A \delta x + \lambda^T B \delta u - \lambda^T \delta \dot{x}\right] dt + Q_f x(t_f) \delta x(t_f) \tag{3.4}$$

The partial derivatives of the Lagrangian are $\frac{\partial \mathcal{L}}{\partial x} = x^T Q$ and $\frac{\partial \mathcal{L}}{\partial u} = u^T R$. The last term in the integral may be modified using integration by parts:

$$-\int_0^{t_f} \lambda^T \delta \dot{x} dt = -\lambda^T(t_f)\delta x(t_f) + \lambda^T(0)\delta x(0) + \int_0^{t_f} \dot{\lambda}^T \delta x dt. \tag{3.5}$$

The term $\lambda^T(0)\delta x(0)$ is equal to zero, or else the control system would be non-causal (i.e., then future control could change the initial condition of the system).

Finally, the total variation of the augmented cost function in 3.4 simplifies as follows:

$$\delta J_{aug} = \int_0^{t_f} \left[x^T Q + \lambda^T A + \dot{\lambda}^T\right] \delta x dt + \int_0^{t_f} \left[u^T R + \lambda^T B\right] \delta u dt + (x(t_f)^T Q_f - \lambda^T(t_f))\delta x(t_f). \tag{3.6}$$

Each variation term in 3.6 must equal zero for an optimal control solution that minimizes J. Thus, we may break this up into three equations:

$$x^T Q + \lambda^T A + \dot{\lambda}^T = 0 \tag{3.7}$$

$$u^T R + \lambda^T B = 0 \tag{3.8}$$

$$x(t_f)^T Q_f - \lambda^T(t_f) = 0. \tag{3.9}$$

Note that the constraint in 3.9 represents an initial condition for the reverse-time equation for λ starting at t_f. Thus, the dynamics in 3.2 with initial condition $x(0) = x_0$ and 3.9 with the final-time condition $\lambda(t_f) = Q_f x(t_f)$ form a two-point boundary value problem. This may be integrated numerically to find the optimal control solution, even for nonlinear systems.

Because the dynamics are linear, it is possible to posit the form $\lambda = Px$, and substitute into 3.7 above. The first equation becomes:

$$\left(\dot{P}x + P\dot{x}\right)^T + x^T Q + \lambda^T A = 0. \tag{3.10}$$

Taking the transpose, and substituting 3.2 in for $\dot{x}$, yields:

$$\dot{P}x + P(Ax + Bu) + Qx + A^T Px = 0. \tag{3.11}$$

From 3.8, we have

$$u = -R^{-1}B^T\lambda = -R^{-1}B^T Px. \tag{3.12}$$

Finally, combining yields:

$$\dot{P}x + PAx + A^T Px - PBR^{-1}B^T Px + Qx = 0. \tag{3.13}$$

This equation must be true for all x, and so it may also be written as a matrix equation. Dropping the terminal cost and letting time go to infinity, the $\dot{P}$ term disappears, and we recover the ***Continuous Algebraic Riccati equation (CARE)*** :

$$PA + A^T P - PBR^{-1}B^T P + Q = 0. \tag{3.14}$$

Although this procedure is somewhat involved, each step is relatively straightforward. In addition, the dynamics in 3.2 may be replaced with nonlinear dynamics $\dot{x} = f(x, u)$, and a similar nonlinear two-point boundary value problem may be formulated with $\frac{\partial f}{\partial x}$ replacing A and $\frac{\partial f}{\partial u}$ replacing B. This procedure is extremely general, and may be used to numerically obtain nonlinear optimal control trajectories.

References

[1] S. Brunton, and J. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge: Cambridge University Press, 2019. doi:10.1017/9781108380690

[2] S. Brunton, *Linear Quadratic Regulator (LQR) Control for the Inverted Pendulum on a Cart [Control Bootcamp]*, 2017.