# SSY281 Model Predictive Control
## Assignment 1 – Review and preliminaries

The purpose of this assignment is to become familiar with the preliminary steps required for MPC, including the linearization of nonlinear state-space models, discretization, and selection of steady-state targets.

**Instructions**

The assignments comprise an important part of the examination in this course. Hence, it is important to comply with the following rules and instructions:

- The findings from each assignment are described in a short report. The report should be uploaded to Canvas *before the deadline*. A report uploaded a second or a day after the deadline are penalized equally.
- The report should provide clear and concise answers to the questions, including your motivations, explanations, observations from simulations, etc. Conclusions should be supported by relevant results if applicable.
- Figures included in the report should have legends, should be readable, should have proper scaling to illustrate the relevant information, and axes should be labeled.
- Since the assignments are part of the examination in the course, plagiarism is of course not allowed. If we observe that this happens anyway, it will be reported.
- Name the report as A1_X.pdf, where X is your group number (see Canvas page). The name of the group members should be included in the report.
- If applicable, a MATLAB or Python code should be uploaded which reproduces all numbers and figures in your report. Make sure that one can run your code and see your results without any error. Name the script as A1_X.m, or A1_X.py.

Table 1: Points per question

| Question: | 1 | 2 | 3 | 4 | 5 | Total |
|---|---|---|---|---|---|---|
| Points: | 4 | 0 | 1 | 3 | 1 | 9 |

A cart-pole system, as illustrated in the figure below, is described by the differential equations

$$\ddot{s}(t) = \frac{\frac{4}{3}\left(F(t) - ml\dot{\theta}(t)^2 \sin\theta(t)\right) + mg\sin\theta(t)\cos\theta(t)}{\frac{4}{3}M - m\cos^2\theta(t)},$$
$$\ddot{\theta}(t) = \frac{Mg\sin\theta(t) + \cos\theta(t)\left(F(t) - ml\dot{\theta}(t)^2 \sin\theta(t)\right)}{l\left(\frac{4}{3}M - m\cos^2\theta(t)\right)}.$$

where $s$ is the longitudinal displacement of the cart, $\theta$ is the angle of the pole, $g = 9.81\,\mathrm{m/s^2}$ is the gravitational acceleration, $m = 0.1\,\mathrm{kg}$ is mass of the pole, $M = 1.1\,\mathrm{kg}$ is the cumulative mass of the cart and the pole, $l = 0.5\,\mathrm{m}$ is the distance from the pole-cart attachment to the pole's center of mass, and $F[N]$ is longitudinal force exerted on the cart, e.g., by propelling the cart with a motor. The goal is to control $F$ such that the pole is kept in the vertical position, i.e., $\theta_{\mathrm{ref}} = 0\,\mathrm{rad}$.

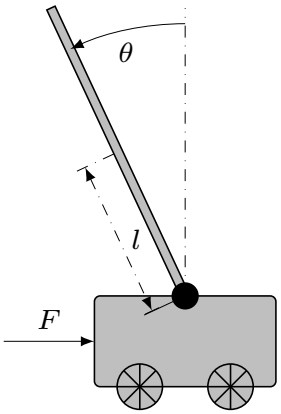


Figure 1: Cart-pole system; see here for details.

1. **Linearization and discretization of a state-space model**

   Let the nonlinear system be defined as

$$\dot{x}(t) = f(x(t), u(t)), \quad x(t) = \begin{bmatrix} s(t) \\ \dot{s}(t) \\ \theta(t) \\ \dot{\theta}(t) \end{bmatrix}, \quad u(t) = F(t),$$

   where $x$ and $u$ are the state and control vectors, respectively.

   (a) [2p] Using $\sin(\theta) \approx \theta$, $\cos(\theta) \approx 1$ for small $\theta$, first approximate the nonlinear system and then linearize about $x = \mathbf{0}$, $u = \mathbf{0}$. Provide the matrices $A_c$, $B_c$ and $C_c$, up to 4 decimal places, of the linear continuous-time system

$$\dot{x}(t) = A_c x(t) + B_c u(t)$$
$$y(t) = C_c x(t)$$

   where the output vector is $y(t) = \begin{bmatrix} s(t) & \theta(t) \end{bmatrix}^\top$. Find the eigenvalues of $A_c$. Is the uncontrolled ($u = 0$) system stable? Motivate your answer.

   **Hint:** You may use the `eig` function to find eigenvalues. For Python you may use `numpy.linalg.eig`.

   (b) [1p] Using sampling interval $h = 0.1\,\mathrm{s}$, find the matrices $A$, $B$ and $C$, up to 4 decimal places, for the discrete-time model

$$x(k+1) = Ax(k) + Bu(k),$$
$$y(k) = Cx(k).$$

Find the eigenvalues of $A$. Is the uncontrolled discrete system stable? Motivate your answer.

**Hint:** See Lemma 2.1 from the Lecture notes. You may use the `expm` function to compute matrix exponent (`scipy.linalg.expm` in Python).

(c) [1p] In the case of delay of $0.8h$ seconds, where $h$ is the sampling interval from above, calculate $A_a$, $B_a$ and $C_a$, up to 4 decimal places, in the following augmented system

$$\begin{aligned} \zeta(k+1) &= A_a\zeta(k) + B_a u(k) \\ y(k) &= C_a\zeta(k) \end{aligned}, \quad \zeta(k) = \begin{bmatrix} x(k) \\ u(k-1) \end{bmatrix}. \tag{1}$$

Find the eigenvalues of $A_a$ and compare to those of $A$. What do you observe?

**Hint:** See Lemma 2.2 from the Lecture notes.

2. [0p] **Getting familiar with the cart-pole simulator**

A class of a cart-pole simulator exists in the folder "helper_funcs". The folder also includes the file "test_cartpole_open_loop.m" and "test_cartpole_open_loop.py" which simulates the system with a dummy control input in Matlab and Python, respectively.

Here is a brief description of some useful commands:

**Matlab**

```
addpath('../helper_funcs'); % add path to helper_funcs, if needed
plant = cartpole(); % creates a cart-pole object
plant.animate = true; % enables animation
plant.animationFig = figure('WindowState','maximized'); % figure for the animation
plant.pauseTime = 0.005; % [s] pause time between animation frames
plant.x = [0 0 0 0]'; % set the plant's initial state
plant.simulate(0, 0.025); % simulate with control u=0 for sample time h=0.025s
```

**Python**

```
import sys
sys.path.append('../helper_funcs') # add path to helper_funcs, if needed
from cartpole import CartPole
import numpy as np
plant = CartPole(animate=True) # creates a cart-pole object and enables animations
fig = plant.set_animation_figure() # figure for the animation
plant.pauseTime = 0.005 # [s] pause time between animation frames
plant.x = np.array([0.0, 0.0, 0.0, 0.0]) # set the plant's initial state
plant.simulate(0, 0.025) # simulate with control u=0 for sample time h=0.025s
```

**Note**. If no figure is provided in `plant.animationFig`, then the simulation will run without producing an animation.

We need to make the content of the "helper_funcs" folder available to the current working session in Matlab or Python. One way to achieve this is to add path to "helper_funcs" in your working session, see first line in the code above. The notation "../" means one folder up from the current working directory. If you are unsure of the working directory, then consider adding the absolute path to helper_funcs, or copy the cartpole class file to the same directory where the home assignment is executed.

3. [1p] **Linear-quadratic control**

In the rest of the assignment, we continue with the discretized, linearized cart-pole system

$$x(k+1) = Ax(k) + Bu(k)$$

using a sampling interval $h = 0.025\,\text{s}$. The corresponding system matrices are

$$A = \begin{bmatrix} 1 & 0.025 & 0.0002 & 0.0000 \\ 0 & 1 & 0.0180 & 0.0002 \\ 0 & 0 & 1.0049 & 0.0250 \\ 0 & 0 & 0.3954 & 1.0049 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0003 \\ 0.0244 \\ 0.0005 \\ 0.0366 \end{bmatrix}. \tag{2}$$

We will develop a linear-quadratic regulator (LQR) for the cart-pole system. The details on how to design an LQR will be studied later in the course. Here we will use a ready function to develop the controller, and we provide only a brief overview of its operation.

The LQR is the optimal controller that solves the infinite-horizon LQ problem

$$\begin{aligned} &\min_{u(0:\infty)} \sum_{k=0}^{\infty} x(k)^\top Q x(k) + u(k)^\top R u(k) \\ &\text{s.t.} \quad x(k+1) = Ax(k) + Bu(k) \end{aligned} \tag{3}$$

where $Q$ and $R$ are penalties on the states and control actions, respectively. The optimal solution of the LQ problem is the feedback policy $u(k) = Kx(k)$.

We will now develop an LQR for the cart-pole system by using the initial state and penalties

$$Q = I_4, \quad R = 1 \tag{4}$$

where $I_4$ is a $4 \times 4$ identity matrix. Provide the optimal feedback gain $K$.

Simulate the obtained closed-loop systems for 10 s (400 time steps) starting from the initial condition

$$x(0) = \begin{bmatrix} -0.5\,\text{m} & 0 & \frac{\pi}{12}\,\text{rad} & 0 \end{bmatrix}^\top. \tag{5}$$

At each time instant $t$, apply the optimal control $u^*(t) = Kx(t)$ to the nonlinear plant,

$$x(t+h) = \text{plant.simulate}(u^*(t), h)$$

for a period of $h$ seconds. Provide a plot of the states and control trajectories. What do you observe?

**Hint**. Check the function `dlqr` and run it as $K = -\text{dlqr}(A, B, Q, R)$ to obtain the optimal feedback gain $K$. Notice the minus sign. The `dlqr`/`control.dlqr` function, in Matlab / Python, defines the feedback policy as $u(t) = -\tilde{K}x(t)$, while here we follow the notion used throughout the Lecture Notes, with the policy defined as $u(k) = Kx(k)$.

4. **Steady-state targets**

In this question we investigate different set points for the output, i.e., the cart position $s$ and the pole angle $\theta$. Notice that the LQR is designed to steer the system to the origin, rather than to a particular set point. Hence, the goal is to perform a variable change $\delta x = x - x_s$, $\delta u = u - u_s$, where $x_s$ and $u_s$ are steady-state targets, such that the new variables $\delta x$ and $\delta u$ are steered to the origin.

Calculate the state and input steady-state targets $(x_s, u_s)$ corresponding to the output set-point $y_{sp}$ for the cases below. Can $y_{sp}$ be tracked? Answer this question for each of the cases and motivate your answer.

(a) [1p] $y_{sp} = \begin{bmatrix} s_{sp} & \theta_{sp} \end{bmatrix}^\top$, $s_{sp} = -1\,\text{m}$, $\theta_{sp} = \frac{\pi}{12}\,\text{rad}$.

(b) [1p] $y_{sp} = \theta_{sp} = \frac{\pi}{12}\,\text{rad}$.

(c) [1p] $y_{sp} = s_{sp} = -1\,\text{m}$.

**Hint**. A useful command is `cond` to check the condition number of a matrix before doing an inversion.

5. [1p] **Set-point tracking**

Apply the LQR developed in Question 3 to track the output set-point $y_{sp} = s_{sp} = -1\,\text{m}$. Simulate the obtained closed-loop systems for 10 s starting from the initial condition (5) and provide a plot of the states $x(t)$ and control $u(t)$ trajectories.