Problem Set 1

Statistical Contraction Theory for Learning-based Control of Nonlinear Systems

- You are welcome to collaborate, but you must write up your own solutions.
- You can use any programming language, but you must add comments to each section of the codes to clearly explain your intent.
- If you use any resources other than your brain and lecture notes to solve the problems, you must cite them.
- In the following, $\|\cdot\|$ denotes the Euclidean norm for vectors and the induced 2-norm for matrices, i.e., $\|\cdot\| = \|\cdot\|_2$, unless otherwise noted.

1 Controller Design with Contraction Theory

Consider the following nonlinear dynamical system:

$$\dot{x} = f(x,t) + B(x,t)u \tag{1}$$

where t is time, $x \in \mathbb{R}^n$ is the state, and $u \in \mathbb{R}^m$ is the control input. Here, we design the controller u using the following differential feedback with a positive definite matrix function $M(x,t) \succ 0$:

$$\delta u = -K(x,t)\delta x \tag{2}$$

where $K(x,t) = R(x,t)^{-1}B(x,t)^\top M(x,t)$ is the control gain with the state- and time-dependent weight matrix $R(x,t)$. Suppose that the following conditions hold for $\alpha > 0$:

$$\begin{aligned} &\frac{\partial M}{\partial t} + \sum_{k=1}^{n} \frac{\partial M}{\partial x_k} f_k + M\frac{\partial f}{\partial x} + \frac{\partial f}{\partial x}^\top M - 2MBR^{-1}B^\top M \preceq -2\alpha M \\ &\sum_{k=1}^{n} \frac{\partial M}{\partial x_k} b_{i,k} + M\frac{\partial b_i}{\partial x} + \frac{\partial b_i}{\partial x}^\top M = 0,\ \forall i = 1, \cdots, m \end{aligned} \tag{3}$$

where x_k is the kth element of x, $f_k(x,t)$ is the kth element of $f(x,t)$, $b_{i,k}(x,t)$ is the kth element of $b_i(x,t)$, and the arguments are omitted for simplicity.

1. Let $V(x,\delta x,t) = \delta x^\top M(x,t)\delta x$. Show that

$$\dot{V}(x,\delta x,t) \leq -2\alpha V(x,\delta x,t). \tag{4}$$

2. Let $\underline{m} > 0$ be defined via $M \succeq \underline{m}\mathbb{I}$ with $\mathbb{I}$ being the identity matrix. Derive an upper bound of $\|x_0(t) - x_1(t)\|$ using (4), where x_0 and x_1 are arbitrary solution trajectories of (1) with the feedback policy (2).

2 Robustness and Convexity

This problem is optional. Consider the following perturbed nonlinear dynamical system:

$$\dot{x} = f(x,t) + B(x,t)(u + d(x,t))$$

where t is time, $x \in \mathbb{R}^n$ is the state, $u \in \mathbb{R}^m$ is the control input, and $B(x,t)d(x,t)$ represents external disturbance with $\sup_{x,t}\|B(x,t)\| \leq \bar{b}$ and $\sup_{x,t}\|d(x,t)\| \leq \bar{d}$. Suppose again that the controller u is designed using (2) with the conditions (3).

1. Let $\underline{m} > 0$ and $\overline{m} > 0$ be defined via $\underline{m}\mathbb{I} \preceq M \preceq \overline{m}\mathbb{I}$ with $\mathbb{I}$ being the identity matrix. Derive the upper bound of $\|x_0(t) - x_1(t)\|$, where x_1 is an arbitrary perturbed solution trajectory of (1) with the feedback policy (2) and x_0 is an arbitrary unperturbed solution trajectory of (1) with the feedback policy (2) and with $d(x,t) = 0$.

2. Show that (3) can be convexified into the following conditions:

$$-\frac{\partial W}{\partial t} - \sum_{k=1}^{n} \frac{\partial W}{\partial x_k} f_k + \frac{\partial f}{\partial x} W + W \frac{\partial f}{\partial x}^\top - 2BR^{-1}B^\top \preceq -2\alpha W$$
$$-\sum_{k=1}^{n} \frac{\partial W}{\partial x_k} b_{i,k} + \frac{\partial b_i}{\partial x} W + W \frac{\partial b_i}{\partial x}^\top = 0, \ \forall i = 1, \cdots, m$$

   where $W = M^{-1}$.

3 Toy Problems for Coding in Python

3.1 Neural Networks

Consider the following dynamics of a spacecraft orbiting around an unknown astronomical body:

$$\dot{\text{æ}}_c = \frac{d}{dt}\begin{bmatrix} r \\ v \\ h \\ \Omega \\ i \\ \theta \end{bmatrix} = \begin{bmatrix} v \\ -\frac{\mu}{r^2} + \frac{h^2}{r^3} \\ 0 \\ 0 \\ 0 \\ \frac{h}{r^2} \end{bmatrix} + \begin{bmatrix} 0 \\ \zeta_v(\text{æ}_c) \\ \zeta_h(\text{æ}_c) \\ \zeta_\Omega(\text{æ}_c) \\ \zeta_i(\text{æ}_c) \\ \zeta_\theta(\text{æ}_c) \end{bmatrix} \tag{5}$$

where $\zeta(\text{æ}_c) = [\zeta_v(\text{æ}_c), \zeta_h(\text{æ}_c), \zeta_\Omega(\text{æ}_c), \zeta_i(\text{æ}_c), \zeta_\theta(\text{æ}_c)]^\top$ represents the uncertain part of the dynamics (and the definitions of all the symbols are given in Table 1 if interested).

In this problem, you are welcome to write your own code if you want, but you can also use my template `template_neural_net.py` in `python_files` if you are new to neural networks. You will use the trained neural network later for designing a nonlinear estimator to explore this unknown astronomical body *with rigorous theoretical guarantees*.

1. Install PyTorch.

2. The data files in `python_files` contain `input_data.npy` and `output_data.npy`:

   `input_data.npy` = `(11240,7) numpy array` with 11240 data points of $[r, v, h, \Omega, i, \cos(\theta), \sin(\theta)] \in \mathbb{R}^{1\times 7}$
   `output_data.npy` = `(11240,5) numpy array` with 11240 data points of $\zeta(\text{æ}_c)^\top \in \mathbb{R}^{1\times 5}$.

   When training a neural network, one of the most important steps is to normalize the data to have the same scales for all the inputs and outputs for stability in learning. To this end, divide each column of `input_data.npy` and `output_data.npy` by the element with the largest absolute value in each column. All the normalized elements will have values from -1 to 1.

3. Use the normalized data and train the neural network that approximates $\zeta(\text{æ}_c)$. In particular, use the stochastic gradient descent or its variations until the test error for the normalized data computed by

   ```
   torch.mean( torch.abs( normalized output data - your learned output ) )
   ```

   (i.e., `lossT` in `template_neural_net.py`) is below a certain threshold. (Hint: I would recommend setting the threshold to a value less than 0.0005. You may have to be patient depending on your PC's spec.)

3.2 Numerical Integration

1. Download the `unknown_planet` zip file.

(a) `unknown_planet.py` contains several helper functions with an example code for numerically integrating the dynamics (1) of Problem 4 in Problem Set 2. It also contains a function for numerically computing the Jacobian of a function, `getdfdX`.

(b) The `data` directory contains

- the trained neural net hyperparameters (`net_data.pth`, `input_maxes.npy`, and `output_maxes.npy`) and
- the true state trajectory and measurement data (`Xtrue.npy` and `ytrue.npy`), where the measurement y is given by

$$y = H\text{æ}_c = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r \\ v \\ h \\ \Omega \\ i \\ \theta \end{bmatrix} = \begin{bmatrix} r \\ \Omega \\ i \\ \theta \end{bmatrix}.$$

- In particular, `net_data.pth`, `input_maxes.npy`, and `output_maxes.npy` are the hyperparameters (you are welcome to use your own data here) and

`Xtrue.npy` = `(5620,6) array` for the time history of the true state $\text{æ}_c^\top = [r, v, h, \Omega, i, \theta] \in \mathbb{R}^{1\times 6}$

$$= \begin{bmatrix} r(0) & v(0) & h(0) & \Omega(0) & i(0) & \theta(0) \\ r(\Delta t) & v(\Delta t) & h(\Delta t) & \Omega(\Delta t) & i(\Delta t) & \theta(\Delta t) \\ r(2\Delta t) & v(2\Delta t) & h(2\Delta t) & \Omega(2\Delta t) & i(2\Delta t) & \theta(2\Delta t) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r(k\Delta t) & v(k\Delta t) & h(k\Delta t) & \Omega(k\Delta t) & i(k\Delta t) & \theta(k\Delta t) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r(5619\Delta t) & v(5619\Delta t) & h(5619\Delta t) & \Omega(5619\Delta t) & i(5619\Delta t) & \theta(5619\Delta t) \end{bmatrix}$$

`ytrue.npy` = `(5619,4) array` for the time history of the measurement $y^\top \in \mathbb{R}^{1\times 4}$

$$= \begin{bmatrix} y(0)^\top \\ y(\Delta t)^\top \\ y(2\Delta t)^\top \\ \vdots \\ y(k\Delta t)^\top \\ \vdots \\ y(5618\Delta t)^\top \end{bmatrix}$$

where $\Delta t = 0.001$.

2. Review each function in `unknown_planet.py` and ensure that the code runs. You should get the following three figures of the time evolution of the spacecraft's horizontal and vertical positions, $(p_x, p_y) = (r\cos\theta, r\sin\theta)$:
   (a) (p_x, p_y) of the nominal dynamics, i.e., (1) of Problem 4 in Problem Set 2 with $\zeta = 0$,
   (b) (p_x, p_y) computed by the measurement data `ytrue.npy`, and
   (c) (p_x, p_y) of the learned dynamics, i.e., (1) of Problem 4 in Problem Set 2 with ζ replaced by your neural network approximation ζ_{nn}.
3. Install the Python Control Systems Library and familiarize yourself with the `control.lqr` function.
4. You can use `unknown_planet` in your homework if you want.

Table 1: Summary of the symbols in (5)

Symbol	Description
æ_c	State of spacecraft, $\text{æ}_c = [r, v, h, \Omega, i, \theta]^\top$
r	Radial distance from center of target body
v	Radial velocity
h	Specific angular momentum
Ω	Right ascension of ascending node
i	Inclination of orbit
θ	True anomaly
μ	Gravitational constant of target body