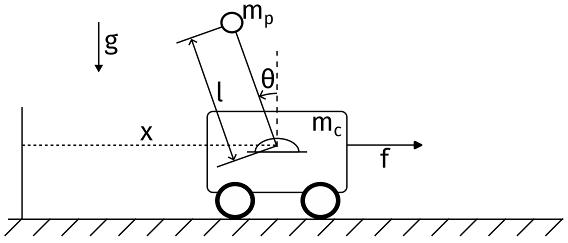
ME/ECE/MATH 497/597: Optimization Theory and Practice Fall 2023 | Project II: Unconstrained Optimization

In this project, we will find a neural network that stabilizes the upright (unstable) equilibrium of the cart-pole (inverted pendulum) system, depicted in the figure below.

The figure shows our parametrization of the system. x is the horizontal position of the cart, θ is the counter-clockwise angle of the pendulum (zero is the upright position). We will use $\boldsymbol{q} = \begin{bmatrix} x & \theta \end{bmatrix}^\top$, and $\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} & \dot{\boldsymbol{q}} \end{bmatrix}^\top$. The task is to stabilize the unstable fixed point $\boldsymbol{x} = \boldsymbol{0}$.



The Cart-Pole system.

1. [50 points] We will first learn how to design an optimal linear controller for a generic linear time-invariant (LTI) control system. In order to derive the form of the optimal linear controller, let us consider a generic LTI system in state-space form.

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \tag{1}$$

where $\boldsymbol{x}$ is an n-vector, depicting the system state, $\boldsymbol{u}$ is an m-vector, depicting the control input, $\boldsymbol{A}$ is the state-matrix and $\boldsymbol{B}$ is the input matrix. Consider the infinite-horizon cost function given by

$$J = \int_0^\infty \left(\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^\top \boldsymbol{R}\boldsymbol{u}\right) \mathrm{d}t, \quad \boldsymbol{Q} = \boldsymbol{Q}^\top \succeq \boldsymbol{0}, \quad \boldsymbol{R} = \boldsymbol{R}^\top \succ \boldsymbol{0}.$$

Our goal is to find the optimal cost-to-go function $J^*(\boldsymbol{x})$ which satisfies the Hamilton-Jacobi-Bellman equation:

$$\min_{\boldsymbol{u}} \left[\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^\top \boldsymbol{R}\boldsymbol{u} + \frac{\partial J^*}{\partial \boldsymbol{x}} (\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u})\right] = 0, \qquad \forall \boldsymbol{x}. \tag{2}$$

It is well-known that for linear control systems such as (1), the optimal cost-to-go function is quadratic in the states:

$$J^*(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{S}\boldsymbol{x}, \quad \boldsymbol{S} = \boldsymbol{S}^\top \succeq 0. \tag{3}$$

(a) [20 points] Assuming the form given in (3) for the optimal cost-to-go function, find the optimal policy $\boldsymbol{u}^* = \pi^*(\boldsymbol{x})$ as a function of $\boldsymbol{R}$, $\boldsymbol{B}$, $\boldsymbol{S}$, and $\boldsymbol{x}$.

(b) [20 points] Substitute the form of the optimal policy you found in part (a) into the Hamilton-Jacobi-Bellman equation (2) to derive an equation that needs to be satisfied by $\boldsymbol{S}$.

(c) [10 points] Solve the equation you derived in part (b) for the following system:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \boldsymbol{Q} = \boldsymbol{I}, \quad \boldsymbol{R} = {}^1\!/_2.$$

Then, find the optimal controller for this system: congratulations, you have just derived an optimal controller for the double-integrator system!

2. [50 points] Using the Lagrangian formulation, the equations of motion of the cart-pole system may be derived to obtain

$$\begin{aligned}
&\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \boldsymbol{\tau}_g(\boldsymbol{q}) + \tilde{\boldsymbol{B}}\boldsymbol{u},\\
&\boldsymbol{M}(\boldsymbol{q}) = \begin{bmatrix} m_c + m_p & -m_p\ell\cos\theta \\ -m_p\ell\cos\theta & m_p\ell^2 \end{bmatrix},\quad \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}}) = \begin{bmatrix} 0 & m_p\ell\sin(\theta)\dot{\theta} \\ 0 & 0 \end{bmatrix},\\
&\boldsymbol{\tau}_g(\boldsymbol{q}) = \begin{bmatrix} 0 \\ m_p g\ell\sin\theta \end{bmatrix},\quad \tilde{\boldsymbol{B}} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},
\end{aligned} \tag{4}$$

where, we have identified $\boldsymbol{u} = f$, the external (control) force applied on the cart. Suppose that the particular system we are interested in has $m_c = 0.2$kg, $m_p = 0.5$kg, $\ell = 0.5$m. At Boise State University, the gravitational acceleration is $g = 9.80364\,\mathrm{m\,s^{-2}}$.

This is a nonlinear system. Let us linearize it around the operating point, i.e., the upright equilibrium point $\boldsymbol{x} = \boldsymbol{0}$.

$$\begin{bmatrix} m_c + m_p & -m_p\ell \\ -m_p\ell & m_p\ell^2 \end{bmatrix}\begin{bmatrix} \ddot{x} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 \\ m_p g\ell\theta \end{bmatrix} + \begin{bmatrix} f \\ 0 \end{bmatrix}.$$

This is a linear time-invariant system, which may be put into the standard form

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{m_p g}{m_c} & 0 & 0 \\ 0 & \frac{(m_c+m_p)g}{m_c\ell} & 0 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{1}{m_c} \\ \frac{1}{m_c\ell} \end{bmatrix} f = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u} \tag{5}$$

(a) [20 points] Using the results of question 1 and the linearized equations of motion of the cart-pole system in (5), find the optimal linear controller. Deduce the $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices using the loss function in equation (6).
Hint. Use Python Control System Library's `LQR` function. Then verify that the solution this `LQR` function outputs satisfies the equation you derived in question 1 part (b).

(b) [30 points] In this part, we will find a stabilizing controller $\boldsymbol{u}$ as a neural network using data-driven techniques. We do this using the following steps.

1. Postulate that the controller has the form $f = \boldsymbol{W}^{(1)}\sigma(\boldsymbol{W}^{(0)}\boldsymbol{x})$, for some unknown weights $\boldsymbol{w} = \{w_{ij}^k\}$, $k = 0, 1$. Here, σ is an activation function to be determined. Choose from the list provided in PyTorch's relevant documentation page.
2. Simulate the nonlinear system (4) starting from initial states near the upright equilibrium $\boldsymbol{x} = \boldsymbol{0}$.
3. The resulting trajectory will incur a cost at each state $\boldsymbol{x}$ according to how much

farther this state is from the upright equilibrium:

$$\begin{aligned} l(\boldsymbol{x}) &= \frac{1}{20}x^2 + m_p g\ell(1-\cos\theta) + \frac{1}{2}(m_c+m_p)\dot{x}^2 + m_p\ell\cos(\theta)\dot{x}\dot{\theta} + \frac{1}{2}m_p\ell^2\dot{\theta}^2 + \frac{1}{10}f^2 \\ &\approx \frac{1}{20}x^2 + m_p g\ell\theta^2 + \frac{1}{2}(m_c+m_p)\dot{x}^2 + m_p\ell\cos(\theta)\dot{x}\dot{\theta} + \frac{1}{2}m_p\ell^2\dot{\theta}^2 + \frac{1}{10}f^2. \end{aligned} \tag{6}$$

4. Each time we sample the trajectory $\phi(t;\boldsymbol{x}_0,\boldsymbol{w})$ of the nonlinear system, for a particular state $\boldsymbol{x}$, we compute this running cost and add them up to yield the cost function
$$g(\phi) = \sum_{\text{sample time}} l(\boldsymbol{x}).$$
5. Finally, we solve the unconstrained optimization problem, that minimizes $g(\phi)$ by changing the controller weights $\boldsymbol{w}$.

I have provided a PyTorch code for you to use to perform this optimization. Utilizing that code and modifying as necessary, find weights $\boldsymbol{w}$ that stabilizes the upright equilibrium point. Compare the results to the optimal controller you found in part (a).

- Try `ReLU`, `LeakyReLU`, `Tanh`, `Sigmoid`, `ELU`, `SiLU` as activation functions. Which of these work well, which ones do not? Discuss why this might be.
- Experiment with the number of hidden layers and the number of neurons used in each of these hidden layers. What network topology works better than others? Discuss why this might be.
- Perform Monte Carlo simulations to determine which controllers performs better with respect to the loss function in equation (6).

A Setting up the Python/PyTorch Environment

A.1 Install Mamba

Please follow the instructions in the following link for your operating system.
https://github.com/conda-forge/miniforge#mambaforge

After installing Mamba, it is time to create an *environment*.

A.2 Conda (Virtual) Environments

Virtual environments are a convenient way to isolate Python installations associated with different projects. An *environment* is a replication of Python itself and some (or all) of its libraries. With so many independently developed Python libraries, each having many different versions and each version having various dependencies (on other libraries), things can get out of hand really quickly.

Remark 1. *You should create a different environment for every project you start working on.*

In order to create an environment, choose a **name** for your environment, say `opt-proj-2`. Then, you need to open a *terminal* (in Linux) or an *Conda Prompt* (in Windows or macOS) and type the following command.

```
$ mamba create --name opt-proj-2 python=3.11.6
```

The command above creates a Mamba (Conda) environment named `opt-proj-2` and includes all Conda packages in it, as well as Python version `3.11.6`.

After it finishes creating the environment, it is time to *activate it*, meaning, making that Python installation the one to be used now. In the same terminal (or Conda prompt), type:

```
$ mamba activate opt-proj-2
```

Now, your command prompt should show the environment you are in. You will need to *activate it* every time you open a new terminal.

A.3 Install Dependencies

We will use a few libraries that facilitate development immensely. These libraries are:

1. `numpy`: Array and matrix library,
2. `matplotlib`: Plotting library,
3. `python-control`: Python control systems library,
4. `slycot`: Subroutine library in systems and control theory,
5. `jupyter`: Web-based interactive computing platform.

You can install all of these by using the following command.

```
$ mamba install matplotlib numpy control slycot jupyter
```

A.4 PyTorch

PyTorch is a *deep learning framework*. It is an "open source machine learning framework that accelerates the path from research prototyping to production deployment."

We can go straight to the *Start Locally* https://pytorch.org/get-started/locally/ section of PyTorch's websites, and it will automatically select the options that best suit your local environment, and it will show you the command to run.

Here, we want to select the following.

- PyTorch Build: Always select a *Stable* version.
- Package: *Conda*
- Language: *Python*
- Compute Platform: *CPU*

Copy and paste the command given in this page, and run it after replacing `conda` with `mamba`. After the installation is complete, further install `TensorBoard` by typing

```
$ mamba install tensorboard
```

A.5 Jupyter

After activating your environment and navigating into the main folder, you should now be able to run

```
$ jupyter lab --no-browser --port=8888
```

This will start up `Jupyter Lab`, which you can then access by running your browser and navigating to the address printed on your terminal. Open up the file `main_blank.ipynb` and start working on the project. Enjoy!