IT3105 – Project 1 - lab report

A General Purpose JAX-based Controller

Group: 11
Students: Kristian Steinhaug Knudsen

Introduction

In this project, we built a general-purpose PID controller that could simulate different controllable systems and apply two types of PID controllers: the traditional three-parameter model and a neural-network-based version.

For each controller and controlled system, our program used Python's JAX module to compute gradients and update the controller parameters using a basic gradient descent or ascent algorithm.

This report presents performance measures as illustrations and details the parameters used for different plants and controllers. It also includes a *full* mathematical description of the third plant."

Table of Contents

Plant 1 – Bathtub 3
Neural Controller 3
Table 1: Params used for bathtub with nn 3
Visualization: 4
Summary bathtub nn: 4
Classical Controller 5
Table 2: Params used for bathtub with cc 5
Visualization: 6
Summary bathtub cc: 7
Plant 2 – Cournot 8
Neural Controller 8
Table 3: Params used for cournot using nn 8
Visualization: 9
Summary cournot nn: 9
Classical Controller 10
Table 4: Params used for cournot using cc 10
Visualization: 11
Summary cournot cc: 12
Plant 3 – Pendulum 13
Neural Controller 13
Table 5: Params used for pendulum using nn 13
Visualization: 14
Summary pendulum nn: 14
Classical Controller 15
Table 6: Params used for pendulum using cc 15
Visualization: 16
Summary pendulum cc: 17
Mathematics of the 3rd plant 18

Plant 1 – Bathtub

Neural Controller

Table 1: Params used for bathtub with nn

Plant	
C	0.0015
A	0.15
Controller	
Layers	12, 4, 2
Activations	ReLu, sigmoid, tanh, none
Init range	0.01 to 0.025
Simulation	
Initial state	0.5
Set point	0.5
Time steps	20
Disturbance	-0.01 to 0.01
Seed	1337
Training	
Epochs	20
Learning rate	0.00001

Visualization:

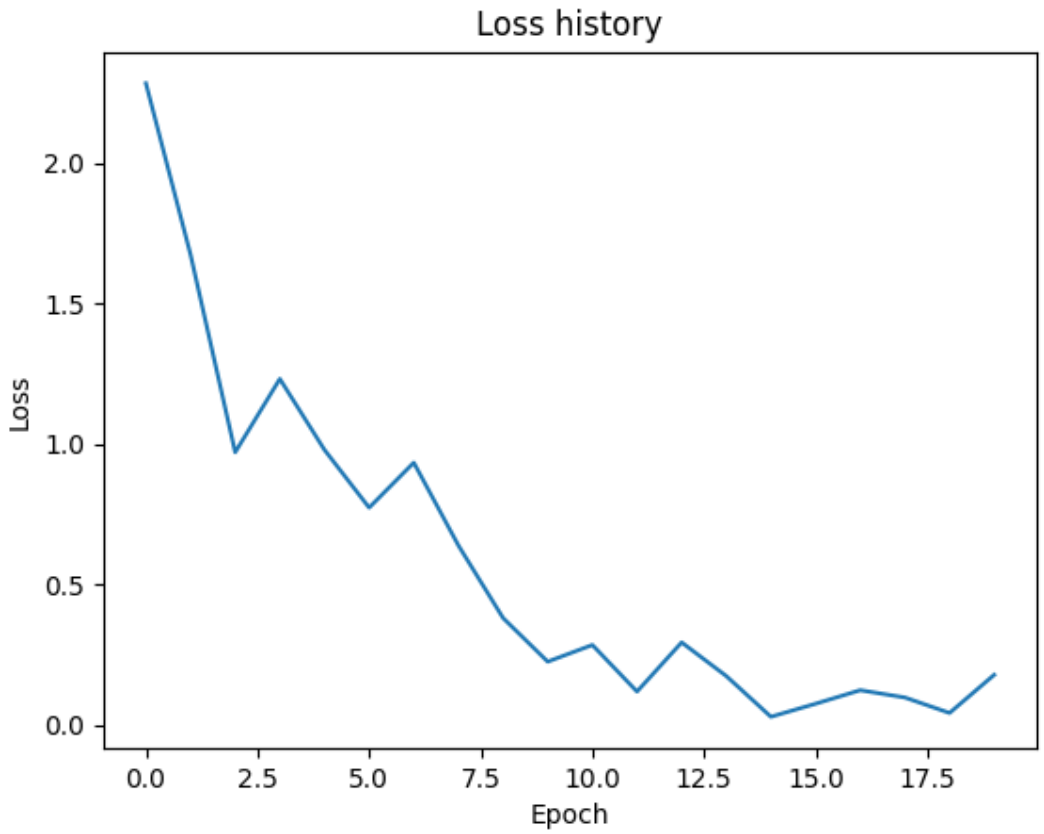

Figure 1 Loss for bathtub plant using nn controller over 20 epochs

Summary bathtub nn:

The result turned out well. We chose to lower the learning rate to demonstrate the learning process however, it seems it could get similar results even with a higher lr and lower epoch range.

The main factor that's holding it back is the disturbance. You can see different bumps, and because of this it will never even out.

Classical Controller

Table 2: Params used for bathtub with cc

Plant	
C	0.0015
A	0.15
Controller	
K_p	0
K_i	0
K_d	0
Simulation	
Initial state	0.5
Set point	0.5
Time steps	20
Disturbance	-0.001 to 0.001
Seed	1337
Training	
Epochs	20
Learning rate	0.00005

Visualization:

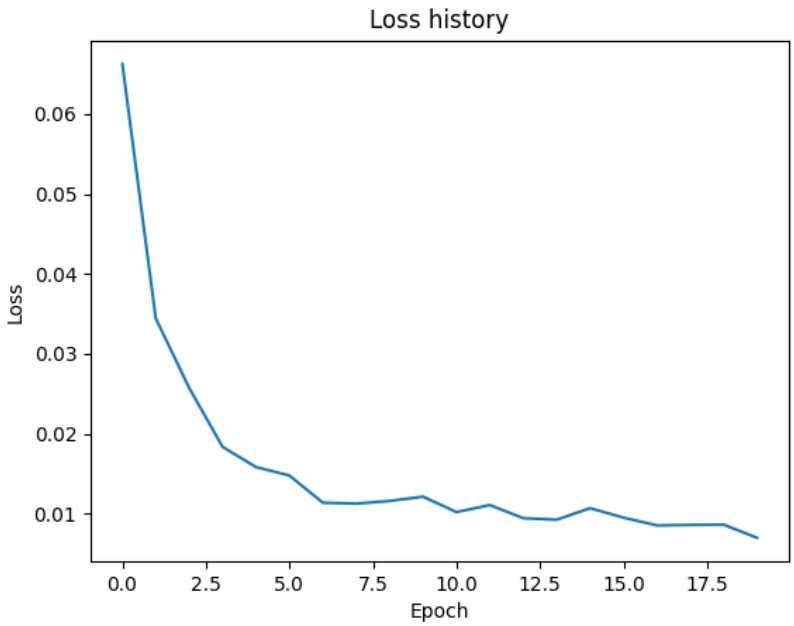

Figure 2 Loss for bathtub plant using cc controller over 20 epochs

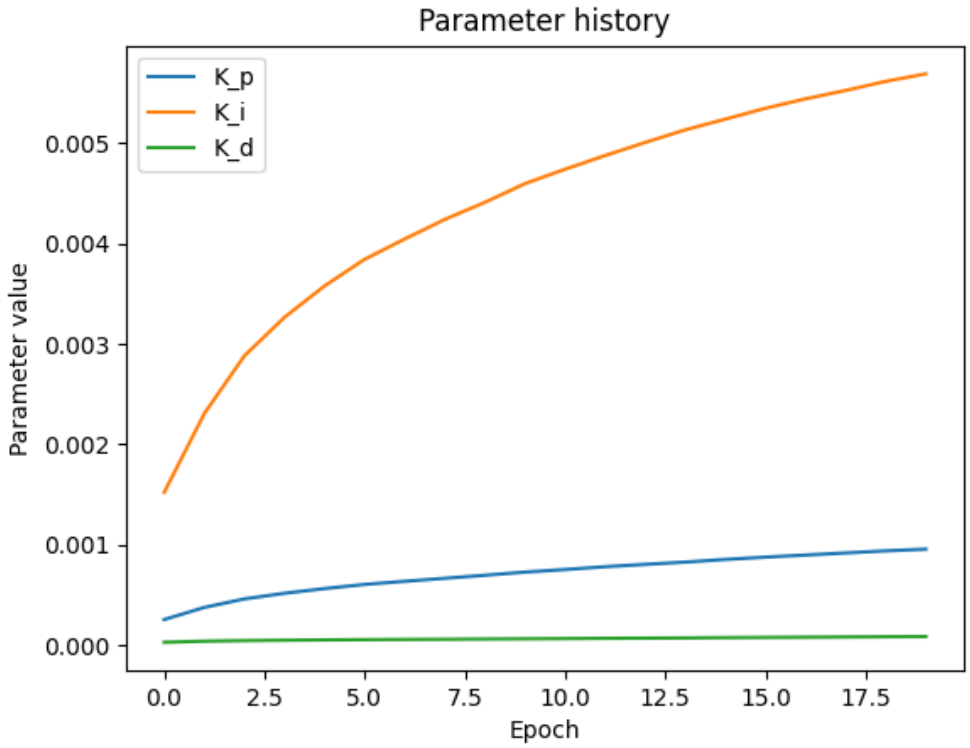

Figure 3 Control parameters change over 20 epochs for bathtub plant using cc

Summary bathtub cc:

For this we lowered the disturbance and increased the learning rate compared to the nn run. As we can see it quickly evens out after 10 epochs, displaying slower learning than the neural net controller.

The control parameters start at 0. We wanted to let the system figure out the behavior by itself. As a consequence, we can see that the integral part raises in dominance. This may be because the integral part on average contributes greater to the error as it the proportional part is not quick enough to catch up.

In a manually tuned system one would probably expect the proportional part to be the highest, but in this case that's not what the system figured out. The performance was satisfactory nonetheless.

Plant 2 – Cournot

Neural Controller

Table 3: Params used for cournot using nn

Plant	
p_{max}	2.0
c_m	0.1
Controller	
Layers	12, 4, 2
Activations	ReLu, sigmoid, tanh, none
Init range	0.01 to 0.025
Simulation	
Initial state	0.5
Set point	0.3
Time steps	20
Disturbance	-0.000001 to 0.000001
Seed	1337
Training	
Epochs	10
Learning rate	0.005

Visualization:

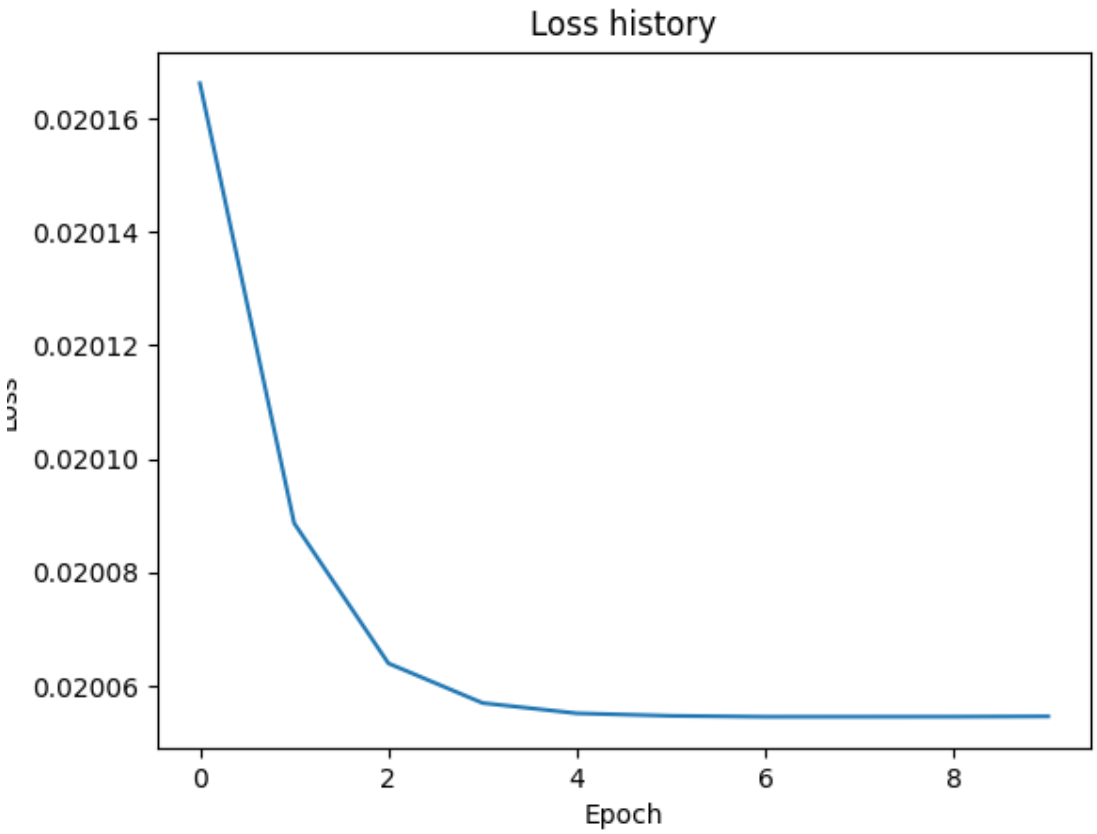

Figure 4 loss for Cournot using Neural net

Summary cournot nn:

We decided to keep the disturbance low because we observed some interesting behaviour. Our current parameters seemed to have a max performance. This is probably due to the parameters we chose for the cournot competition. The neural net model seemed to have found the optimal strategy very fast.

We have not confirmed this by hand and there may be some gradient losses for certain values. Remember we limit certain variables by clipping them at <0 and >1, making the learning process unstable if we ever go well beyond these values.

Classical Controller

Table 4: Params used for cournot using cc

Plant	
p_{max}	2.0
c_m	0.1
Controller	
K_p	0
K_i	0
K_d	0
Simulation	
Initial state	0.5
Set point	0.3
Time steps	20
Disturbance	-0.000001 to 0.000001
Seed	1337
Training	
Epochs	20
Learning rate	0.01

Visualization:

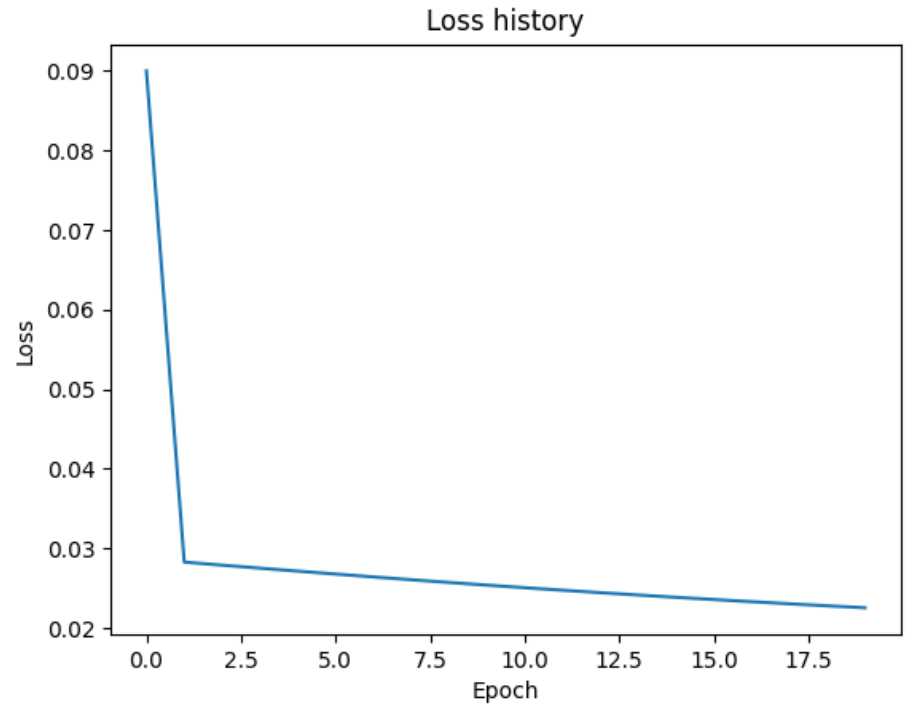

Figure 5 loss history for cournot using nn

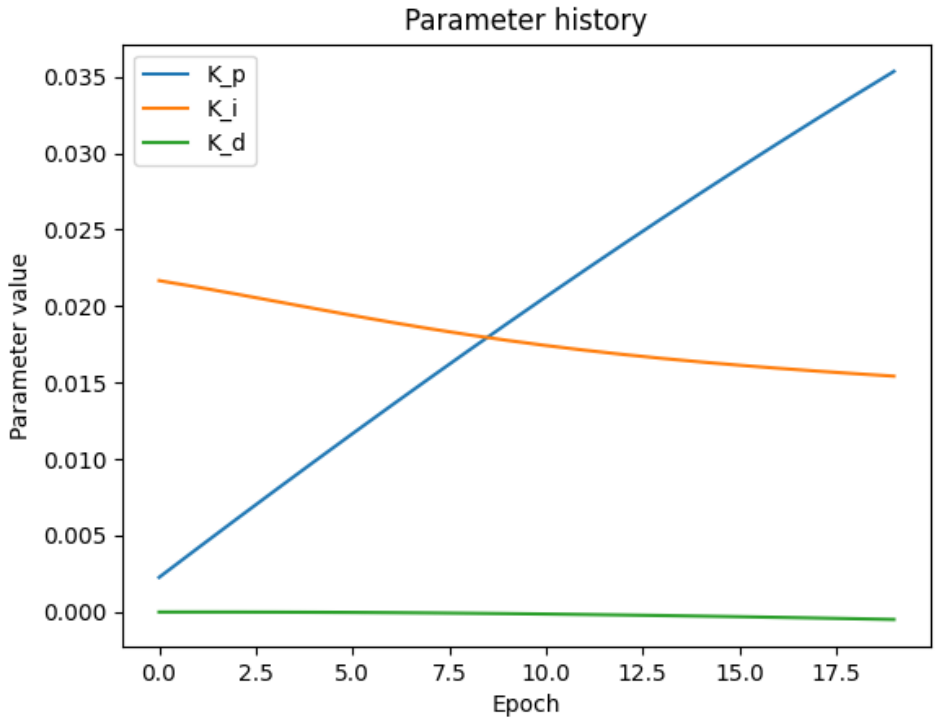

Figure 6 controller parameters for cournot using cc

Summary cournot cc:

The loss graph displays a sharp drop after the first epoch, then gradual decrease over time. Our theory of why this is, is that it learns the behavior of the cournot competition very fast like our neural network controller. However, our PID controller oscillates around the target. Our decrease in loss is due to our program getting better at dampening this effect.

The graphs may not display it clearly, but the derivative variable is negative and its scalar is increasing in the negative direction, while the integral is decreasing. This will cause less overshoot and a better dampening effect.

Plant 3 – Pendulum

Neural Controller

Table 5: Params used for pendulum using nn

Plant	
C_{Drag}	0.5
$Area$	0.1
$mass$	1.0
$Voltage$	12.0
Controller	
Layers	12, 4, 2
Activations	ReLu, sigmoid, tanh, none
Init range	0.01 to 0.025
Simulation	
Initial state	0.5
Set point	0.5
Time steps	20
Disturbance	-0.01 to 001
Seed	1337
Training	
Epochs	20
Learning rate	0.00001

Visualization:

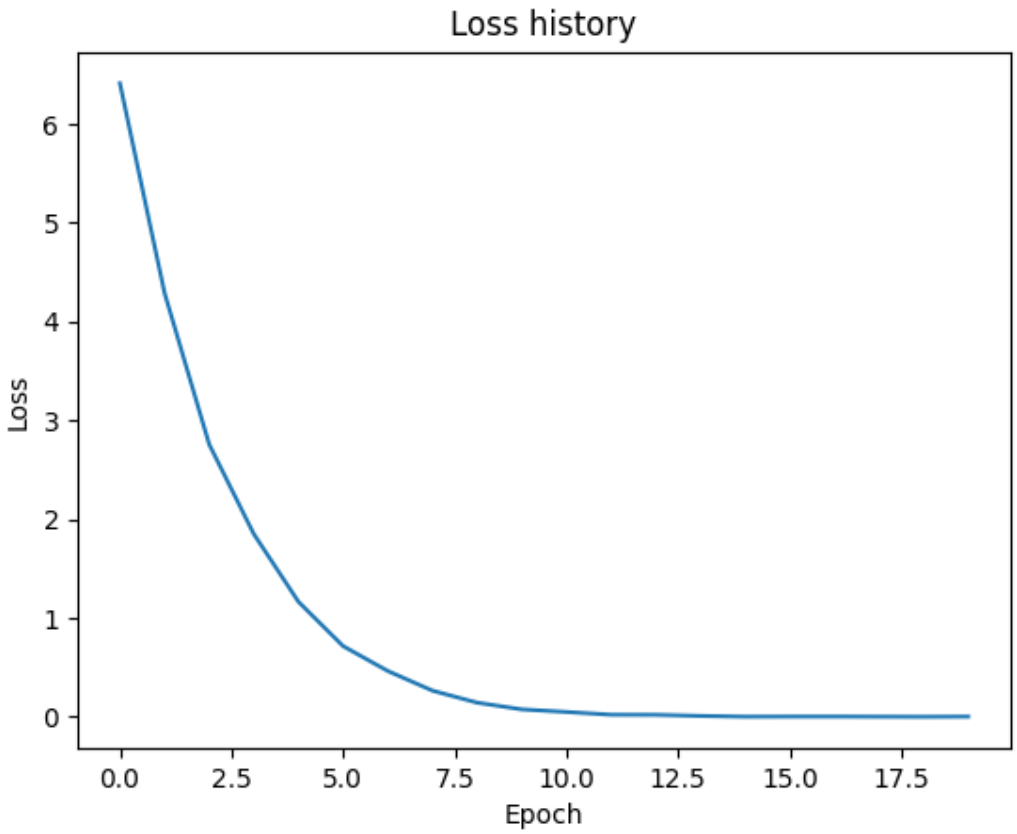

Figure 7 loss over 20 epochs for the pendulum plant using nn controller

Summary pendulum nn:

Not much to say about this. We have optimized the learning rate to have a smooth curve. It seems to level out at 0.002 loss. If we removed the disturbance this would be 0. Our controller is excellent at tuning for this plant.

Classical Controller

Table 6: Params used for pendulum using cc

Plant	
C_{drag}	0.5
$Area$	0.1
$mass$	1.0
$Voltage$	12.0
Controller	
K_p	0
K_i	0
K_d	0
Simulation	
Initial state	0.5
Set point	0.5
Time steps	20
Disturbance	-0.01 to 0.01
Seed	1337
Training	
Epochs	20
Learning rate	0.00005

Visualization:

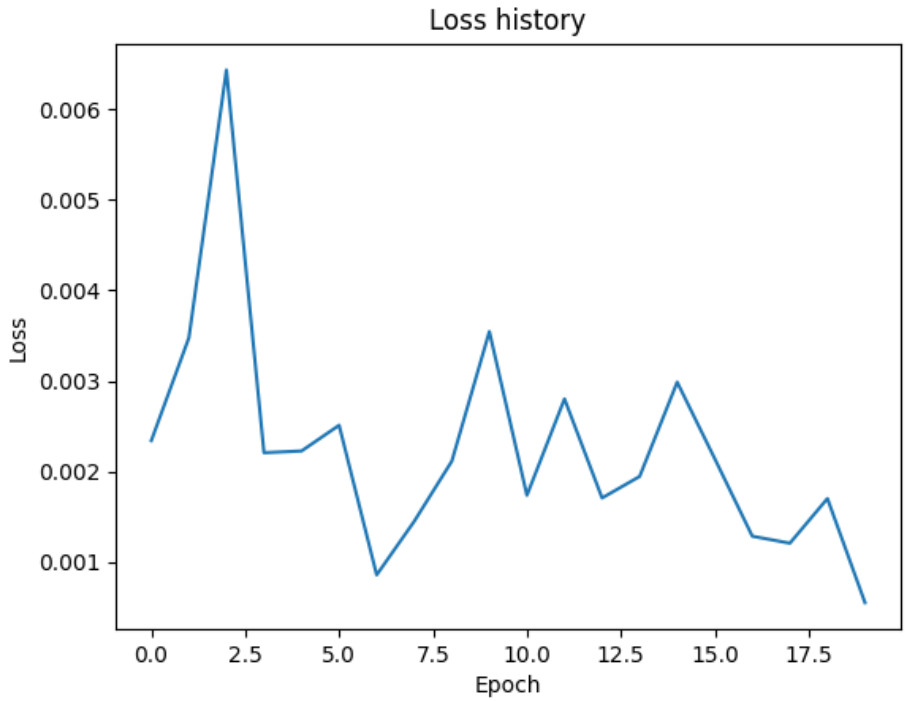

Figure 8 loss over 20 epochs for the pendulum plant using cc

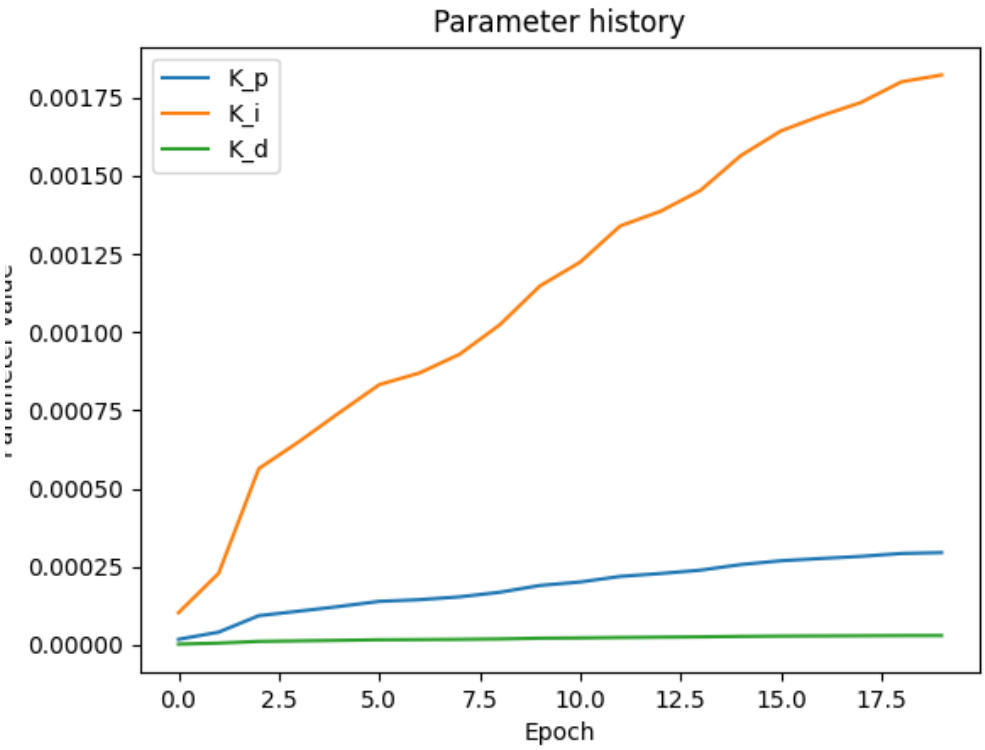

Figure 9 Controller parameters over 20 epochs on the pendulum plant using cc.

Summary pendulum cc:

It does not seem to ever reach a balance and it keeps on learning even after 20 epochs. We suspect this is due to the combination of disturbance and poor initial controller parameters. It keeps on increasing the integral parameter which is a bad sign and will probably cause it to overshoot.

Running it for longer with lower learning rate and higher initial proportional parameter would probably solve it. However, the disturbance will always be a factor that decreases the performance.

Mathematics of the 3rd plant

Suppose we have a pendulum with an electric motor at its base. The electric motor delivers energy according to the formula:

$$E = V\ I\ dt$$

We assume each timestep is 1.0 seconds.
The voltage is a constant, controlled by a hyperparameter, while the current is controlled by the controller.

This leaves us with the control signal:

$$U_{controlsignal} = U = E(I) = V\ I$$

The goal is to simulate a perpetual motion machine using the controller to control the amperage at each timestep.

The system loses energy primarily due to air resistance. The following is a mathematical description of how much energy the pendulum loses at every timestep.

The drag force is given by:

$$F_{drag} = \frac{1}{2}\ C_D\ \rho\ A\ v^2$$

Where:

- C_D is the drag coefficient
- $\rho\ is\ the\ air\ density\ (typically\ 1.225\ kg/m^3)$
- A is the cross-sectional area
- v is the speed relative to the air

To overcome the air resistance, we consider the rate at which work is done:

$$P = F\ v = \frac{W}{t}$$

Thus, the power lost to drag is:

$$P = \frac{1}{2}\ C_D\ \rho\ A\ v^3$$

If we knew the speed exactly, we could simply use the time delta to determine the energy needed. However, we must make estimations.

At the bottom of the pendulum the potential energy is zero and the kinetic energy is at its maximum. We can calculate v_{max} from this.

The total energy is the sum of the potential and kinetic energy:

$$E_{tot} = E_p + E_k$$

At the bottom of the swing, where the potential energy is 0 the total energy becomes purely kinetic:

$$E_{tot} = \frac{1}{2} m\, v_{max}^2$$

Solving for v_{max}:

$$v_{max} = \sqrt{2 * \frac{E_{tot}}{m}}$$

We can use this to find the v_{avg}.

A pendulum is a harmonic oscillator and its position over time is described by:

$$x(t) = A \sin(\omega\, t + \varphi)$$

This only applies if the angle is under 30 degrees.

Differentiate for the velocity function:

$$v(t) = x'(t) = A\, \omega \cos(\omega\, t + \varphi)$$

$$A\, \omega = Amplitude = v_{max}$$

Since cos starts at 0, we don't care about any shifts when finding the average speed over ¼ of an oscillation. So, we're left with:

$$v(t) = v_{max} \cos(\omega\, t)$$

Assuming we start at the beginning of an oscillation, the average speed over one-quarter of the period is equal to the average speed over the whole oscillation:

$$v_{avg} = \frac{1}{T/4} \int_0^{T/4} |v(t)|\, dt$$

Thus:

$$v_{avg} = \frac{2}{\pi}\, v_{max}$$

Plugging this into the formula for air resistance we get:

$$P = \frac{1}{2}\, C_D\, \rho\, A\, v_{avg}^3$$

$$P = \frac{1}{2}\, C_D\, \rho\, A \left(\frac{2}{\pi}\, v_{max}\right)^3$$

$$P = \frac{1}{2}\, C_D\, \rho\, A \left(\frac{2}{\pi} \sqrt{2 * \frac{E_{tot}}{m}}\right)^3 = \frac{W}{t}$$

This means that we lose P energy every second, and since we assume $\Delta t = 1$, the energy loss per timestep is:

$$E_{loss} = \frac{1}{2} C_D \rho A \left(\frac{2}{\pi} \sqrt{2 * \frac{E_{tot}}{m}} \right)^3$$

Including the effects of the controller input and an external disturbance, the new total energy is given by:

$$E_{new} = E_{tot} - E_{loss} + U + D$$

The final equation for the new total energy will be:

$$E_{new} = E_{tot} - \frac{1}{2} C_D \rho A \left(\frac{2}{\pi} \sqrt{2 * \frac{E_{tot}}{m}} \right)^3 + V I + D$$

- E_{tot} *will be given*
- V, C_d, m and A are constant hyperparameters
- ρ *is set at* 1.225..
- I *is the control parameter to be optimized*
- D is the disturbance measured in the same unit as E_{tot}