**Coursework 3: PID Tuning using Reinforcement Learning**

Team Abyss:Zhongqi Zhuang, Elton Lam, John Huang and Nicholas Gerard

*Department of Chemical Engineering, Imperial College London, U.K.*

**1. Disclaimer**

We have spoken directly to the module leader who has confirmed that due to an error in the code given to us, this coursework is better suited as a data-driven optimisation problem instead of a reinforcement learning (RL) problem. As such, much of the work done as part of Coursework 1 Part 2 [1] was repeated here.

**2. Identifying the error**

RL operates on the principle of decision-making based on the outcomes of past actions. In the context of PID tuning using RL, the system aims to optimise controller gain values to minimise error and match setpoints. However, in this problem, the controller gains do not change since the policy that converts weights into the K values is absent from J\_ControlCSTR, transforming the problem into a static data-driven optimisation (DDO) problem. When the policy is moved inside J\_ControlCSTR, **Figure 1** shows the zig-zag response produced for the controlled variables, indicative that the values for the controller gains are changing which is expected from a RL algorithm.

A screenshot of a graph

Description automatically generated

**Figure 1**: Output of RL compatible model

For a DDO problem, 21 weights in the policy were deemed unnecessary as optimisation algorithms such as Jordanla can directly optimise for the seven required Ks more efficiently.

**3. Intuition**

The Jordanla algorithm employed in this coursework explores the solution space cautiously to converge towards an optimum by iteratively improving the location of a polygon by reflection and bisection. Details about the algorithm can be found in Section 2 of the report for Coursework 1 Part 1 [2]. Additionally, SIMC tuning rules were used in this coursework to yield a good starting point for the algorithm; this is explained in the report for Coursework 1 Part 2 [1].

**4. Methodology**

The main methodology behind the algorithm is the same as in Coursework 1 Part 2 [1] and is explained in Section 3 of that report. The additional functions present in this coursework are shown in the pseudocode in Section 6 and are there just to ensure compatibility with scripts PID\_Tuning.py and ML4CE\_evaluate\_PID\_algorithms\_part3.

**5. Performance**

**A graph of a graph

Description automatically generated with medium confidence**

**Figure 2**: Output of RL compatible model

As with Coursework 1 Part 2, the DDO algorithm employed generally leads to good control for most setpoint, as shown in **Figure 2**. It falls short however, for some setpoints, as shown in **Figure 3**, where it takes some time for the manipulated variable to realise control action. This is believed to be due to integral windup [3]; the error accumulates as the setpoint is never reached. When there is a change in setpoint at the ten-minute mark, the controller must first offset the previously accumulated error before control action can begin in the opposite direction. A solution to this would be to implement integral clamping on the PID controller in PID\_Tuning\_Task.py, which is outside the scope of what is allowed in this coursework.

**A graph of different types of temperature

Description automatically generated**

**Figure 3:** Suboptimal control due to integral windup

**6. Pseudocode**

The pseudocode for the additional functions used in this algorithm is shown below. The Pseudocode for Jordanla can be found in Section 5 of the report for Coursework 1 Part 2 [1].

***import*** J\_ControlCSTR ***from*** *PID\_Tuning\_Task.py*

Bound\_values(*uk, bounds*)

***for******each*** *index* ***and*** *j* ***in*** *bounds*

***if*** *uk[index] < lower bound*

*uk[index] ← lower bound*

***else******if*** *uk[index] > upper bound*

*uk[index] ← upper bound*

***return*** *uk*

Map\_weights\_to\_Ks(*w, x, xs, bounds*)

*u1 ← w[0], u2 ← w[1], …, u7 ← w[6]*

*uk ←* Bound\_values(*[u1,…,u7], bounds*)

***return*** *uk*

Run\_ddo\_algo(*problem\_specs, evals*)

*x\_dim ← problem\_specs[‘nu’]*

*bounds ← problem\_specs[‘bounds’]*

Objective\_func(*w*)

*x ← None*

*xs ← None*

*Ks ←*Map\_weights\_to\_Ks(*w, x, xs, bounds*)

*data\_results, z ←* J\_ControlCSTR(*Ks*)***return*** *z*

*best\_x ←* Jordanla(Objective\_func*, x\_dim, bounds, evals*)

***return*** *best\_x*

Algorithm\_Abyss(*problem\_specs,params, data\_res*)

*evals ← params[‘e\_tot’]*

*w\_opt ←* Run\_ddo\_algo(*problem\_specs, evals*)

***return*** *w\_opt,* Map\_weights\_to\_Ks

**7. If this was a reinforcement learning problem**

Initially, the coursework was thought to be a reinforcement learning problem, prompting the exploration of various approaches to address its challenges. Firstly, several modifications were made to naive\_policy\_train, which included the introduction of additional weights, sigmoid activation functions, and bias terms for each controller gain in order to capture non-linear relationships within the system. Sigmoid activation functions were chosen over tanh as the controller gains were always positive due to their bounds.

Additionally, refinements were implemented in Generalised\_policy\_search, incorporating a more sophisticated strategy for selecting and perturbing the weights. Furthermore, a pivotal shift in strategy occurred when it was recognised that model predictive control offered an advantageous solution alignment with the coursework requirements. Model predictive control excels in real-time optimization, which made it a suitable candidate for the problem at hand.

**8. References**

[1] Report for CW1 b

[2] Report for CW1 a

[3]<http://cse.lab.imtlucca.it/~bemporad/teaching/controllodigitale/pdf/Astrom-ACC89.pdf>