Solving For Controller Feedback of Dynamical Systems Using Reinforcement Learning

Ethan D. Crouse
Supervisor: Dr. Steffen W. R. Werner

May 5, 2025

1 Introduction

Consider the following continuous linear dynamical system

$$\begin{aligned} E\dot{x} &= Ax + Bu \\ E &\in \mathbb{R}^{n\times n} \\ A &\in \mathbb{R}^{n\times n} \\ B &\in \mathbb{R}^{m\times p} \end{aligned}$$

x contains the system's state, and u acts as a feedback control of the system and is computed given the current state and returns the control. The continuous dynamical system is asymptotically stable if and only if all of the eigenvalues of $E^{-1}A$ satisfy $\lambda_j < 0$, and stable if the eigenvalues satisfy $\lambda_j \leq 0$ (Largest eigenvalue is equal to 1) for all initial states x_0

If any eigenvalues of $E^{-1}A$ are greater than 0, the system is unstable for all initial states x_0 and will grow unbounded.

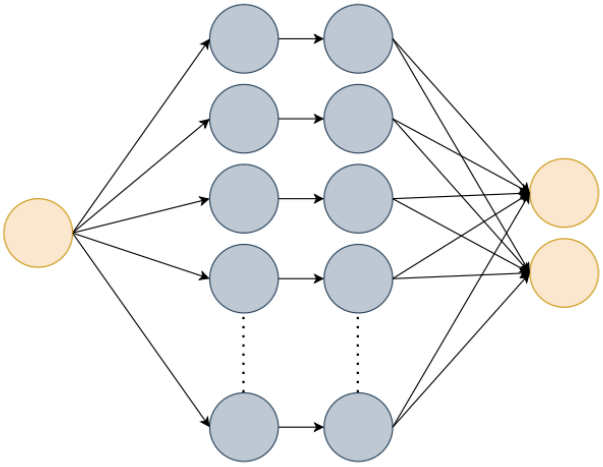
In the real world, unstable continuous dynamic systems are common. Finding their stability is extremely important across many applications because unstable systems can often lead to failure, damage, or danger if not properly managed. By designing methods to stabilize them, we can make otherwise dangerous, volatile, or inefficient systems safe, reliable, and useful.

2 Project Overview

The idea of this research project was to write a program in MATLAB utilizing the Reinforcement Learning package that can take input matrices A, B, and E that form the dynamical system $Ex(k+1) = Ax(k)+Bu(k)$. Where $x(k)$ is the system's state at k and u is the feedback control for the current state.

A feedback control is a mechanism by which a system continuously measures its state and uses that information to adjust its inputs to drive the system toward a desired behavior. In our case, we want to drive to a steady state and stay there. Concretely, at each time step (or instant in continuous time), the controller observes the current state x and solves for the control feedback $u = K(x)$

The program will train a neural network using reinforcement learning to solve the stabilization problem, where the goal is to learn a control feedback $u(k)$ that drives the state x(k) to the desired steady state. The neural network acts as a feedback controller that receives the current state as input and outputs an appropriate control action.



3 Steps of the Project

3.1 Incorporation Insights from Previous Research

Dr. Steffen W. R. Werner's research in [1] proves that the unstable eigen-space of a dynamic system can be used to solve for the stability of the system in place of the entire system. This is extremely valuable when we are trying to solve for stability of dynamical systems because instead of finding stability of a high dimensional $n \times n$ matrix we can narrow it down to solving for the stability of a $\hat{n} \times \hat{n}$ where $\hat{n}$ is the number of unstable eigenvalues of the system. In most cases, $\hat{n}$ is significantly smaller than n which leads to much lower computational cost.

We can incorporate this insight into this project by only training the feedback control neural network on the unstable eigenspace of the system. With a given unstable dynamical system we get an unstable eigenspace W that contains the eigenvectors corresponding to the unstable eigenvalues. If W contains z unstable eigenvalues Then we can use W to truncate the matrices A, B, and E

$$\begin{aligned} \hat{A} = W^T AW \quad & \hat{A} \in \mathbb{R}^{z \times z} \\ \hat{B} = W^T B \quad & \hat{B} \in \mathbb{R}^{z \times p} \\ \hat{E} = W^T EW \quad & \hat{E} \in \mathbb{R}^{z \times z} \end{aligned}$$

3.2 Building the Neural Network

The program will use reinforcement learning to train a neural network that will act as our control feedback. Before getting into the neural network we need to understand what reinforcement learning is.

Reinforcement learning works by training an agent that is given a set of parameters to follow and an objective. The objective is computed using a reward function. In the case of this project the reward function will be the distance from our steady state. The closer the system is to the steady state, the higher the objective, and the agent runs through trial and error, making small adjustments to maximize the objective.

We will use the code from [1]. The program contains an actor network that will act as the control feedback once trained, and a critic network, which is necessary for the training. The action network takes in the current state $x(k)$, then goes through the hidden layers of the neural network, and finally produces an output that we can use as our $u(k)$.

The critic network is necessary for the training and has two inputs, $x(k)$ and $u(k)$, a few hidden layers, and a single value as the output. The critic network is responsible for evaluating how good the new state is. Its main job is to estimate the value function, which guides the learning of the actor network and decides how good or bad the change in the actor network which helps the training agent decide how to change the actor network.

The following is the structure of the critic and actor network

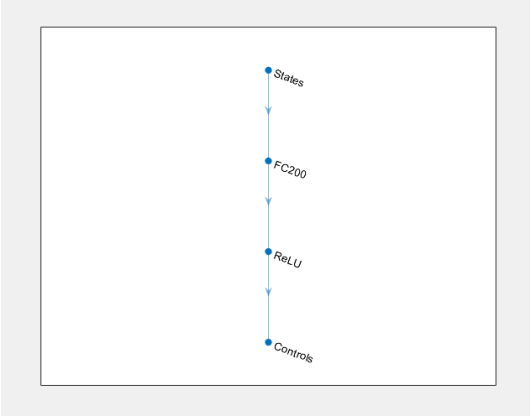


Figure 1: Actor Network

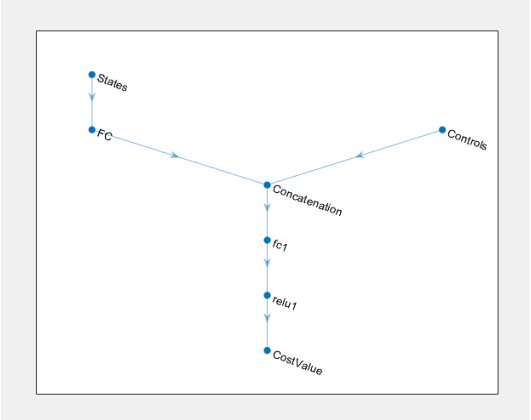


Figure 2: Critic Network

3.3 Testing on a Simple System

During the initial fine-tuning phase of building the reinforcement learning model, it was more productive to test on a simple system, and once I got results I was satisfied with, I would move on to the large real-world example.

I used the following simple discrete dynamical system:

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

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} x(k+1) = \begin{bmatrix} 1.2 & 0.3 \\ 0.4 & 0.8 \end{bmatrix} x(k) + \begin{bmatrix} 0.5 \\ -0.2 \end{bmatrix} u(k)$$

Figure 3 shows the system simulated with u set to ones. As we can see the system is in fact unstable due to the presence of instability.

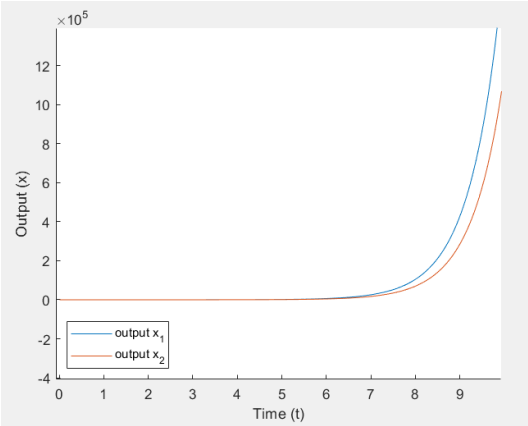


Figure 3: Unsteady System Simulated

The program finds the unstable eigenspace and computes the truncated matrices. Once the truncated matrices are computed the training agent solves for the feedback control using reinforcement learning.

When we train the agent on this simple system, we find a good enough reward in around 50 episodes of training. The following is the reward per episode over the training process.

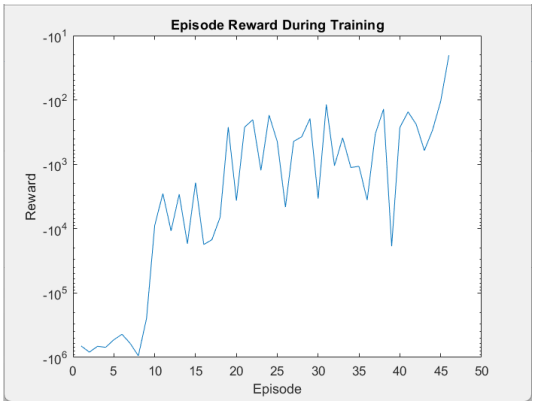


Figure 4: Training Reward Per Episode

This is a good representation of what the training should look like over time in reinforcement learning. The agent learns from its past episodes and improves over time to maximize the objective.

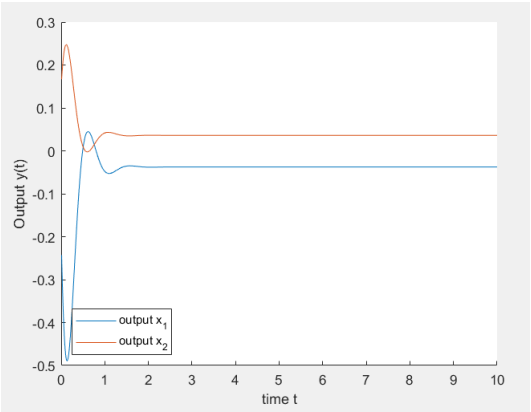


Figure 5: Truncated matrices simulation

Figure 5 shows the result when we simulate the system using the feedback control on the system with the truncated matrices. These are the smaller matrices that the neural network was trained on.

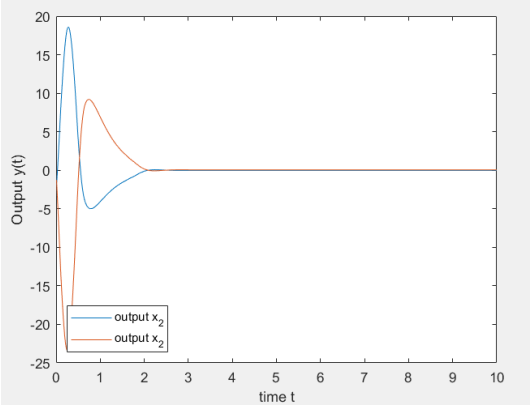


Figure 6: Full matrices simulation

Figure 6 shows the results when we simulate the system using the feedback control, but in this case, we are simulating the entire system with the full matrices. The system is reaching stability at the steady state $\vec{0}$, which is exactly what the agent was trained to do. This is significant because we are reaching stability of the full system when we are only training on the truncated version of the system's unstable eigenspace.

3.4 The HF2FD5 Data:

Consider the high-dimensional system, HF2D5, introduced in [2]. The system is in continuous time with $N = 4489$ and $p = 2$. The discrete-time version that we use is obtained in [1] with the implicit Euler discretization with sampling time $\tau = 0.1$. Both of the models have a single unstable eigenvalue, which causes the dynamical system to grow unbounded.

The HF2D5 system models a disturbed two-dimensional heat flow across a rectangular domain,

where external disturbances affect the system dynamics. As I mentioned before, without any efforts to stabilize the system, it will grow unbounded, which in this real-world example could lead to overheating of the rectangular domain, which could lead to damage due to overheating.

We can simulate the system with u equal to ones to show how the system grows unbounded due to the presence of the instability.

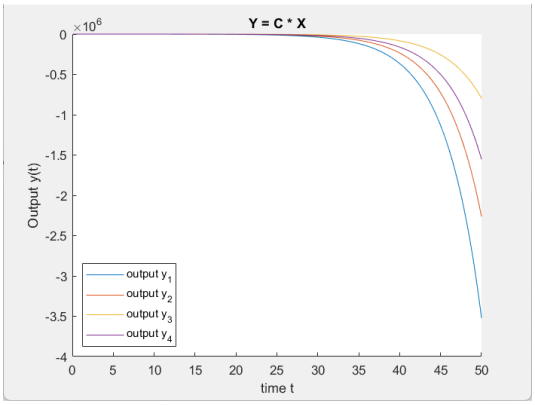


Figure 7: Unstable System simulated

3.5 Training the Agent on the HF2D5 data

When we train the agent on the heat-flow system, we reach a model with good results at episode 23. Figure 8 shows the episode reward for the training.

Figure 8: Reward over training (HF2D5)

Figure 9 shows the system simulated on the truncated matrix that the agent used to train. The system is staying constant at approximately 0.35.

Figure 10 is the system simulated using the full matrices. The system is reaching stability using the feedback control that was trained on the truncated matrices.

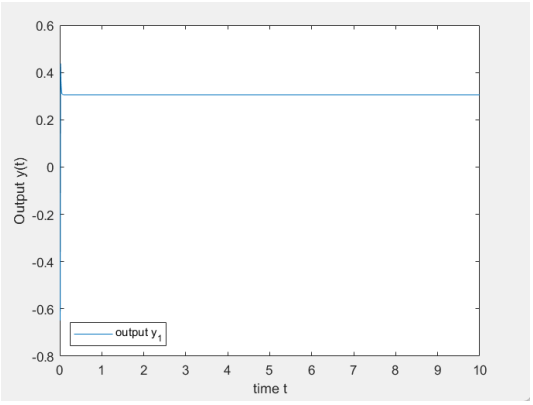


Figure 9: Truncated matrices simulation (HF2D5)

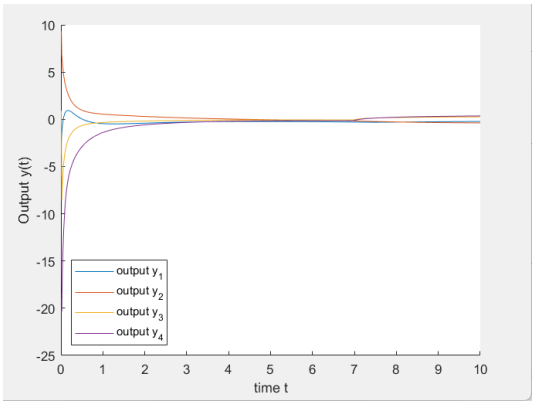


Figure 10: Full matrices simulation (HF2D5)

4 Results

The trained actor network consistently produces high-quality feedback control, successfully driving the system toward stability in both reduced and full-state simulations. Employing truncated matrices from the unstable eigenspace not only accelerates training by over 99 percent per episode but also yields robust control policies that generalize effectively to the full 4489×4489 dynamics.

It is important to note that our simulations using the feedback control are unstable and volatile before stabilization is reached, and in real-world applications, this may exceed safe thresholds, potentially causing damage precisely the scenario the controllers aim to prevent. Additionally, the model is trained for a finite number of iterations, and its performance may degrade when operating outside the range of states seen during training.

Current results rely on discretizing the underlying continuous-time model; future work should explore training directly on continuous dynamics to avoid discretization errors and extend stability guarantees beyond the training region.

References

[1] Werner SWR, Peherstorfer, *B. 2023 Context-aware controller inference for stabilizing dynamical systems from scarce data.Proc. R. Soc. A 479: 20220506.*, https://doi.org/10.1098/rspa.2022.0506

[2] *Leibfritz F. 2004 COMPleib: COnstrained Matrix-optimization Problem library–a collection of test examples for nonlinear semidefinite programs, control system design and related problems. Tech.-report University of Trier.*, http://www.friedemann-leibfritz.de/COMPlib_Data/COMPlib_Main_Paper.pdf.