Reinforcement Learning for Self-adapting Time Discretizations of Complex Systems

Conor D Gallagher

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Adrian Sandu, Chair
Muhammad Ali Gulzar
Anuj Karpatne

August 3rd, 2021
Blacksburg, Virginia

Keywords: Ordinary Differential Equations, Controllers, Reinforcement Learning

Reinforcement Learning for Self-adapting Time Discretizations of Complex Systems

Conor D Gallagher

(ABSTRACT)

The overarching goal of this project is to develop intelligent, self-adapting numerical algorithms for the time discretization of complex real-world problems with Q-Learning methodologies. The specific application is ordinary differential equations which can resolve problems in mathematics, social and natural sciences, but which usually require approximations to solve because direct analytical solutions are rare. Using the traditional Brusellator and Lorenz differential equations as test beds, this research develops models to determine reward functions and dynamically tunes controller parameters that minimize both the error and number of steps required for approximate mathematical solutions. Our best reward function is based on an error that does not overly punish rejected states. The Alpha-Beta Adjustment and Safety Factor Adjustment Model is the most efficient and accurate method for solving these mathematical problems. Allowing the model to change the alpha/beta value and safety factor by small amounts provides better results than if the model chose values from discrete lists. This method shows potential for training dynamic controllers with Reinforcement Learning.

Reinforcement Learning for Self-adapting Time Discretizations of Complex Systems

Conor D Gallagher

(GENERAL AUDIENCE ABSTRACT)

This research applies Q-Learning, a subset of Reinforcement Learning and Machine Learning, to solve complex mathematical problems that are unable to be solved analytically and therefore require approximate solutions. Specifically, this research applies mathematical modeling of ordinary differential equations which are used in many fields, from theoretical sciences such and physics and chemistry, to applied technical fields such as medicine and engineering, to social and consumer-oriented fields such as finance and consumer purchasing habits, and to the realms of national and international security and communications. Q-Learning develops mathematical models that make decisions, and depending on the outcome, learns if the decision is good or bad, and uses this information to make the next decision. The research develops approaches to determine reward functions and controller parameters that minimize the error and number of steps associated with approximate mathematical solutions to ordinary differential equations. Error is how far the model's answer is from the true answer, and the number of steps is related to how long it takes and how much computational time and cost is associated with the solution. The Alpha-Beta Adjustment and Safety Factor Adjustment Model is the most efficient and accurate method for solving these mathematical problems and has potential for solving complex mathematical and societal problems.

Dedication

Dedicated to Dr. D and Dr. Gallagher

Acknowledgments

First, I want to thank Dr. Adrian Sandu, my advisor, for recruiting me to his research team and providing me the thesis topic of developing intelligent, self-adapting algorithms for the time discretization of complex systems. Dr. Sandu allowed me to develop new expertise in Computer Science and explore the applications of Machine Learning to solve mathematical and societal problems. I learned a lot from him, my research project, and my lab group. I especially want to thank Abhinab, Amit, Andrey, Arash, Austin, Jostein, Oguz, Rachel, and Steven for their insights and contributions to my education and research.

I appreciate the input of my Computer Science committee members, Drs. Muhammad Ali Gulzar and Anuj Karpatne, who expanded my knowledge and application of Data Analytics. My education in VT Computer Science has been terrific at propelling me into my career of combining data science, mathematics, and Machine Learning.

During my MS Computer Science degree, I was a graduate teaching assistant in the Virginia Tech Center for the Enhancement of Engineering Diversity. I enjoyed launching the freshman engineers into their VT careers and beyond. I appreciate all my "CEED" colleagues, including Dr. Bevlee Watford, Ms. Susan Arnold Christian, Dr. DeAnna Katey, Ms. Becky Shelor, Dr. Kim Lester, and the GTA crew of Alex, Aarathi, Becca, Bemnet, Daniela, Domonic, Hannah, Jasmine, Mekonen, Oreoluwa, Perry, and Taylor. I appreciate the opportunity to work on water related projects though the VT Water INTERface Interdisciplinary Graduate Education Program.

My partner, Aarathi, is my joy and support before, during, and after graduate school. Thanks for all your encouragement; the walks in Hahn Garden, Pandapas Pond, and Hethwood Pond; finding and catching reptiles and amphibians; exploration of Bburg and VT

eateries; and those long days and nights together studying in Hancock, Torg, and Newman Library.

My friends were a continuous source of fun and diversion – thanks Alex, Devon, Eric, Robert, and Sherwood. May the games continue…

Lastly, my family is always there for me – thanks Mom and Dad! and Owen, Stevie, Par, Raghu, Zyra, Kalista, and Mango.

Contents

List of Figures x

List of Tables xiii

1 Motivation 1

2 Introduction 3

2.1 Differential equations 3

2.2 Runge-Kutta Methods and Step Size Controllers 4

2.3 Reinforcement Learning 6

2.3.1 Q-Learning 9

2.3.2 Test Problems 10

2.4 Project Objectives 13

3 Methods 16

3.1 Reinforcement Learning 16

3.1.1 Reward Function 17

3.1.2 Actions and States 19

3.1.3 Controllers and Runge-Kutta 19

3.1.4 Test Problems 20

3.1.5 Pipeline 21

4 Results and Discussion **22**

4.1 Training on Lorenz 63 ODE: Reward Functions 22

4.1.1 Reward as 1/(1-error) approach 22

4.1.2 Ratio approach 23

4.1.3 Reward of Error/1.5 24

4.2 Dynamically Tuning Controller Parameters 24

4.2.1 Alpha-Beta Model 24

4.2.2 Safety Factor Model 26

4.2.3 Alpha-Beta Adjustment Model 27

4.2.4 Safety Factor Adjustment Model 28

4.2.5 Alpha-Beta and Safety Factor Adjustment Model 29

4.2.6 Alpha-Beta Adjustment and Safety Factor Adjustment Model 29

4.3 Different Time Intervals for Lorenz 63 30

4.4 Different Tolerances for Lorenz 63 35

4.5 Gustafsson Controller 35

4.6 Summary of All Models 36

5 Conclusions **41**

Bibliography **44**

List of Figures

2.1 4^{th} order Runge-kutta method where k_1, k_2, k_3 and k_4 are the estimated slopes at different points in the interval [17]. 5

2.2 Reinforcement Learning schematic: the agent is the learner or algorithm that selects the action, a_t, that impacts the environment; current time step, t; current state, s_t; current reward, r_t; next time step, $t+1$. CC BY-NC [4] . . 8

2.3 Top: the solution of the Brusselator in an unstable regime and the phase space reaching a limit cycle as compared to Bottom: the solution of the Brusselator in a stable regime and the phase space reaching a fixed point [8]. 11

2.4 The two colors: yellow and blue represent the 3D evolution of two trajectories in the Lorenz attractor starting at initial points differing by 10^{-5} distance along the x-axis. The figure is made using $rho = 28$, $\sigma = 10$ and $\beta = 8/3$. (a) Initial time segment where the two trajectories are coincident, (b) the two trajectories slowly diverging and (c) the final time segment showing the two trajectories diverged [2]. 13

2.5 Simulation of the Lorenz 96 model where $n = 3$ [1]. 14

2.6 The proposed hybrid "numerical+learning" solution approach. The time discretization algorithm (blue) outputs the numerical solution, local error estimates, and diagnostics such as convergence rates. The learning algorithm (red) uses this information to adaptively reconfigure the numerical solver for the next step; it proposes a step size (or time step policy), a discretization method and order, updated values of solver parameters, etc. 15

3.1 Pipeline of how the Reinforcement Learning model is interacting with an ODE controller 21

4.1 Exploration and exploitation phase of the training for Lorenz 63 when Reward = 1/(1-error) and Punishment = –error. 23

4.2 Number of steps for exploitation phase of the training for Lorenz 63 when Reward = error/1.5 and Punishment = 0. 25

4.3 Error for exploration and exploitation phase of the training for Lorenz 63 when Reward = error/1.5 and Punishment = 0. 26

4.4 Behavior of alpha-beta model solving the Lorenz 63 problem. 27

4.5 Behavior of alpha-beta model solving the Lorenz 63 in the region where rejections occur. 28

4.6 Behavior of alpha-beta adjustment model solving the Lorenz 63 in the region where rejections occur. 29

4.7 Behavior of safety factor adjustment model solving the Lorenz 63 in the region where rejections occur. 30

4.8 Behavior of alpha-beta and safety factor adjustment model. 31

4.9 Close-up scale for behavior for alpha-beta and safety factor adjusted. 32

4.10 Alpha-beta adjustment and safety factor adjustment model. 33

4.11 0-5 Time interval to solve Lorenz 63. 33

4.12 0-10 Time interval to solve Lorenz 63. 34

4.13 0-15 Time interval to solve Lorenz 63. 34

4.14 Steps for Gustafsson Controller. 36

4.15 Error for Gustafsson Controller. 37

4.16 Gustafsson behavior. 38

4.17 Comparing the step ratio to error ratio for all problems 40

List of Tables

4.1 Ratio of number of steps for training on one model of different lengths. . . . 32

4.2 Ratio of errors for training on one model of different lengths 32

4.3 Ratio of errors for application of alpha-beta and safety factor adjustment model and training on different Runge-Kutta tolerances 35

4.4 Comparison of models for dynamically tuning controller parameters to baseline 37

4.5 Ratio of errors for all problems 39

4.6 Ratio of steps between baseline and model for all problems 39

4.7 Difference of steps between baseline and model for all problems 39

Chapter 1

Motivation

Mathematical models are increasingly complex and create demands and burdens on computational resources in terms of both computation time and memory storage. Application of mathematical modeling is expanding to almost every field, from theoretical sciences such as physics and chemistry, to applied technical fields such as medicine and engineering, to social and consumer-oriented fields such as finance and consumer purchasing habits, and to the realms of national and international security and communications. The desired solution to any of these problems, whether it is direct and simple or complex and interdisciplinary, will require accuracy and rapidity. Reinforcement Learning, an artificial intelligence method that is a subset of Machine Learning, can contribute greatly to solving problems. Reinforcement Learning explicitly focuses on solving the whole problem without breaking it into sub-problems. Through an iterative process that enables learning from actions and rewards, which are the consequences and successes from the decisions made, Reinforcement Learning moves toward maximizing the cumulative reward and rapidly and accurately solving the problem even if there are unforeseen issues.

Ordinary differential equations are prominent mathematical models employed to solve a myriad of critical problems where there is a complex system with rates of change, including: power grids, climate change, disease spread, cancer growth, traffic flow, fluid dynamics, chemical reactions, consumer behaviors, and more. Science, engineering, and society depend on ordinary differential equations. The challenge is that ordinary differential equations

are not usually solvable mathematically or analytically and thus use numerical methods to generate approximate solutions.

This work explores a novel approach to construct adaptive numerical integration algorithms via Machine Learning. We seek to build algorithms that try different time integrations and learn from these experiences as the integration progresses leading to higher accuracy solutions at a lower computational complexity. Reinforcement Learning, through its trial and error process, learns to move towards the optimal solution balancing accuracy and computational load and is thus ideal for efficiently and accurately solving ordinary differential equations. The novel application of Reinforcement Learning to develop intelligent, self-adapting algorithms for the time discretization of complex systems solutions to ordinary differential equations with the goal of increasing accuracy and reducing computational complexity motivates this work.

Chapter 2

Introduction

The goal of this research is to apply Machine Learning, or more specifically, Reinforcement Learning, to more efficiently solve ordinary differential equations (ODE) through optimizing highly interdependent factors of adaptivity of step size, order, and type of discretization (e.g., solver). The research will formalize the decision process of selecting these interdependent factors in a manner that can be extended to any family of integration methods, and to employ Reinforcement Learning to compute the optimal adaptation policies.

2.1 Differential equations

Ordinary differential equations are a prominent mathematical model extensively used in engineering and science to relate functions and their derivatives of one or more variables [3]. ODE can solve a myriad of important problems where there is a complex system with rates of change, including: power grids, climate studies, disease spread, traffic flow, fluid dynamics, chemical reactions, and more. The classic example is having the acceleration and desiring to calculate velocity as shown in Equation (2.1).

$$a = \frac{dv}{dt} = f(t, v(t)) \tag{2.1}$$

The objective is to solve for $v(t)$ with only starting conditions $v(t_0)$ and the differential

equation $\frac{dv}{dt}$. The challenge is that in general, ODE are unlikely to be solved mathematically or analytically. Consequently, numerical methods have been developed to generate approximate solutions, and it is desirable to know how close those approximate solutions are to the real solutions.

2.2 Runge-Kutta Methods and Step Size Controllers

A widely and successfully used numerical method to solve ODE is the Runge-Kutta (RK) family of iterative methods which apply temporal discretion steps to develop approximate solutions to ODE. Runge-Kutta methods contain an order which corresponds to accuracy of the step; the best known Runge-Kutta method is the 4^{th} order method, known as RK4 as shown in Figure 2.1 [3].

$$sc_i = Atol_i + \max(|y_{0i}|, |y_{1i}|) * Rtol_i \tag{2.2}$$

$$err = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_{1i} - \hat{y_{1i}}}{sc_i}\right)^2} \tag{2.3}$$

The higher the order of the Runge-Kutta method, the higher both the accuracy and also the computational cost. Implementing Runge-Kutta requires a starting y-value, step size (Δt) in the time domain, and the differential equation. The overall approach is to calculate the derivative at multiple points within the time interval, then combine the derivative values with different weights to calculate the value of y at a next time step. The order of the Runge-Kutta decides what points to pick in the time interval, $t + \Delta t$ and the weights to apply to the derivative values. This Runge-Kutta method is repeated until one has moved

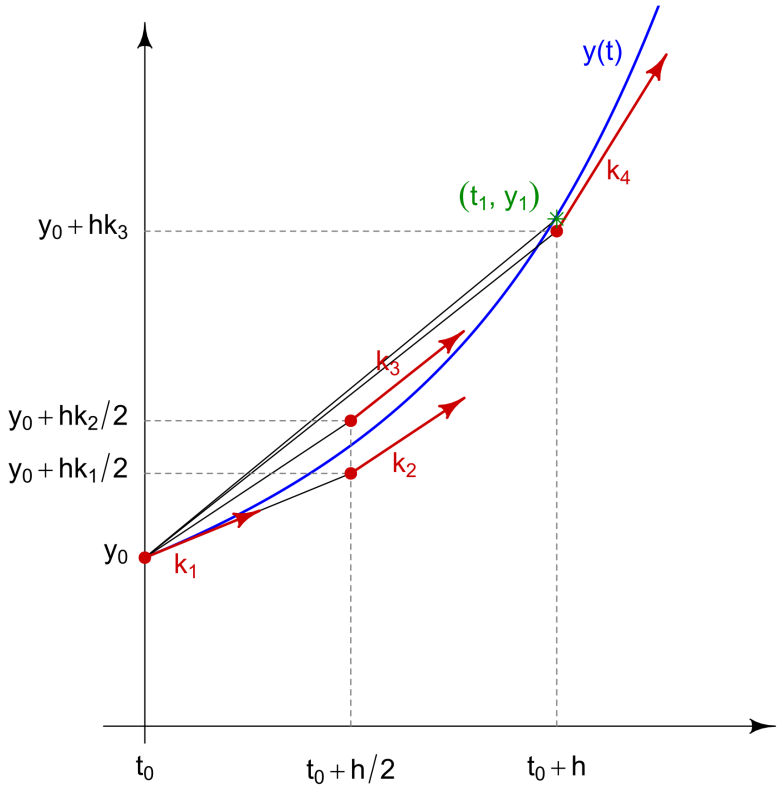


Figure 2.1: 4^{th} order Runge-kutta method where k_1, k_2, k_3 and k_4 are the estimated slopes at different points in the interval [17].

as far in the time domain as necessary. To have more confidence in the answer, a preferred method is an adaptive Runge-Kutta which compares two different Runge-Kuttas of the same step size but different orders. The difference between the two Runge-Kuttas in the adaptive Runge-Kutta is referred to as the error, i.e. the difference between y_1 and $\hat{y_1}$. By comparing the results, the adaptive Runge-Kutta can decide if the answers meet a certain tolerance and whether to accept the proposed step or reject it. There are two types of tolerances: absolute and relative as shown in equation 2.2. Absolute tolerance ($Atol$) sets the hard lower bound on the error tolerance and the relative tolerance ($Rtol$) counters the absolute tolerance by allowing for flexibility in error.

To calculate the error of the adaptive Runge-Kutta method, the difference in values $(y_1 - \hat{y_1})$ is normalized by the combination of relative and absolute tolerance, sc, component-wise as shown in eq. (2.3); note $sc \leq 1$. If the normalized error is below one then the step is accepted; if the normalized error is above one the proposed step is rejected. If the step was rejected the problem does not move forward in the time domain and a new step size is needed. The function that uses the normalized error as an input to change the step size is called the controller.

Selection of the step size is critical as it influences accuracy, error, and computational cost. The Adaptive Runge-Kutta decides whether to accept or reject the current integration step within preset tolerances, then the controller chooses a new step size or step size/order combination to a goal error of 1 [5]. To ensure we are moving forward in the time domain, a safety factor between 0 and 1 is applied to the controller's output. This increases the likelihood of the step size being accepted.

2.3 Reinforcement Learning

Machine Learning can be divided into three broad categories.

First, Supervised Learning, which uses fully labeled data where the correct answer is effectively known to train the algorithm. Typical examples are classification for categorical variables where the different categories are available and regression for continuous variables where the values of the response variable are known for the given explanatory variable values.

Second, Unsupervised Learning, where the training data are unlabeled, i.e., the algorithm is simply provided example data and asked to extract information from it. An example of Unsupervised Learning is using clustering algorithms to find structure within the data.

Because there is no formal "ground truth" available for the Unsupervised Learning, formally evaluating accuracy can be problematic.

Third, Reinforcement Learning (RL), which trains an algorithm using a reward system to achieve a particular objective. The reinforcement learning agent attempts to take the best next step in the process to optimize the overall final reward [12, 16].

Reinforcement Learning is a widely applicable and versatile form of artificial intelligence that uses a computational approach for goal-directed learning from interactions as shown in Figure 2.2. There are multiple steps in RL underpinned by the Markov Decision Process (MDP) [16]. The key player in RL is the agent, which is the learner or decision maker; the agent can be an algorithm solving ODE, or in a social setting it could be a person making a decision. Over discrete time steps, T, the agent takes actions, A, on their environment. The environment is outside of the agent, although the agent may have some knowledge about it and interacts with it. The actions are iterative steps that lead to different states, S, where a state, s_t could be the location of the agent in the environment. An action, a_t always leads the agent from it's current state, s_t to the next state, s_{t+1}. Each state-action pair for a particular time step, t has an associated reward, r_{t+1}, that evaluates the effectiveness of the transition from s_t to s_{t+1} using action, a_t. Rewards can be positive or negative, where a positive reward reinforces the effectiveness of an action and a negative reward is a punishment or failure. At each time step, the agent is taking an action and changing the state, resulting in a series of state-action pairs called the policy, Π_t. The policy is what the agent learns to optimize at a given time; this value can be stochastic and use a mathematical function or be a look-up from a table.

To summarize, using capital letters for random variables and lower case letters for the particular realization, the key components for Reinforcement Learning are:

- s_t is the state at time t out of possible system states S
- a_t is the action taken from state s_t out of possible actions A at time t
- $r_t(s_t, a_t)$ is the reward for taking action a_t from state s_t at time t

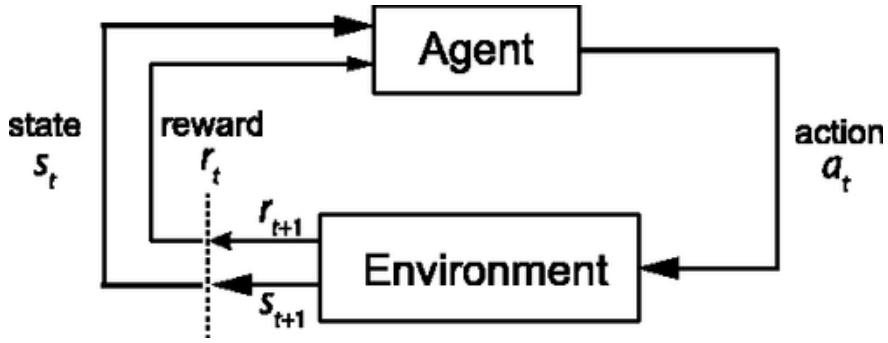


Figure 2.2: Reinforcement Learning schematic: the agent is the learner or algorithm that selects the action, a_t, that impacts the environment; current time step, t; current state, s_t; current reward, r_t; next time step, $t+1$. CC BY-NC [4]

RL is designed to maximize the cumulative reward and has three key components (Sutton and Barto 2018):

- Closed loop system because early actions-rewards influence later actions-rewards
- Actions are not prescribed or supervised, but are trial and error steps
- Learning occurs over time as action-reward progress progresses.

Unlike other types of Machine Learning, Reinforcement Learning distinguishes between an exploration and an exploitation phase. Exploitation takes the best decision given the current state, available actions, and what the agent has previously tried and found to be rewarding. But to gain such knowledge, the agent has to explore options it has not previously attempted. Neither exploitation nor exploration are sufficient alone. This exploration/exploitation trade-off is still an active area of research [16, 20].

One approach is to explore early in the problem, then slowly reduce the amount of exploration as time progresses. This is termed the ϵ-greedy approach, where ϵ is the probability of exploration at the start of the problem. ϵ decreases over time using a decay rate of λ [16].

RL can be implemented in many ways, of which Q-Learning is one of the more popular methods.

2.3.1 Q-Learning

Q-Learning is a particular type of Reinforcement Learning. "Q" refers to the expected rewards when selecting the next action to take given the current state. Q-Learning is considered model-free, because the transition probabilities when going from state s to state s' take action a and are not used or learned.

The "value" of an action a is the expected reward when a is selected from a given state. This is termed $q_*(a)$. The corresponding estimated value at time step t is termed $Q_t(a)$. These Q values are used to guide the selection of the next action taken. The Q-values are stored in a matrix (table) where the rows represent the states and the columns represent the actions. The matrix cells store the Q values for the state/action pair.

The Q-Learning update equation is thus

$$Q_{t+1}(s_t, a_t) = (1 - \alpha)\, Q_t(s_t, a_t) + \alpha \left[r_t(s_t, a_t) + \gamma \max_{a' \in A} Q_t(s_{t+1}, a') \right] \tag{2.4}$$

where $0 \leq \alpha \leq 1$ is the learning rate and $0 \leq \gamma \leq 1$ is the discount rate. A learning rate of 1 has the agent only consider the most recent reward. The discount factor weights the value of future (as opposed to current) rewards. A discount rate of 1 has the agent attempt long-term higher reward [20].

2.3.2 Test Problems

Brusselator

The Brusselator equations [13] (Figure 2.3) describe a system of autocatalytic oscillators, originally defined for a series of chemical reactions. For compounds A, B, C, D, X, Y, the starting chemical reactions for the Brusselator problem are

$$A \rightarrow X$$

$$2X + Y \rightarrow 3X$$

$$B + X \rightarrow Y + D$$

$$X \rightarrow E$$

If chemicals A and B are continually supplied and chemicals D and E are continually removed such that none of the four concentrations change over time and are therefore constant, only the concentrations X and Y are time variable. The resulting differential equation system, once normalized, is shown below.

$$\dot{x} = 1 - (b+1)\, x(t) + a\, x(t)^2\, y(t) \tag{2.5}$$

$$\dot{y} = b\, x(t) - a\, x(t)^2\, y(t) \tag{2.6}$$

where a, b > 0 are parameters and x, y ≥ 0 are the dimensionless concentrations.

The problem is nonlinear because of the $a\, x(t)^2\, y(t)$ term in each equation. Depending on

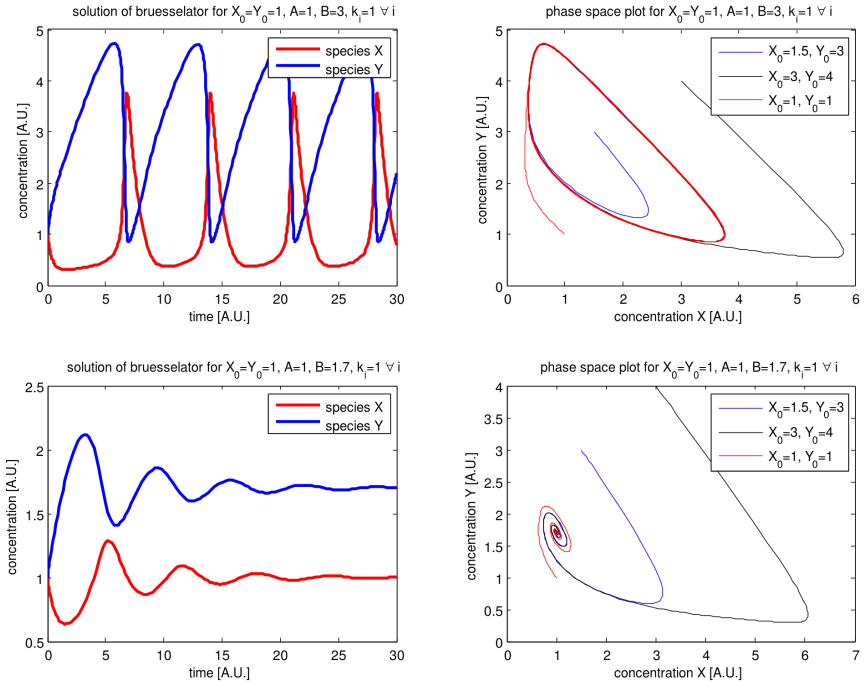


Figure 2.3: Top: the solution of the Brusselator in an unstable regime and the phase space reaching a limit cycle as compared to Bottom: the solution of the Brusselator in a stable regime and the phase space reaching a fixed point [8].

the values of the parameters a and b, the result can be periodic, spiral, or decay to fixed values. In particular $a = 1$ and $b = 4.5$ for periodic, and $a = 1$ and $b = 2$ for spiral.

Lorenz 63

Lorenz researched modeling of atmospheric systems which lead to the development of chaos theory. He developed several example nonlinear differential equation systems. The Lorenz 63 model [9] (Figure 2.4) derives a three dimensional system of deterministic ordinary differential equations that describe a fluid's convective motion held between two infinite horizontal parallel plates, where the upper plate is cooled and the lower plate is heated [19]. The solution can display very erratic dynamics which oscillate irregularly but remain bounded in a phase space [15].

$$\dot{x} = \sigma\left(y(t) - x(t)\right) \tag{2.7}$$

$$\dot{y} = x(t)\left(\rho - z(t)\right) - y(t) \tag{2.8}$$

$$\dot{z} = x(t)y(t) - \beta z(t) \tag{2.9}$$

where σ, ρ, and β are parameters > 0. The system is nonlinear because of the quadratic terms $x \cdot y$ and $x \cdot z$.

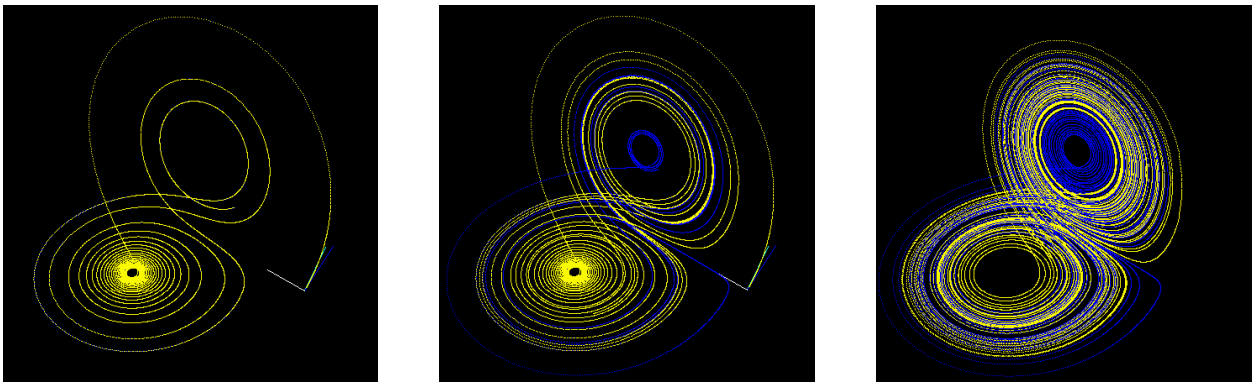
References [14] and [7] discuss solutions for different parameter values. The particular parameter values used by Lorenz (and in this research) were $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. The solution shows an initial transient period followed by an aperiodic irregular oscillation. If any two of the variables are plotted against each other, a strange attractor is displayed. An importance of this model is that it is highly sensitive to the initial values. Even slightly different values lead to very different solutions. The ability to predict future states is very limited because of this 'sensitive dependence' [19].

Lorenz 96

The Lorenz 96 model was designed to test the predictability of atmospheric and weather forecasting [10, 11] (Figure 2.5). It consists of a system of differential equations with a cyclic permutations of the variables. For the j_{th} variable:

$$\dot{x} = x_{j-1}\left(x_{j+1} - x_{j-2}\right) - x_j + F \qquad j = 1, ...n \tag{2.10}$$

Because the model is circulate symmetric [18], the boundary conditions are



(a) Time t=1 (b) Time t=2 (c) Time t=3

Figure 2.4: The two colors: yellow and blue represent the 3D evolution of two trajectories in the Lorenz attractor starting at initial points differing by 10^{-5} distance along the x-axis. The figure is made using $rho = 28$, $\sigma = 10$ and $\beta = 8/3$. (a) Initial time segment where the two trajectories are coincident, (b) the two trajectories slowly diverging and (c) the final time segment showing the two trajectories diverged [2].

$$x_{j-n} = x_{j+n} = x_j \tag{2.11}$$

In the original model development, the variables represent some atmospheric quantity along a circle of constant latitude of the earth, divided into n equal parts. The solution can have very complex dynamics depending of the values for the parameters n and F.

2.4 Project Objectives

The overarching goal of this project is to develop intelligent, self-adapting algorithms for the time discretization of complex systems, by augmenting traditional numerical analysis approaches with Machine Learning methodologies. The new hybrid "numerical+learning" solvers proposed herein use current simulation data to optimize performance on the specific problem at hand. Our specific objectives are as follows:

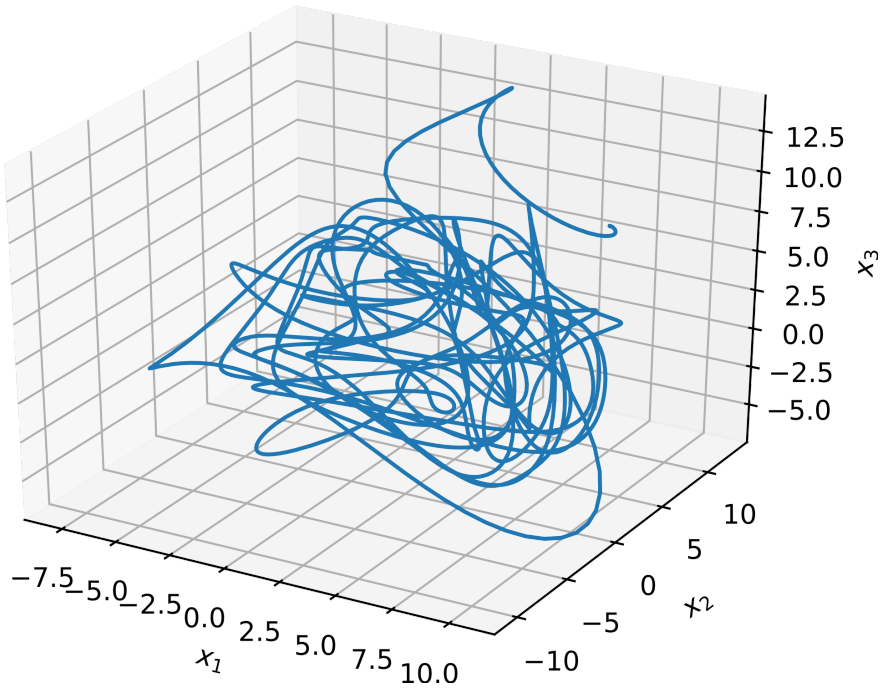


Figure 2.5: Simulation of the Lorenz 96 model where $n = 3$ [1].

1. Formalize the selection of step sizes, orders, discretizations, and solvers, as one Markov Decision Process, and employ reinforcement learning to compute optimal online adaptation policies.

2. Develop Reinforcement Learning-based performance models and use them to endow time integration codes with the ability to self-tune parameters such as to maximize performance for the simulation at hand.

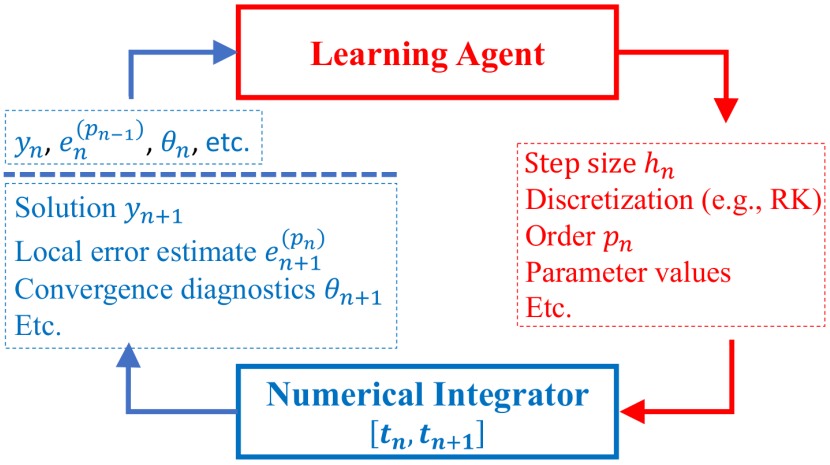


Figure 2.6: The proposed hybrid "numerical+learning" solution approach. The time discretization algorithm (blue) outputs the numerical solution, local error estimates, and diagnostics such as convergence rates. The learning algorithm (red) uses this information to adaptively reconfigure the numerical solver for the next step; it proposes a step size (or time step policy), a discretization method and order, updated values of solver parameters, etc.

Chapter 3

Methods

3.1 Reinforcement Learning

The goal of this research is to make a controller for ODE via a Reinforcement Learning model using an adaptive Runge-Kutta as the step size calculator. Reinforcement Learning requires three inputs: Actions, States, and Reward functions. The Actions for the model change the parameters to pre-existing step size controllers. The States are the history of past actions including the result of whether the step was accepted. The Reward function informs the model and decides to accept or reject a change in step size. We explore multiple Reward functions using the error of the step, step size, and if the step was accepted or rejected.

We train and test the Reinforcement Learning model on different ODE. We investigate two variations of the Brusselator problem which models an autocatalytic chemical reaction. We also explore Lorenz 96 and Lorenz 63 problems which are non-linear fluid dynamics problems with chaotic motion.

The Reinforcement Learning model used is Q-Learning, which requires a discrete action, state space, reward function and applying it to an ODE controller (Figure 2.6).

3.1.1 Reward Function

All Reinforcement Learning models require a reward function. We divide ours into two reward functions: one for when the step is rejected by the controller and one for when it is accepted by the controller which moves the solution forward in the time domain. When the step is rejected it is referred to as a punishment. We evaluate many different reward and punishment combinations. One point to note is that if the step is accepted then the error is bounded by 0-1 and if it is rejected then it has to be greater than one. This means that for rewards the error is less than one and for punishments the error is greater than one. Below are the different reward functions and punishment functions we evaluate. The error refers to the error of the adaptive Runge-Kutta calculated from 2.3. The current step size is refers to the step size that the adaptive Runge-Kutta is attempting to take.

In this work we consider the following reward functions:

1. Error/constant
2. 1/(1-error)
3. Current step size
4. Current step size * error
5. Ratio
6. Ratio minus Ratio of Baseline

In this work we also consider the following punishment functions:

1. - Error/constant
2. - Log(error)

3. 0

4. Current step size

5. - Current step size * error

6. Ratio

7. Ratio minus Ratio of Baseline

$$\frac{Current\ location\ *\ Total\ baseline\ number\ of\ steps}{Final\ Location\ *\ Model\ current\ number\ of\ steps} \tag{3.1}$$

It is possible to solve the ODE with a baseline controller and count the number of steps. This count becomes the baseline number of steps; the Reinforcement Learning model is rewarded when it solves the ODE faster, or, in fewer steps. To be able to apply each action, we determine how far we are into the ODE problem and how many steps were used. The model is rewarded for going faster than the baseline solver. As an example, if the model is half-way through solving an ODE problem and has only used a quarter of the steps as compared to the number of steps used by a baseline controller, then the model is rewarded more than if it used half of the total baseline controller steps. In terms of the ratio, using Equation (3.1), the value is $= \frac{5*100}{10*25} = 2$.

To account for the fact that this ratio assumes that all the problems being solved use equal step sizes throughout the problem, our innovation is to calculate the ratio while solving the baseline problem to get the baseline ratio at each step and time during the solution. We then know our position in time and ratio while training the model and can subtract the baseline ratio from the same position in time and reward the difference. This allows the model to be trained to do actions that lead to faster solutions than the baseline version. Mathematically,

the ratio now becomes:

$$\frac{Current\ location\ *\ Total\ baseline\ number\ of\ steps}{Final\ Location\ *\ Model\ current\ number\ of\ steps} - \frac{Current\ location\ *\ Total\ baseline\ number\ of\ steps}{Final\ Location\ *\ Baseline\ current\ number\ of\ steps} \tag{3.2}$$

3.1.2 Actions and States

The actions, or what the model can change, are safety factor, and the parameters of the controllers. The model can be set up to do this in two ways. The first way is to choose the action from a list of available values. As an example, the model can choose a safety factor of 0.7, 0.8 or 0.9. The second way is to adjust the action by a certain factor, for example, adjusting the safety factor by adding/subtracting 0.05 to the safety factor. The adjustments come from a predetermined list. Due to size constraints of the problem, we are unable to test all actions at once.

A state is the last three actions taken by the model and whether that action was accepted or rejected.

3.1.3 Controllers and Runge-Kutta

In Equation (3.4), e_n is the error from our most recent step and the e_{n-1} is the error from the 2nd last step. P is the order of the Runge-Kutta. The step size changes by the optimal factor, f_{opt}. tol is treated to be one [6]. α and β are parameters to the controller which manipulate the error; α and β can be considered analogous to how the error is weighted. The explicit Dormand–Prince order 5 and 4 is used for the adaptive Runge-Kutta.

$$f_{opt} = \left(\frac{tol}{e_n}\right)^{k_I} \left(\frac{e_{n-1}}{e_n}\right)^{k_P} \tag{3.3}$$

$$f_{opt} = \left(\frac{tol^{\alpha+\beta}}{||e_n^{(p,h_n)}||^{\alpha}||e_{n-1}^{(p,h_{n-1})}||^{\beta}}\right)^{\frac{1}{p+1}} \tag{3.4}$$

$$h_{n+1}^{(p)} = \text{fac} \cdot h_n. \;\; \text{fac} = min\left\{f_{max}, max\left\{f_{min}, f_{safe} \cdot f_{opt}\right\}\right\} \tag{3.5}$$

$$k_I + k_P = \frac{1}{p} \tag{3.6}$$

The second one is called Gustafsson and is defined in Equation (3.3). k_I and k_P are similar to alpha and beta where they act as weights to the normal factor and error of ratios, respectively. Both controllers use Equation (3.5) where h_n is the current step size, h_{n+1} is the new step size, f_{min} is the minimum change in step size, and f_{max} is the maximum change in step size. Lastly, f_{safe} is the safety factor that helps ensure that the step will be accepted. If a step is rejected the computational cost doubles as the amount of calculations needed to move forward in the time domain doubles. By selecting a f_{safe} that ensures acceptance of the step, we avoid the possibility of doubling the computational cost. Normal safety factors are $0.8, 0.9, (0.25)1/(p+1)$, or $(0.38)1/(p+1)$.

3.1.4 Test Problems

The solution code for the Brusselator and Lorenz test problems is taken from https://github.com/ComputationalScienceLaboratory/ODE-Test-Problems released under the MIT license. The Reinforcement Learning code is written in Matlab.

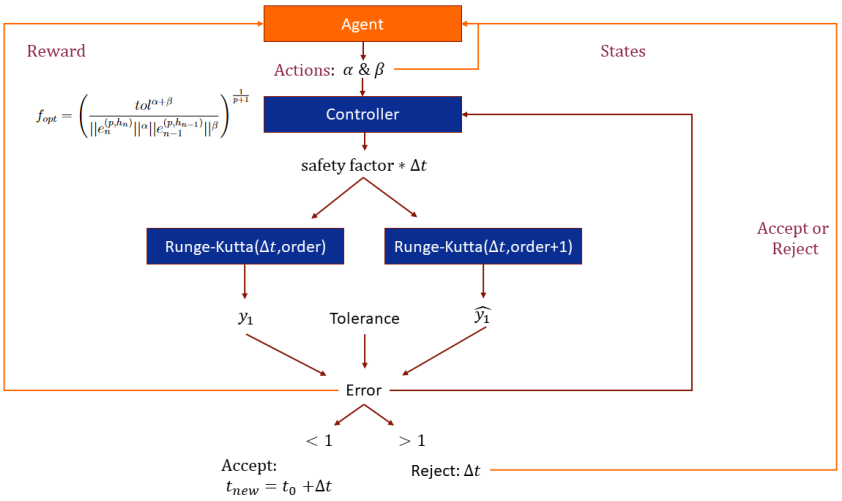


Figure 3.1: Pipeline of how the Reinforcement Learning model is interacting with an ODE controller

3.1.5 Pipeline

The complete pipeline of applying reinforcement learning to differential equation is captured by the Figure 3.1. At each step the reinforcement learning agent decides the inputs to the controller, such as alpha and beta for the alpha-beta controller. Then the controller decides the step size for the adaptive runga kutta. The results of the adaptive runge kutta is used to estimate the local error. The error estimate decides if the calculated step is accepted or rejected. The result of the current step and the past actions and their respective results together form the current state for the agent. The controller then uses the local error estimate to calculate the next step size the agent also uses the local error as an input for the reward function. Once one step is taken the result and the action change the state causes the agent to pick new inputs to the controller and the process repeats.

Chapter 4

Results and Discussion

4.1 Training on Lorenz 63 ODE: Reward Functions

For the base example we use Lorenz 63 with the baseline set to alpha = 0.9 and beta=0.1. The number of training episodes is preset to $e + 06$. The models and the baseline use an absolute tolerance and relative tolerance of $e - 07$. Lorenz 63 parameters are $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$. The problem is solved from Time = 0 to Time = 5.

4.1.1 Reward as 1/(1-error) approach

The Reinforcement Learning model tunes alpha and beta using the controller (Equation (3.4)). Many different reward functions are attempted. The first reward and punishment pair are:

- Reward = 1/(1-error)
- Punishment = –error

Figure 4.1 shows the exploration phase of the training. We can see that as more runs are performed, the model takes more steps because it is learning to avoid getting rejects when the punishments are high. This trend exists for most of the rewards and punishments when their difference are high such as when we use the current step size as reward. The model is being overly punished for a reject.

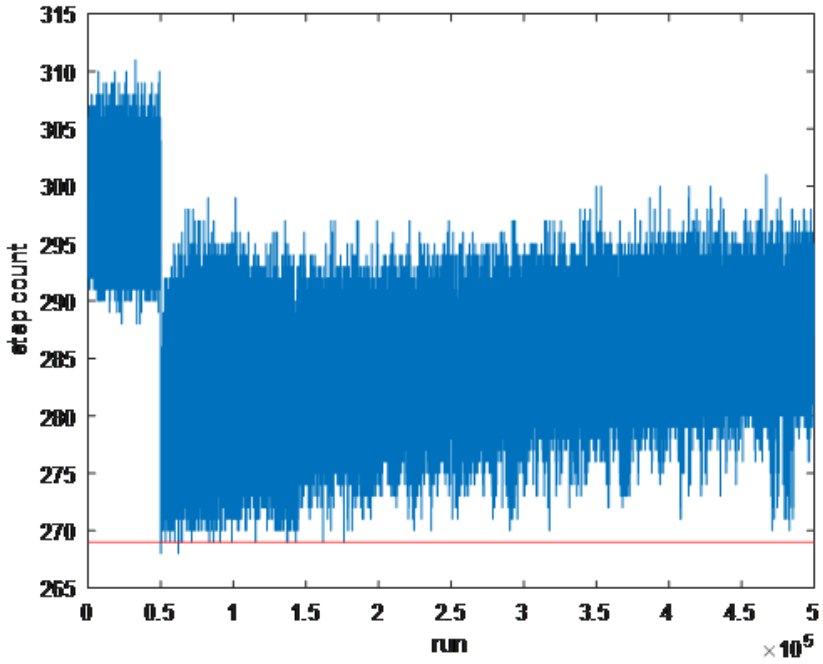


Figure 4.1: Exploration and exploitation phase of the training for Lorenz 63 when Reward = 1/(1-error) and Punishment = –error.

4.1.2 Ratio approach

The next reward function is the ratio method, or, how fast we go vs how fast we should be going.

- Reward = Ratio
- Punishment = Ratio

The ratio method does not out-perform the baseline because after one bad step, all the future steps in that solution will be off. This leads to the state action pair in the Q-table to have Q-values that will lead to sub-optimal solutions. The other issue is that the ratio method assumes an equal step size while solving the problem. To account for these issues, a 2nd ratio method is used (Equation (3.2)).

In the 2nd ratio method the reward is how much the action speeds up or slows down while solving the problem. This ratio method also does not out-perform the baseline. It is not able to replicate the baseline solution even with the 2nd ratio method. That is because after the first several actions do better than the baseline, the 2nd ratio method sets the model on a path of local minima that always perform worse than the baseline solution.

4.1.3 Reward of Error/1.5

The next reward function we investigate is this reward/punishment pair:

- Reward = error/1.5 (error is bounded by 1-0)
- Punishment = 0

We can see from the Figure 4.2 and 4.3 that this reward does not perform worse as we run more episodes and it is able to meet the baseline. Since this is the best performing rewards/punishment pair, for the rest of the experiments we will use a reward of error/1.5 a and punishment of 0.

4.2 Dynamically Tuning Controller Parameters

These models dynamically change the inputs to the controllers which change the step size.

4.2.1 Alpha-Beta Model

This approach allows the model to decide between discrete options for alpha and then back calculate beta as 1-alpha (Equation (3.4)). The graph in Figure 4.2 is during the end of the

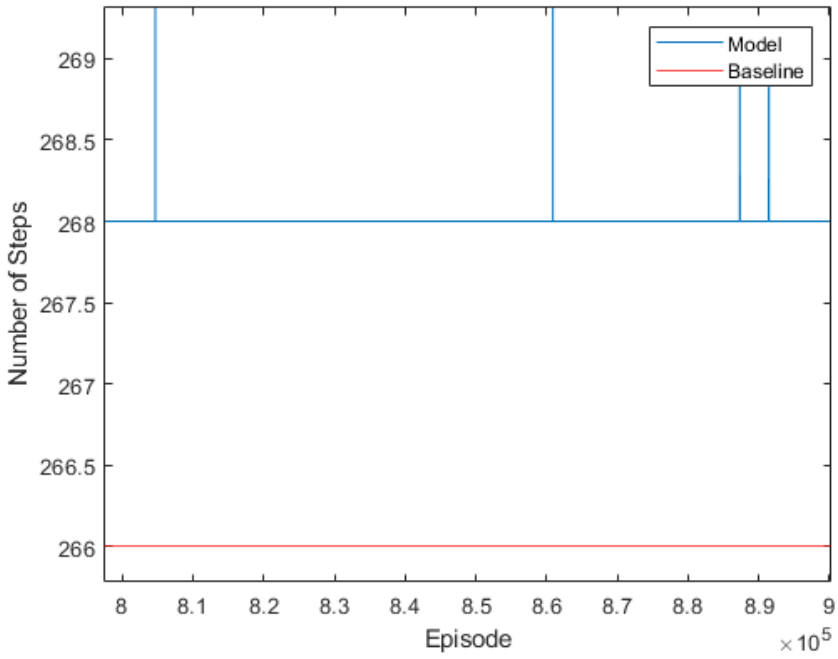


Figure 4.2: Number of steps for exploitation phase of the training for Lorenz 63 when Reward = error/1.5 and Punishment = 0.

exploitation phase of training and shows that the problem stabilizes in 268 steps, one step less than the baseline.

In Figure 4.3, looking at the end which is the exploitation phase of the training, we see the final error from the model solution. We get this by comparing the final location of the model solution and comparing that to a solution with a very tight tolerance (i.e., tolerance of e$-$10 which was preset) which we will treat as the actual solution. The graph shows that the model performs slightly worse than the baseline version.

An interesting result is the behavior of alpha once the model is complete, as shown in Figure 4.4. In the regions of no rejection, there is a pattern where alpha cycles through $0.4 \rightarrow 0.9 \rightarrow 0.7 \rightarrow 0.8$ and repeats. The safety factor is set to be constant

Another interesting result shown in Figure 4.5 is that after a rejection, the alpha-beta model

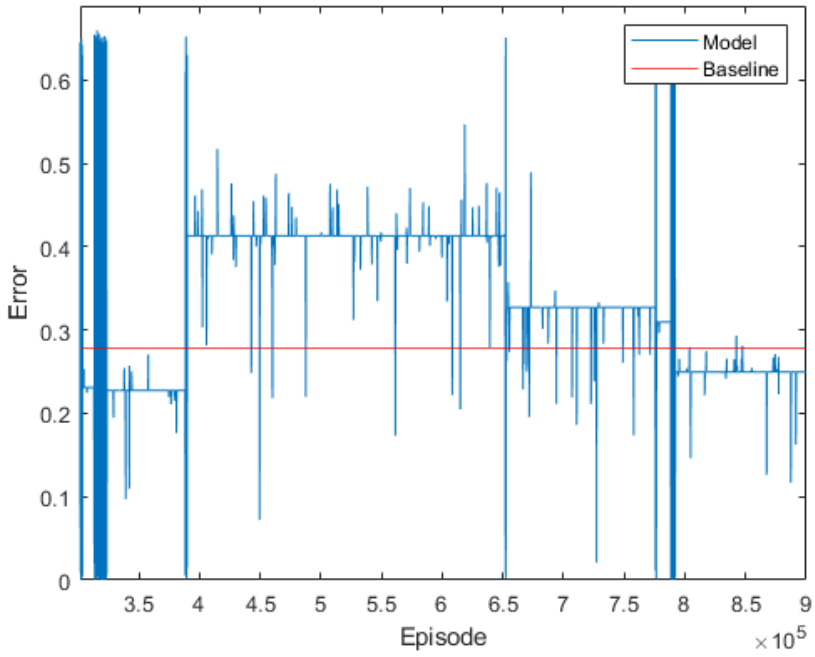


Figure 4.3: Error for exploration and exploitation phase of the training for Lorenz 63 when Reward = error/1.5 and Punishment = 0.

keeps alpha high for the next 4-6 steps and then plummets alpha to 0 or a much lower value. This generates the idea that after a rejection from the controller, an immediate adjustment is not appropriate, but adjusting after a few steps could be a better approach.

4.2.2 Safety Factor Model

The safety factor helps insure that the step size is going to be accepted because the safety factor helps the step to not be rejected. When the model is allowed only to choose a safety factor from a discrete list of values (i.e., 0.7579, 0.8, 0.8241, 0.85, 0.90), we see similar results to when it controls just alpha in terms of error and the number of steps to complete the problem. The only interesting result is it adjusts the safety factor in the first ten steps and then just leaves it at 0.9. Overall, this method does not give insights into developing

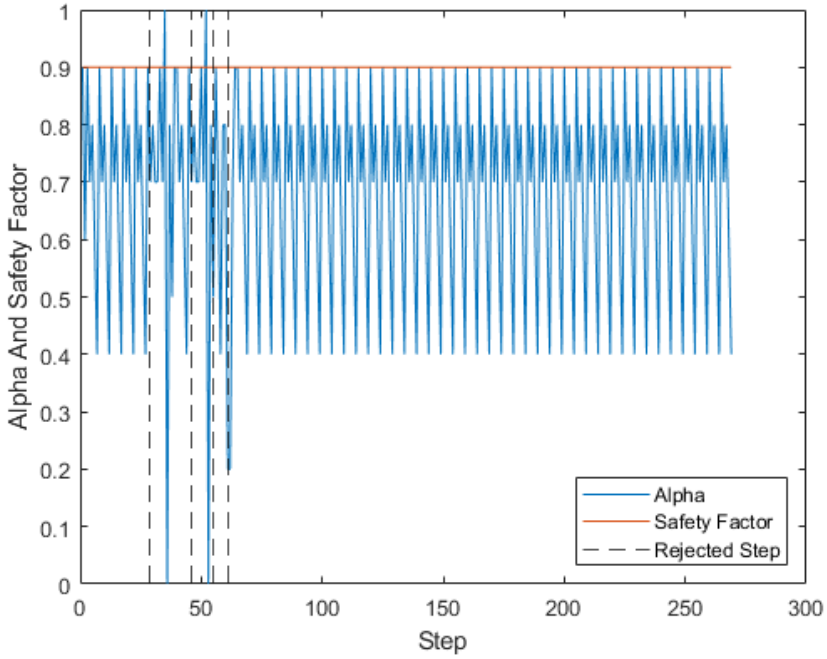


Figure 4.4: Behavior of alpha-beta model solving the Lorenz 63 problem.

controllers.

4.2.3 Alpha-Beta Adjustment Model

This model allowed adjustment of alpha by a discrete value (i.e., -0.1, -0.05, -0.01, 0, 0.01, 0.05, 0.1) (Figure 4.6). The method lets the model fine tune the alpha values without vacillating. A similar trend is seen of where it takes 264 steps to complete vs. the baseline of 266 steps. The error has doubled from the baseline to be around 0.61. The behavior of the model is that it finds a loop that keeps alpha between 0.95 and 1. But at the first sign of the rejected step it has a delayed reaction before it starts to lower alpha.

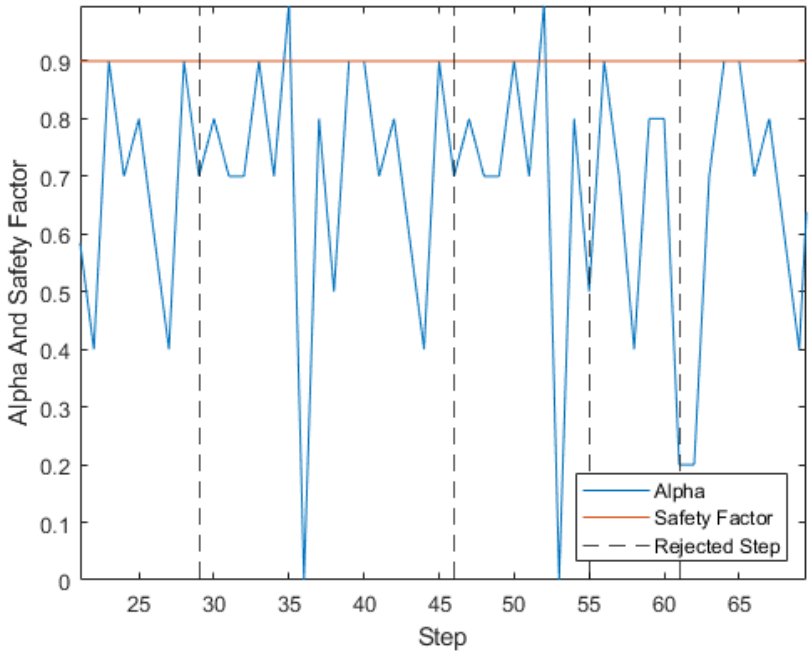


Figure 4.5: Behavior of alpha-beta model solving the Lorenz 63 in the region where rejections occur.

4.2.4 Safety Factor Adjustment Model

When just the safety factor is adjusted by discrete amounts, it solves the Lorenz 63 problem in 274 steps compared to the baseline of 266 steps. The biggest change that happens is that the safety factor adjustment model runs into a total of 28 rejected states compared to the baseline which has 8. This is to be expected when the safety factor increases. The steps the safety factor adjustment model takes are bigger in order to solve the problem faster resulting in less accepted states. The interesting result to note is that in the beginning of the problem when multiple rejections occur close together, the model decreases the safety factor because it realizes that it was dealing with a tighter region. Comparing that to when it was solving the problem later, the model increases the step size after a rejection because it accounts for the controller to lower the step size as seen in Figure 4.7.

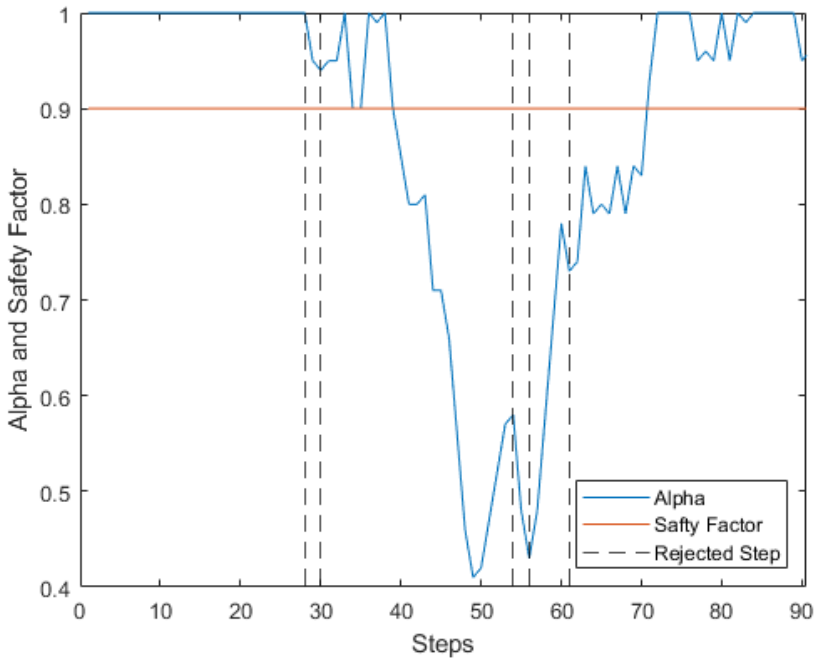


Figure 4.6: Behavior of alpha-beta adjustment model solving the Lorenz 63 in the region where rejections occur.

4.2.5 Alpha-Beta and Safety Factor Adjustment Model

This model can choose alpha from a discrete list and then also choose how to adjust the safety factor. The model finishes Lorenz 63 in 283 steps with 32 rejections, compared to the baseline of 266 steps and 8 rejections. The model error is 0.488 compared to the baseline error of 0.339. The behavior is shown in Figure 4.8; the trend is that alpha tends to alternate between a high value and a low value while the safety factor seems to stay between 0.9 and 1. This model has no noticeable trend and the initial results do not look promising.

4.2.6 Alpha-Beta Adjustment and Safety Factor Adjustment Model

This method adjusts both the alpha-beta and the safety factor. To solve the Lorenz 63 alpha-beta adjustment and safety factor adjustment model, our model requires 273 steps with 27

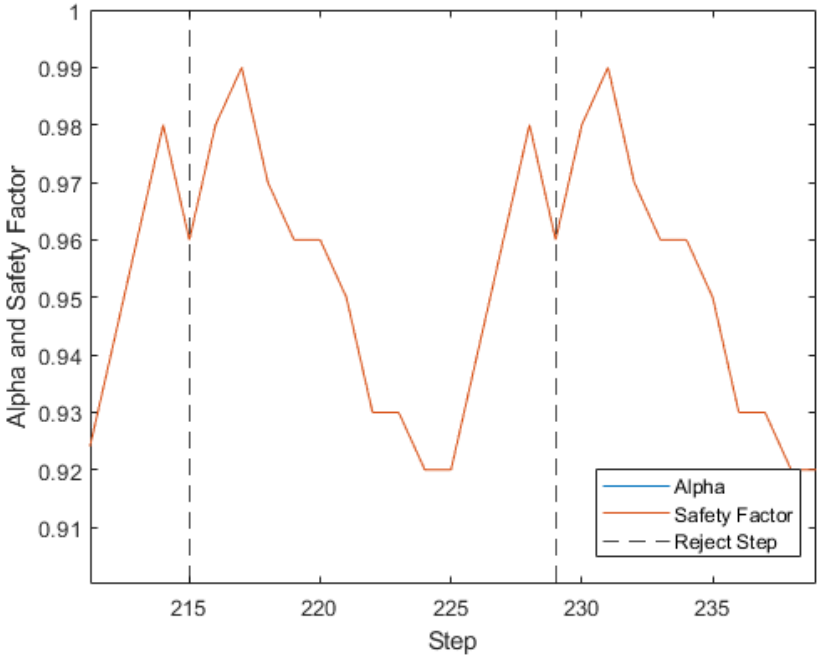


Figure 4.7: Behavior of safety factor adjustment model solving the Lorenz 63 in the region where rejections occur.

rejects and resulted in an error of only 0.065. While it is close to the baseline of 266 steps, the baseline error is 0.33. Overall, this is the best model for tuning controller parameters and yields the greatest accuracy even if the step size is slightly higher than the baseline. As seen in Figure 4.10, compared to the alpha-beta adjustment method (Figure 4.7), we see a similar alpha-value drop around 50 steps, but the alpha-beta adjustment and safety factor adjustment model afterwards has more fluctuation and seemed to respond better.

4.3 Different Time Intervals for Lorenz 63

We extend the model in the time domain to solve the problem from 0 to 10 and 0 to 15 using the alpha-beta adjustment and safety factor adjustment model. We can see interesting results in the 0-5 problem (Figure 4.11) which converges on a solution fairly fast and has

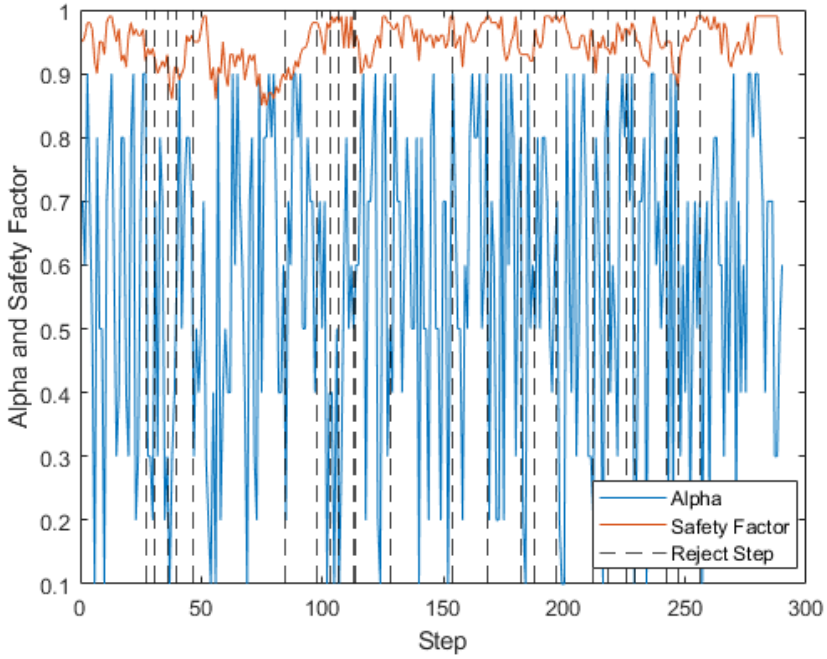


Figure 4.8: Behavior of alpha-beta and safety factor adjustment model.

minor changes. While the interval 0 to 10 (Figure 4.12) is constantly getting better over episodes, the interval 0 to 15 (Figure 4.13) policy never converges and is constantly changing. So for long time domains, more training is needed for the policy to converge.

Next we look at the metrics of accuracy and number of steps to train on one model of different time intervals. Table 4.1 (baseline steps over model steps) shows the number of steps of the model are always more than baseline but not really different between the models trained on different time intervals. Each trained model performs worse by about 5 to 10 percent, irrespective of which model they are tested on.

If we look at Table 4.2 for the error, we do not really see a clear pattern, all models performed well against the 0-10 baseline but not the 0-5 or 0-15 baseline. An item to note is that the baseline for the 0-15 interval was a very accurate solution and this happens but is not usually the case. The 0-15 model does not converge to an answer which means its results are really

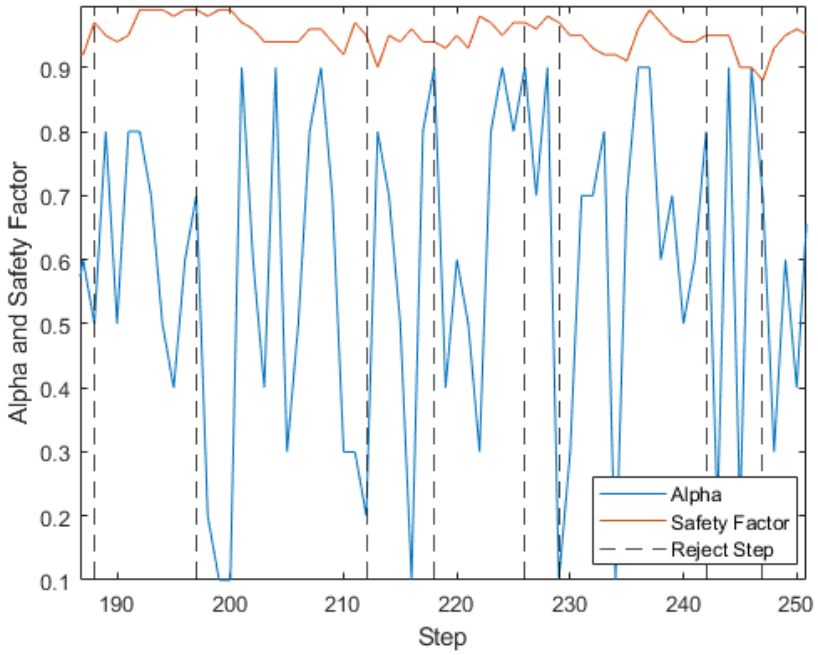


Figure 4.9: Close-up scale for behavior for alpha-beta and safety factor adjusted.

Table 4.1: Ratio of number of steps for training on one model of different lengths.

Tested on:	**Ratio (Baseline Steps/Model Steps) when Trained on:**		
	Lorenz 63, 0-5	**Lorenz 63, 0-10**	**Lorenz 63, 0-15**
Lorenz 63, 0-5	0.912	0.96	0.924
Lorenz 63, 0-10	0.904	0.94	0.929
Lorenz 63, 0-15	0.902	0.94	0.938

just noise. We are not positive that the 0-10 model fully converges on an answer. Overall, the results of training on different time interval models and then testing on the different time intervals are inconclusive.

Table 4.2: Ratio of errors for training on one model of different lengths

Tested on:	**Ratio (Baseline Error/Model Error) when Trained on:**		
	Lorenz 63, 0-5	**Lorenz 63, 0-10**	**Lorenz 63, 0-15**
Lorenz 63, 0-5	0.8556	0.6150	0.7749
Lorenz 63, 0-10	2.9481	1.0279	1.6581
Lorenz 63, 0-15	0.0635	0.0751	0.2892

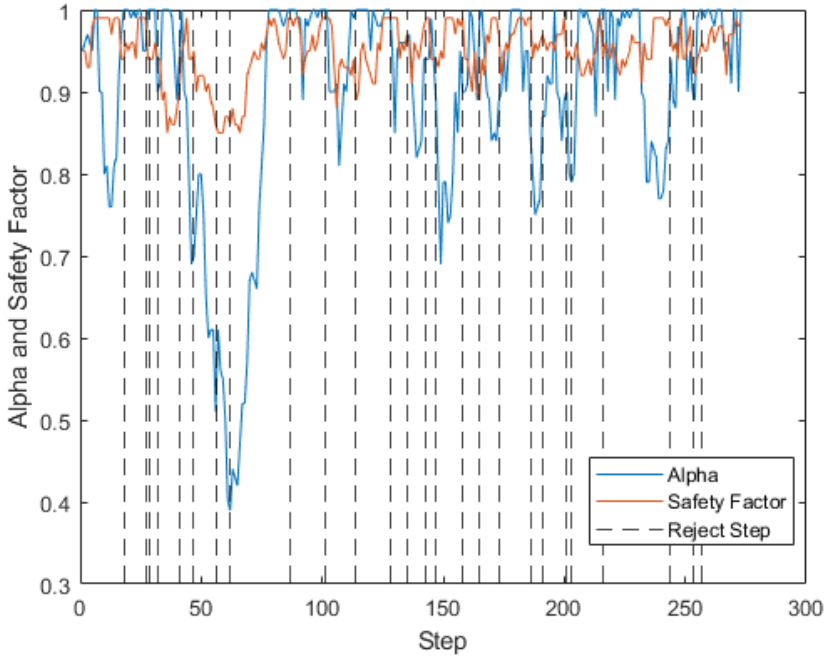


Figure 4.10: Alpha-beta adjustment and safety factor adjustment model.

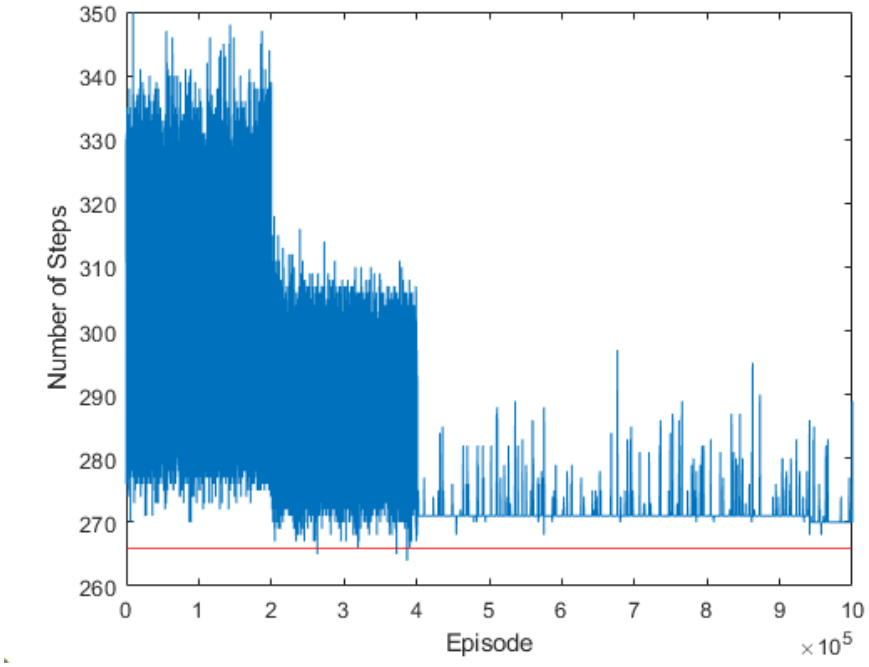


Figure 4.11: 0-5 Time interval to solve Lorenz 63.

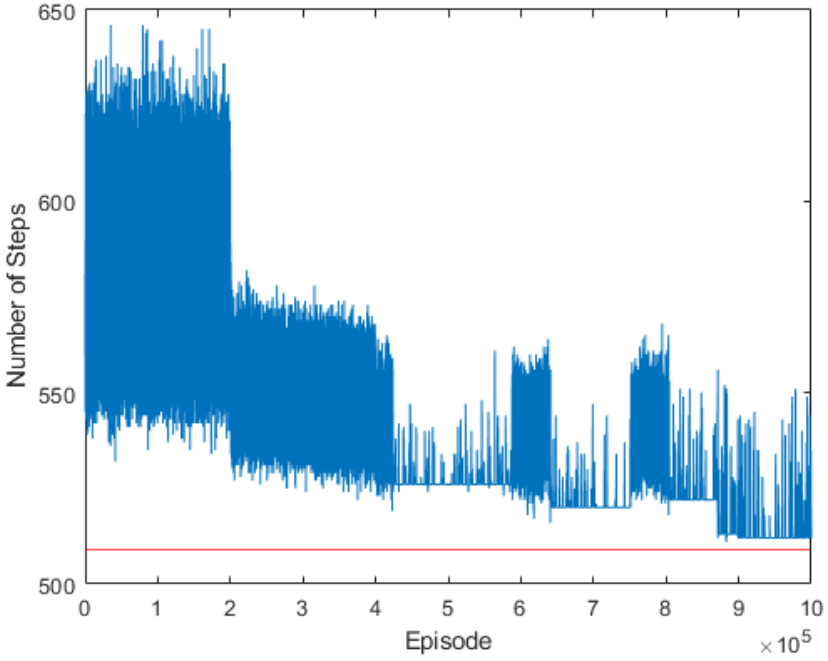


Figure 4.12: 0-10 Time interval to solve Lorenz 63.

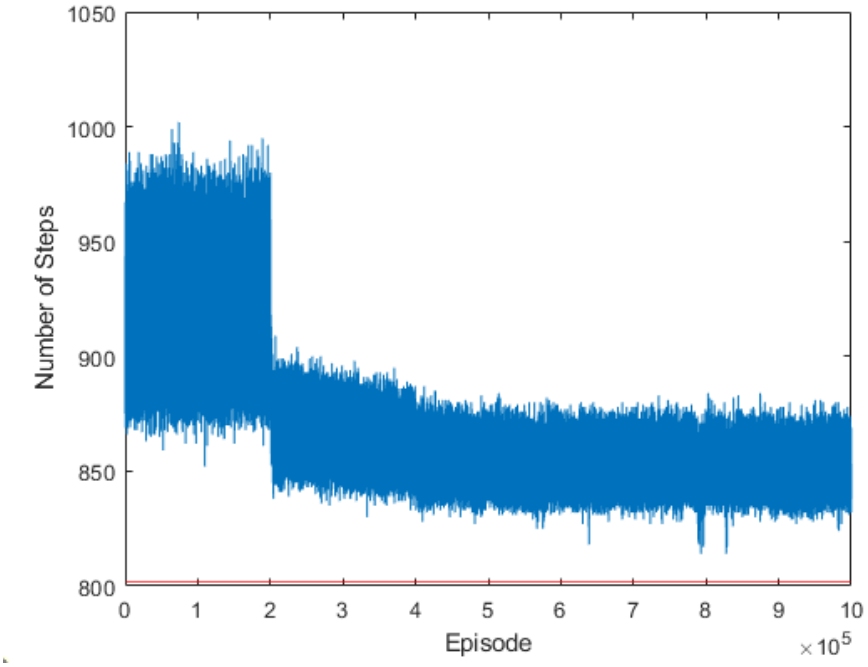


Figure 4.13: 0-15 Time interval to solve Lorenz 63.

4.4 Different Tolerances for Lorenz 63

We apply the alpha-beta and safety factor adjustment model and training on different Runge-Kutta tolerances (Table 4.3). When we use the tolerances e-3 to test the models, they do not get better than when trained on tighter tolerances. In fact, a worse result at e-7 occurs. This might mean at loose tolerances there is not much improvement. One possible reason is that it solves the e-3 too fast to let the model make any improvements. Intuitively, when training a model on a tighter tolerance, we think that when it is tested on a lower tolerance we keep more of the accuracy. We see more accuracy to a degree when we test it on e-5 after training on the higher tolerances. We see the opposite for e-7 where the training on looser tolerances are much more accurate when testing on the higher tolerances. These results could just be a fortuitous solution to the problem. We interpret these data as the model cannot help when the problems are solved quickly for the e-3 it solves in 50-60 steps vs e-7 290-300 steps. For the number of steps, the models take around 7 or 18 more steps with no real pattern.

Table 4.3: Ratio of errors for application of alpha-beta and safety factor adjustment model and training on different Runge-Kutta tolerances

Tested on:	**Ratio (Baseline Error/Model Error) when Trained on:**		
	e-3	**e-5**	**e-7**
e-3	0.7459	1.4093	0.7910
e-5	1.2793	3.5231	6.9210
e-7	113.6626	42.4320	5.2281

4.5 Gustafsson Controller

Instead of the model learning on the equation 3.4, we train on the Gustafsson equation 3.3. The model chooses the k_i then calculates k_p based on equation ($k_i + k_p = 1/\text{order}$). Based on Gustafsson [5], he suggested using $k_i = 0.13$ and $k_p = 0.06$; this will be referred to as the

Gustafsson baseline. We also have $k_i = 0.2$ and $k_p = 0$ referred to as baseline. Figure 4.14 shows the training. We can see that Gustafsson baseline takes more steps than the model and baseline version. When we compare the errors (Figure 4.15), it shows that the model combines the best of both baselines with less steps than Gustafsson baseline and higher accuracy than the normal baseline. The Gustafsson behavior at steady state takes steps at the k_i =0.2 for 2 steps and then a couple steps around $k_i = 0.14$. When the model run into rejects, it alternates between high k_i and low k_i (Figure 4.16).

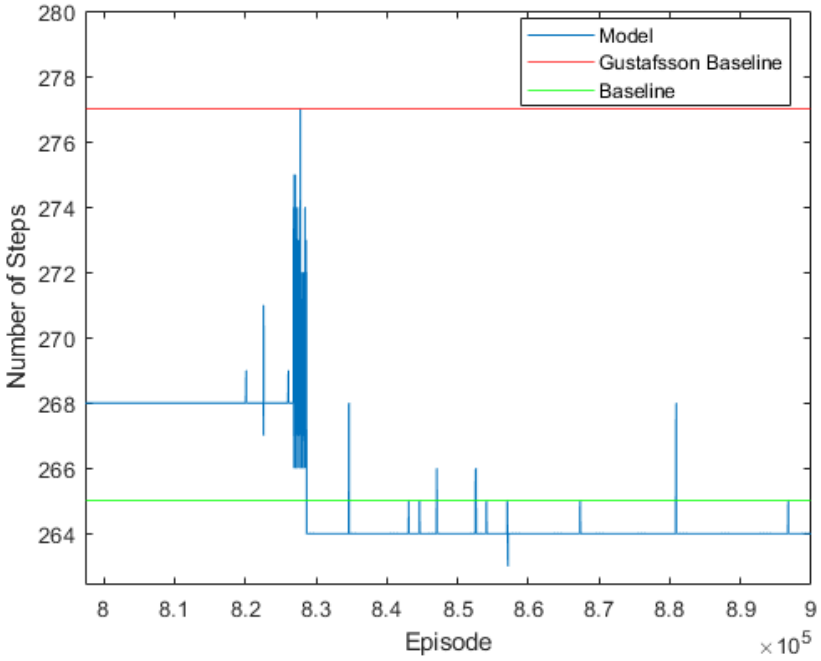


Figure 4.14: Steps for Gustafsson Controller.

4.6 Summary of All Models

If we review these different models and compare performances, Table 4.4 shows that overall that the models take a few more steps than the baseline number of steps. For error, we observe an improvement in the error when we change the safety factor, more specify, when we

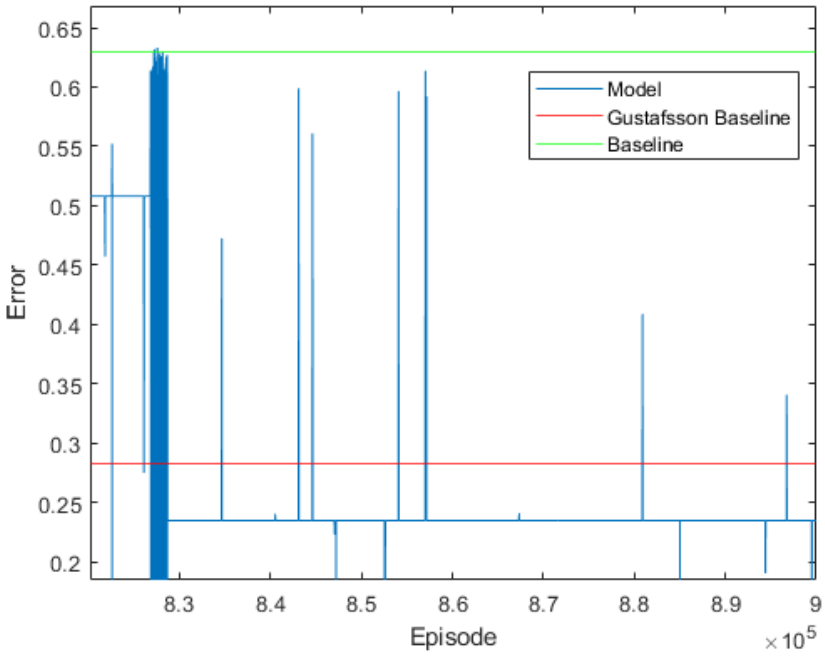


Figure 4.15: Error for Gustafsson Controller.

can adjust to allow for more fine tuning of the safety factor. But it is clear that improvement in error is accompanied by an increase in the number of steps. The best performing model was Alpha-Beta Adjust and Safety Factor Adjust because it has the best decrease in error and with minimal increase of steps.

Table 4.4: Comparison of models for dynamically tuning controller parameters to baseline

Model Method	Ratio (Baseline Error/ Model Error)	Difference (Baseline (266) – Model Steps)
Alpha-Beta	1.1247	-3
Safety Factor	1.1911	-3
Alpha-Beta Adjust	0.6289	1
Safety Factor Adjust	2.8772	-7
Alpha-Beta Safety Factor Adjust	0.8555	-24
Alpha-Beta Adjust and Safety Factor Adjust	5.2281	-7

If we choose the Alpha-Beta Adjust and Safety Factor Adjust and train it on different problems and then test, we can see how well it performs on the problems it has seen and has

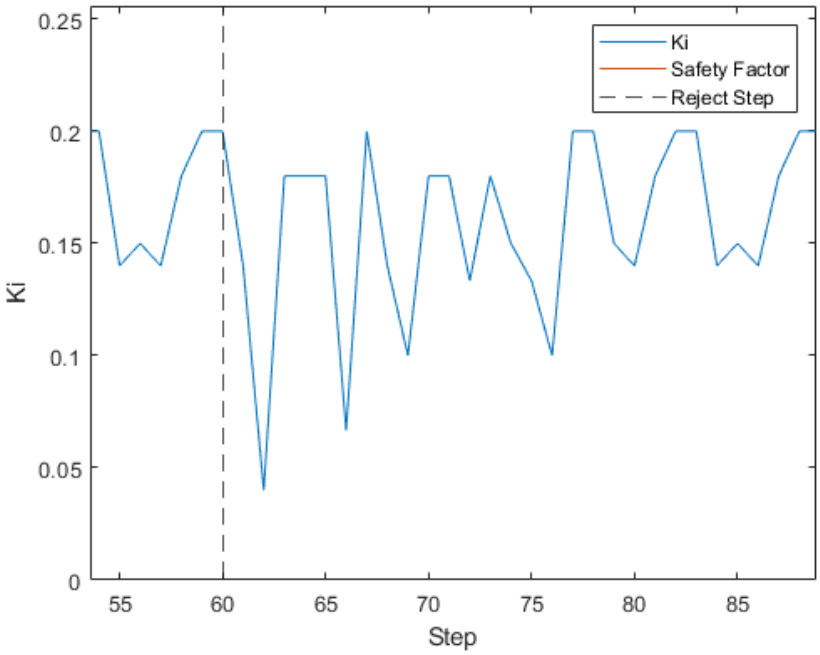


Figure 4.16: Gustafsson behavior.

not seen. Table 4.5 presents the ratio of the baseline error/model error, where >1 means the model was more accurate and < 1 means the baseline was more accurate. One item to note is that the baseline error of the Brusselator Periodic was very accurate at 0.0055; using the same controller with similar alpha and beta values, the accuracy was only 0.022. So the baseline Brusselator Periodic was a bit anomalous, which means most of the ratios are less than one, if we account for anomaly and replace 0.0055 with 0.022, any number above 0.25 means that the models performed better than the baseline. Training on the Brusselator Spiral problem yields good results on other problems. Overall, most of our models perform well on Lorenz 63, Brusselator Spiral, and accounting for fortuitousness of the Brusselator Periodic, it does fine with Lorenz 96. Although, the model never converges to a policy for Lorenz 96 which can be associated to the Lorenz 96 problem having 40 variables. The policy that trains on the Brusselator Spiral and Lorenz 63 seems to transfer well to each other, indicating that there is some potential in training on some problems and transferring them

to more complicated problems where we do not know the actual answer. Training on all the problems seems to limit the benefits of training and does not do as well as training on them as individual problems. That is a challenge because it makes a hybrid of "optimal" patterns to solve the problems and does not individually address them as well.

Looking at the number of steps (Table 4.6 and 4.7) the models take, our models usually take more steps than the baseline. However, for the most part, our models only take 10 percent more steps. The exception is the Brusselator Periodic, which actually takes less steps.

Table 4.5: Ratio of errors for all problems

Tested on:	**Ratio (Baseline Error/Model Error) when Trained on:**				
	All	**Brusselator Periodic**	**Brusselator Spiral**	**Lorenz 63**	**Lorenz 96**
Brusselator Periodic	0.1921	0.3388	0.7533	0.2999	0.1611
Brusselator Spiral	1.4910	0.7074	21.8551	3.2927	1.4944
Lorenz 63	1.2230	1.8980	1.7525	5.2281	0.6432
Lorenz 96	0.9030	1.0187	1.1589	0.9263	1.1822

Table 4.6: Ratio of steps between baseline and model for all problems

Tested on: (# baseline steps)	**Ratio (Baseline Steps/Model Steps) when Trained on:**				
	All	**Brusselator Periodic**	**Brusselator Spiral**	**Lorenz 63**	**Lorenz 96**
Brusselator Periodic (701)	1.0099	1.0441	1.0550	1.0335	0.9929
Brusselator Spiral (294)	0.8882	0.8472	0.9453	0.9046	0.8963
Lorenz 63 (266)	0.9500	0.9366	0.9602	0.9743	0.9779
Lorenz 96 (571)	0.8704	0.9376	0.9136	0.8935	0.9194

Table 4.7: Difference of steps between baseline and model for all problems

Tested on: (# baseline steps)	**Difference (Baseline Steps - Model Steps) when Trained on:**				
	All	**Brusselator Periodic**	**Brusselator Spiral**	**Lorenz 63**	**Lorenz 96**
Brusselator Periodic (701)	7	30	37	23	-5
Brusselator Spiral (294)	-37	-53	-17	-31	-34
Lorenz 63 (266)	-14	-18	-11	-7	-6
Lorenz 96 (571)	-85	-38	-54	-68	-50

Graph Figure 4.17 can be broken down into 4 quadrants by dividing both the error ratio and step ratio to be less than one, where the model performed worse than the baseline or greater than one, where the model performed better than the baseline. Thus, the upper right quadrant is where the models outperformed the baseline by both taking fewer steps and being

Figure 4.17: Comparing the step ratio to error ratio for all problems

more accurate, and the left lower quadrant is where the model performed worse than the baseline in terms of the number of steps and accuracy. In particular, majority of the models take fewer steps on the Brusselator Periodic problem than the baseline, but this comes at the cost of accuracy. Most of the models tested on Lorenz-96 have a similar accuracy to the baseline but require higher number of steps. On the opposite end, the models when tested on Lorenz-63 and Brusselator Spiral, the number of steps increased but the accuracy also increased. Overall, models trained on the same problem that they were tested on did better than the baseline in accuracy.

Chapter 5

Conclusions

In this work, Reinforcement Learning, using Q-Learning, is applied to the well-established alpha-beta and Gustafasson controllers to solve ordinary differential equations for the Lorenz 63 and Lorenz 96 chaos problems and the chemical reaction Brusselator Periodic and Brusselator Spiral problems. The Reinforcement Learning/Q-Learning solutions are compared to baseline results when the controller has constant values of alpha = 0.9 and beta = 0.1.

Overall, the reward function that has the most success is when the step is accepted with a reward value of error/1.5, and the step is rejected with a reward value of 0. The adjustment of the safety factor method adds to the performance of the model by allowing for more fine tuning while the model is solving the problems. In terms of accuracy and minimal effect on step count, the alpha beta adjustment with safety factor adjustment model is the best.

An average increase of 5.5% in step size and an average increase of 132% in accuracy as compared to the baseline model are observed across all problems. If we account for the anomaly of the baseline answer on the Brusselator periodic problem and the runs where the model is tested and trained on the same problem we get a 38.3% improvement of accuracy over the baseline model.

The learned policies usually tested well on similar problems such as the performance of the Brusselator Spiral based learned policy when tested on Lorenz 63 and vice versa. On the other hand, the learned policies do not always scale well when problems are drastically

different. Such as any model performance of the learned policies when tested on Lorenz 96.

Some patterns emerge in the learned policy from different models, such as the idea of a delayed reaction after getting a reject; the controller changes from its normal course of action only after taking several steps after a reject rather than changing the course of action the step immediately after the reject. Another such pattern is that after a rejection, the model increases the safety factor because the controller always decreases the step size. These patterns suggest that reinforcement learning based models are responsive to changes allowing for a dynamically adaptive model.

Advantages of the Reinforcement Learning, using Q-Learning, for solving these ODEs are:

- Inputs do not require exact solutions so it is trainable on more complex problems.
- There is increased accuracy, as measured by global error, for the solution.
- Q learning does not add any significant computational overhead after training.

Disadvantages:

- Model solutions took more steps than baseline, although sometimes just a small amount.
- Up-front cost to develop models.
- Q-Learning is a discrete method, but controllers have continuous inputs, causing a mismatch.

The future works can be approached from multiple directions. One direction is to continue using Q learning with discrete states and actions, but additionally allowing for the model to choose between preset controllers at each time step. The preset controllers can expand from the list used in this work to include controllers that use implicit methods. This makes the

model even more dynamic. Another direction is to use a more sophisticated model that can handle continuous states and actions such as Deep Q-learning. This eliminates the mismatch between the inputs and outputs of the controller being continuous while the model states and actions being discrete as seen in Q-learning.

Bibliography

[1] anonymous user with IP 163.1.81.7. Plot of 3 variables in a lorenz 96 model simulation with 36 variables and f=8. https://commons.wikimedia.org/wiki/File:Lorenz_96.svg, November 2017. Accessed: 2021-07-20; CC BY-SA 3.0.

[2] António Miguel de Campos. chaos in lorenz attractor. https://en.wikipedia.org/wiki/Lorenz_system, March 2006. Accessed: 2021-07-20; Public Domain.

[3] Lennart Edsberg. *Introduction to computation and modeling for differential equations*. John Wiley & Sons, Inc, Hoboken, New Jersey, second edition edition, 2016. ISBN 978-1-119-01846-9 978-1-119-01845-2.

[4] Isaac R. Galatzer-Levy, Kelly V. Ruggles, and Zhe Chen. Data Science in the Research Domain Criteria Era: Relevance of Machine Learning to the Study of Stress Pathology, Recovery, and Resilience. *Chronic Stress*, 2:247054701774755, January 2018. ISSN 2470-5470, 2470-5470. doi: 10.1177/2470547017747553. URL http://journals.sagepub.com/doi/10.1177/2470547017747553.

[5] Kjell Gustafsson. *Control of Error and Convergence in ODE Solvers*. PhD thesis, Lund Institute of Technology, Department of Automatic Control, 1992. URL http://scholar.google.com/scholar_url?url=https://portal.research.lu.se/portal/files/4871604/8566348.pdf&hl=en&sa=X&ei=E0DrYJWeAZqEy9YP8qe2oAc&scisig=AAGBfm3k9KNIMoFzNY_dc_EWma0JblgG_w&nossl=1&oi=scholarr.

[6] Ernst Hairer, Syvert P Nørsett, and Gerhard Wanner. *Solving ordinary differential equations. 1, Nonstiff problems*. Springer-Vlg, 1993.

[7] Edwin Atlee Jackson. *Perspectives of nonlinear dynamics. 2.* Cambridge Univ. Pr, Cambridge, 1994. ISBN 978-0-521-42633-6 978-0-521-35458-5.

[8] Jkrieger. solution of brusselator differential equations for different conditions. https://commons.wikimedia.org/wiki/File:Bruesselator.svg, August 2012. Accessed: 2021-07-20; CC BY 3.0.

[9] Edward N. Lorenz. Deterministic nonperiodic flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, March 1963.

[10] Edward N. Lorenz. Predictability – a problem partly solved. In Tim Palmer and Renate Hagedorn, editors, *Predictability of Weather and Climate*, pages 40–58. Cambridge University Press, Cambridge, 2006. ISBN 978-0-511-61765-2. doi: 10.1017/CBO9780511617652.004. URL https://www.cambridge.org/core/product/identifier/CBO9780511617652A010/type/book_part.

[11] E.N. Lorenz. Predictability a problem partly solved. In *Proc. Seminar on predictability*, volume 1, pages 1–18, Reading, Berkshire, UK, 1995. ECMWF. Pages: 1-18 Place: Shinfield Park, Reading Publisher: ECMWF Volume: 1.

[12] Andreas C. Müller and Sarah Guido. *Introduction to machine learning with Python: a guide for data scientists.* O'Reilly Media, Inc, Sebastopol, CA, first edition edition, 2016. ISBN 978-1-4493-6941-5. OCLC: ocn895728667.

[13] I. Prigogine and R. Lefever. Symmetry Breaking Instabilities in Dissipative Systems. II. *The Journal of Chemical Physics*, 48(4):1695–1700, February 1968. ISSN 0021-9606, 1089-7690. doi: 10.1063/1.1668896. URL http://aip.scitation.org/doi/10.1063/1.1668896.

[14] Colin Sparrow. *The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors*,

volume 41 of *Applied Mathematical Sciences*. Springer New York, New York, NY, 1982. ISBN 978-0-387-90775-8 978-1-4612-5767-7. doi: 10.1007/978-1-4612-5767-7. URL http://link.springer.com/10.1007/978-1-4612-5767-7.

[15] Steven H. Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering*. Westview Press, a member of the Perseus Books Group, Boulder, CO, second edition edition, 2015. ISBN 978-0-8133-4910-7. OCLC: ocn842877119.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

[17] Hilber Traum. Slopes used by the classical runge-kutta method (rk4). https://commons.wikimedia.org/wiki/File:Runge-Kutta_slopes.svg, November 2017. Accessed: 2021-07-20; CC BY-SA 4.0.

[18] Dirk L. van Kekem. *Dynamics of the Lorenz-96 model: Bifurcations, symmetries and waves*. PhD thesis, University of Groningen, 2018. URL https://research.rug.nl/en/publications/dynamics-of-the-lorenz-96-model-bifurcations-symmetries-and-waves.

[19] Dirk L. van Kekem and Alef E. Sterk. Symmetries in the Lorenz-96 Model. *International Journal of Bifurcation and Chaos*, 29(01):1950008, January 2019. ISSN 0218-1274, 1793-6551. doi: 10.1142/S0218127419500081. URL https://www.worldscientific.com/doi/abs/10.1142/S0218127419500081.

[20] Mohan Yogeswaran and S. G. Ponnambalam. Reinforcement learning: exploration–exploitation dilemma in multi-agent foraging task. *OPSEARCH*, 49(3):223–236,

September 2012. ISSN 0030-3887, 0975-0320. doi: 10.1007/s12597-012-0077-2. URL http://link.springer.com/10.1007/s12597-012-0077-2.