Fourier Adaptive Learning for Control

Mason Lee

December 16, 2023

Abstract

This work delves into the domain of adaptive control through the lens of Fourier analysis, concentrating on the challenges posed by turbulent environments where reinforcement learning is traditionally employed. Conventional neural network models, despite their powerful capabilities, frequently encounter difficulties such as overfitting, limited observability, and hindered generalizability. To navigate these challenges, we implement the recently successful Fourier Adaptive Learning for Control (FALCON) algorithm, an approach that synthesizes Fourier methods with control theory. Our implementation is unique in its integration of L1 and L2 regularization strategies within the reinforcement learning framework to foster sparsity and precision in the learned models. Additionally, we incorporate a customized Cross-Entropy Method to suit the FALCON structure, enhancing the model's ability to handle the intricacies of dynamic systems. The paper presents a comprehensive suite of evaluation tools, offering detailed insights into the model's performance, stability, and adaptability. We hope to set the framework for exploration of more complex turbulent environments.

1 Introduction

1.1 Learning the Dynamics of Turbulent Environments

The study and control of highly turbulent environments, characterized by changing underlying dynamics such as fluctuating Reynolds numbers in Navier-Stokes systems, present formidable challenges. These environments, often marked by partial observability and chaotic behavior, necessitate a nuanced understanding and approach. In scenarios like drone navigation through varying atmospheric conditions or flow separation over airfoils, the unpredictability and complexity of these environments are particularly salient. Such settings are not only dynamic but also prone to rapid changes, where even minor alterations in actions can lead to divergent state spaces. This is further compounded in applications like the active control of airplane wings, where precision is paramount for optimizing lift and drag.

In these turbulent environments, the rewards for successful control are often realized on short time scales, emphasizing the need for agile and responsive control systems. Traditional methods have increasingly incorporated advanced computational techniques, with a significant focus on neural networks. However, the integration of modern neural networks, especially in physics-based learning contexts, has been met with substantial hurdles. The inherent unpredictability and complexity of these environments, coupled

with the limitations of conventional neural network architectures, pose significant challenges. These challenges include not only mastering the immediate control requirements but also adapting to rapidly changing conditions, all within the constraints of partial observability and the potential for chaotic dynamics.

1.2 The Challenges of Deep Learning for Physics Governed Environments

A prominent approach in this area is physics-informed machine learning, often implemented within the deep learning paradigm. This method integrates physics equations to constrain the optimization landscape in loss functions, adding physics-based penalties as a form of guidance [5]. Additionally, network architectures are designed to inherently reflect physical principles [2]. Despite these advances, the black-box nature of neural networks and their feature memory characteristics present considerable challenges. Neural networks, as per the universal approximation theorem [1], can approximate arbitrarily complex functions. However, their training structure, typically reliant on stochastic gradient descent methods like ADAM, is not well-suited for dynamic retraining. These methods usually follow learning schedules aimed at progressively reducing entropy and settling into local optima. Consequently, if the optimization landscape shifts during training, neural networks struggle to adapt, especially quickly.

To address these issues, methods like DeepONet [3] have been developed. DeepONet exhibits the capacity to learn the dynamics of multi-physics environments through a trunk-branch processing technique, which encodes a broad range of operators. While this approach is highly expressive and offers robust generalizability, it suffers from long training times. Expanding the physics knowledge domain essentially translates into linearly training these networks independently. Furthermore, the structure of these operator networks highlights the difficulty in partitioning learning across various environments. This challenge is compounded by the necessity to understand a priori the governing parameters of these environments, such as Reynolds number or viscosity, particularly in turbulent contexts where environments are often partially observable and unpredictable [4].

Moreover, traditional neural network ensembles, while effective in certain contexts, lack generalizability constraints, which is a critical requirement in turbulent control scenarios. Additionally, without stability constraints, neural networks can lead to exploding trajectories. Such unconstrained learning can result in unexplored state spaces, unstable oscillations, and potentially hazardous control patterns. Implementing stability constraints in neural networks is often a complex and arduous task, underscoring the need for novel approaches that inherently incorporate these considerations.

1.3 Reinforcement Learning for Control

In the realm of Reinforcement Learning (RL), both model-free and model-based methods, particularly those employing gradient descent policy optimization, face significant hurdles in maintaining expressiveness within dynamically changing environments. A subset of Model-Based Reinforcement Learning (MBRL) is Model Predictive Control (MPC), and in this paper, we specifically focus on utilizing the cross-entropy method within MPC frameworks [6].

The introduction of Fourier Adaptive Learning for Control (FALCON) emerges as a promising solution to these challenges. As outlined by [4], upon whose work this paper

heavily relies, FALCON represents a significant stride forward in addressing the limitations of traditional neural networks and RL methods in turbulent control scenarios. This paper primarily serves as an implementation and validation of the FALCON methodology for educational purposes.

FALCON operates through a structured approach, beginning with a warm-up phase, transitioning into a phase employing LASSO (Least Absolute Shrinkage and Selection Operator) for a sparse Fourier basis representation, and culminating in an active control phase. In this active phase, a dynamics model is utilized to simulate the environment internally. The zeroth-order cross-entropy method plays a pivotal role in this simulation, enabling effective environment modeling and control decision-making.

A critical component of FALCON is its transformation of the state space into a Fourier basis representation. This transformation allows for the application of established control theory and stability concepts from existing literature, facilitating the recovery of dynamic control guarantees. Additionally, FALCON addresses the challenge of injecting bias into learning processes. By normalizing the high and low-frequency nodes, it ensures a more balanced and stable control mechanism.

Regarding stability criteria, while not delving into exhaustive details, this paper will touch upon the convergence and stability of the model, drawing from the mathematics of dynamical systems and the learning of Lyapunov functions. These aspects underscore FALCON's capacity to maintain stability and reliability in control applications.

The primary purpose of this paper is twofold. Firstly, it aims to implement FALCON using the cross-entropy method in Python, exploring the system's utility and understanding the underlying mathematics, design decisions, and stability criteria. Secondly, it seeks to provide a user-friendly framework for comprehending turbulence physics within RL. This includes examining how different parameter tunings and regularization strategies impact the expressiveness and control capabilities of the system. Ultimately, this paper endeavors to bridge the gap between theoretical knowledge and practical application in turbulent control environments, offering insights and tools for researchers and practitioners alike.

In light of these challenges, this paper presents the Fourier Adaptive Learning for Control (FALCON) approach. FALCON leverages the Fourier transform's ability to represent complex systems and dynamics in the frequency domain, offering a more robust and interpretable framework for RL in turbulent environments. By transforming the state space into a Fourier basis representation and introducing frequency-domain constraints, FALCON aims to mitigate issues like overfitting and improve the generalizability of the model.

2 Methods

2.1 Modeling Dynamics with Lagged Time Delay and NARX Models

In the initial phase of our approach, we utilize lagged time delay and Nonlinear AutoRegressive models with eXogenous inputs (NARX) to model the dynamics of the system. This involves creating subsequences of input-output pairs over a specified time window. Specifically, for a window of length T_w, we generate $T_w - h + 1$ subsequences, denoted as $s_i = [y_{i-1}^\top, \ldots, y_{i-h}^\top, u_{i-1}^\top, \ldots, u_{i-h}^\top]^\top \in \mathbb{R}^{h(d_y+d_u)}$, for $h \leq i \leq T_w$. The system dynamics

are then represented as $y_t = F(s_t) + e_t$, where F is the unknown nonlinear function to be estimated.

2.2 Fourier Series and Fast Fourier Transform

To approximate the nonlinear function F, we employ the Fourier series expansion. The Fourier representation transforms the state space into a frequency domain, providing a compact and efficient representation of the system dynamics. The n-th order Fourier expansion of F leads to a D-dimensional Fourier series representation of all s_t, given by $\phi(s_t)$. The Fourier series expansion of a function $f(x)$ can be expressed as:

$$f(x) = a_0 + \sum_{n=1}^{\infty} [a_n \cos(nx) + b_n \sin(nx)] \tag{1}$$

where a_n and b_n are the Fourier coefficients. The Fast Fourier Transform (FFT) algorithm is used for efficient computation of these coefficients.

2.3 Theta Matrix and Fourier Basis Combination

In our model, we represent the system dynamics F as $y_t \approx \Theta^\top \cdot \phi(s_t) + e_t$, where $\Theta^* \in \mathbb{R}^{D \times d_y}$ is the matrix of unknown coefficients. This representation captures the dynamics in the Fourier basis, combining the efficiency of linear models with the expressiveness of nonlinear dynamics.

2.4 The Warm-Up Phase

The warm-up phase of FALCON is a critical preparatory step, aimed at gathering preliminary data about the system to facilitate effective control in later stages. During this phase, the focus is on safely exploring the system for a duration of T_w time steps. For this exploration, FALCON typically employs isotropic Gaussian inputs, $u_t \sim \mathcal{N}(0, \sigma_u I)$, although for safety-critical tasks, smoother or safer exploration techniques may be used, such as time-correlated inputs or known safe nominal controllers augmented with isotropic excitations.

An integral part of the warm-up phase is the improved basis selection process. As the order of the Fourier basis increases, the dimensionality D grows exponentially with the system's dimension, which can lead to computational burdens for large h in higher-order NARX models. To address this, FALCON applies an ℓ_1-constrained Fourier basis and the Least Absolute Shrinkage and Selection Operator (Lasso) method. Specifically, instead of generating bases $\omega_i s$ for all n, only ℓ_1-constrained bases ($\|\omega_i\|_1 \leq n$) are considered, reducing the number of basis vectors from $1 + 2nh(d_y + d_u)$ to $2D'$ where $D' = h(d_y + d_u) + n\frac{n!}{n}$.

2.5 The Warm-Up Phase

During the warm-up phase, FALCON forms the feature representations $\phi(s_i)$ from the collected samples and solves the Lasso problem to optimize the Θ matrix:

$$\min_{\Theta} \frac{1}{2T_w} \|Y_{T_w} - \Theta^\top \Phi_{T_w}\|_F^2 + \alpha \|\Theta\|_1 \tag{2}$$

where $\alpha > 0$ is a regularization parameter that influences the sparsity of the learned Θ^*. A higher α results in a sparser matrix, reducing the number of non-zero coefficients in Θ^* and thus the number of basis vectors D used in subsequent model learning. This L1 optimization, occurring exclusively after the warm-up, refines the basis set, reduces computational requirements, and decreases the data needed for learning dynamics.

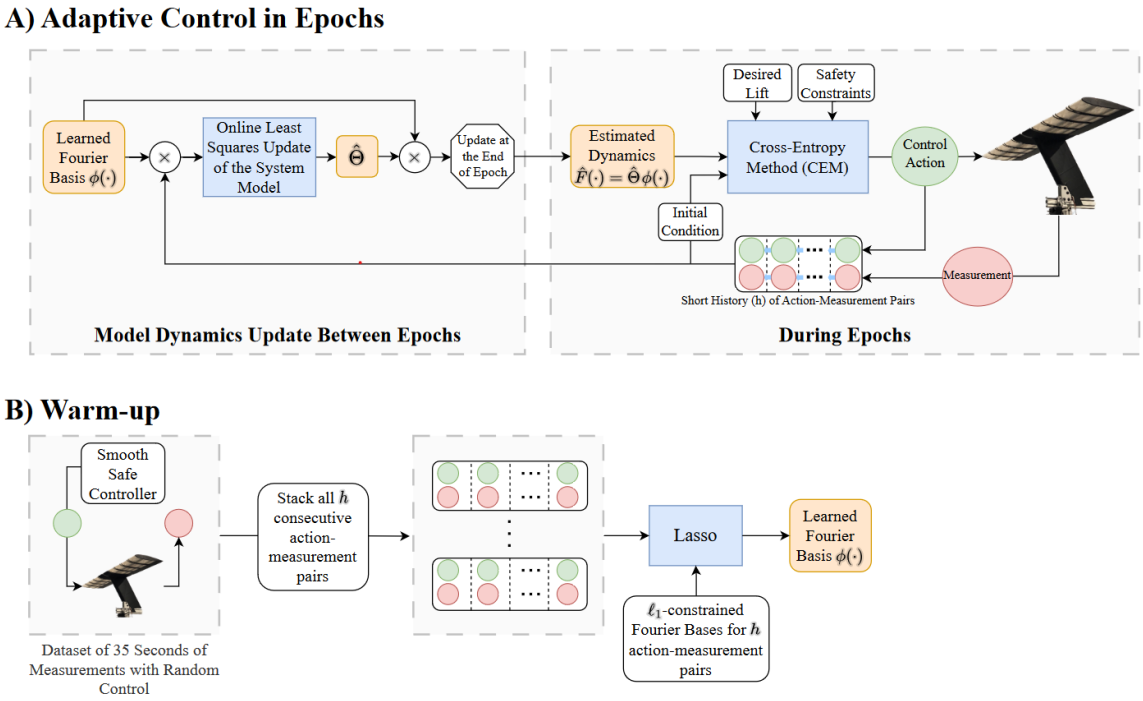

Figure 1: Illustration of the FALCON Framework. The framework is divided into two primary phases: the Warm-Up phase and the Adaptive Control in Epochs phase. In the Adaptive Control phase (A), FALCON characterizes the system dynamics using a linear map derived from a concise history of action-measurement pairs, represented in a Fourier basis established during the Warm-Up phase. The framework continuously refines its understanding of the system dynamics by updating the linear coefficients through online least squares, employing the Cross-Entropy Method (CEM) for control under extreme turbulence conditions, while adhering to specified lift and safety requirements. The Warm-Up phase (B), a preliminary 35-second process, is dedicated to gathering initial data about the system. This data is used to construct subsequences of action-measurement pairs and to identify a relevant Fourier basis via the Lasso problem, which FALCON then utilizes for learning and control throughout the adaptive control epochs.

2.6 Active Control Phase with L2 Regression and Cross Entropy Method

Once FALCON has an estimated model from the warm-up phase, where the Θ matrix is optimized using L1 regression for sparsity, it transitions to the active control phase. In this phase, FALCON employs L2 regression for continuous refinement of the Θ matrix, now focusing on accuracy and stability rather than sparsity. The L2 regression is formulated as:

Algorithm 1 FALCON Algorithm

Input: $T_w, h, tep, \tau, D, C_{0:T}$
Initialization Phase:
for $t = 1$ **to** T_w **do**
    Conduct an exploration with control action u_t
    Store the results as $D_0 = \{y_t, u_t\}_{T_w}^{t=0}$
    Form the state s_t from recent observations and actions using D_0
    Compute a Fourier representation of s_t based on equation
    Determine the most meaningful patterns in the representation using Lasso
end for
Adaptive Control Phase:
while True **do**
    Estimate model parameters $\hat{\Theta}_i$
    Create the fourier basis function $\hat{F}_i(\cdot)$ from these parameters
    for $t = T_w + (i-1) \times tep + 1$ **to** $T_w + i \times tep$ **do**
        Decide control action u_t using the function $\hat{F}_i$ and recent observations/actions
        Make an observation y_{t+1}
        Update the state s_{t+1} and its Fourier representation
    end for
end while

$$\min_{\Theta} \lambda \|\Theta\|_F^2 + \|Y_{T_w} - \Theta^\top \Phi_{T_w}\|_F^2 \tag{3}$$

for $\lambda > 0$. Here, $Y_t = [y_t, \ldots, y_h] \in \mathbb{R}^{d_y \times (t-h+1)}$ and $\Phi_t = [\phi(s_t), \ldots, \phi(s_h)] \in \mathbb{R}^{D \times (t-h+1)}$. The solution $\hat{\Theta}_1$ is used to estimate the system dynamics $\hat{F}_1(s) = \hat{\Theta}_1^\top \phi(s)$, and this estimation process is repeated at the beginning of each epoch with the latest data.

For controlling the system, FALCON utilizes the Cross Entropy Method (CEM), a sampling-based (zeroth order) Model Predictive Control (MPC) policy. CEM solves the optimization problem at each time step, maintaining a distribution (typically Gaussian) to sample action sequences. It iteratively refines this distribution, focusing on lower-cost sequences as informed by the estimated dynamics. The steps of CEM are as follows:

CEM is memoryless, solving an optimization problem at each time step to explore the state space and adaptively sample actions. The method's iterative nature allows it to adapt to the changing dynamics of the system, making it suitable for environments where small changes can lead to significant differences in outcomes. This combination of the updated Θ matrix through L2 regression and the adaptive, explorative nature of CEM forms the crux of FALCON's control strategy in dynamic, turbulent environments.

3 Regret Minimization Framework

In the context of FALCON, the concept of regret minimization plays a crucial role in guiding the learning behavior of the agent. Regret minimization refers to the difference in performance between the learning agent and an optimal controller over a period. The goal is to achieve sublinear regret, which implies that the agent's performance approaches that of the optimal controller as more data is collected.

Algorithm 2 Cross Entropy Method (CEM)

Require: $\tau, K, M, 0 < \gamma < 1, N, \sigma_{init}, \hat{F}, y_{t:t-h+1}, u_{t-1:t-h+1}, C_{t:t+\tau-1}, \mathcal{N}(\mu, \sigma^2 I)$

for $i = 1, 2, \ldots, M$ **do**
  if $i = 1$ **then**
    Set μ to the best action sequence from the previous time-step (Warm-Start)
    Set $\sigma = \sigma_{init}$
  end if
  Sample $K\gamma^{i-1}$ action sequences u_j of length τ using $\mathcal{N}(\mu, \sigma^2 I)$
  Compute trajectories using $\hat{F}$ with initial conditions $y_{t:t-h+1}, u_{t-1:t-h+1}$
  Compute costs of each trajectory using $C_{t:t+\tau-1}$
  Select the best N action sequences based on costs
  Update μ and σ to fit the Gaussian distribution to the best N sequences
end for
Execute the first action of the best sequence from the final iteration

3.1 Definition of Regret

The regret $R(T)$ after T time steps is defined as the cumulative difference in the cost or reward between the agent and an optimal strategy. It can be formally expressed as:

$$R(T) = \sum_{t=1}^{T} \left(c_t(a_t) - c_t(a_t^*) \right) \tag{4}$$

where $c_t(a_t)$ is the cost incurred by the agent for choosing action a_t at time t, and $c_t(a_t^*)$ is the cost for the optimal action a_t^* at the same time.

3.2 Sublinear Regret

Achieving sublinear regret, $R(T) = \mathcal{O}(T^\alpha)$ with $0 \leq \alpha < 1$, is desirable as it indicates that the average regret per time step diminishes as T increases. In other words, the learning agent is effectively improving its performance relative to the optimal controller. This is mathematically represented as:

$$\lim_{T \to \infty} \frac{R(T)}{T} = 0 \tag{5}$$

3.3 Balancing Exploration and Exploitation

The balance between exploration (trying new actions to discover their effects) and exploitation (using known actions that yield high rewards) is critical in achieving sublinear regret. The trade-off can be summarized as follows:

- **Excessive Exploration**: Leads to linear regret due to insufficient exploitation of known good actions.
- **Insufficient Exploration**: Also leads to linear regret due to getting stuck in sub-optimal policies without discovering potentially better actions.

4 Results

4.1 FALCON Design Formulation Analysis

To thoroughly understand the regularization mechanism employed in FALCON, we delve into the transformation of L2 regression into a batch update process. This transformation plays a crucial role in injecting bias into the model, particularly in the regularization of Fourier frequencies, thus shaping the model's behavior.

Transformation from L2 Regression to Batch Updates: The L2 regression in FALCON is initially formulated as equation (3). However, for large dimensions D, solving this equation directly can be computationally intensive and prone to numerical errors. To address this, FALCON adopts a recursive, or batch, update approach. In each epoch, the model estimate $\hat{\Theta}_{i,t}$ and the inverse design matrix V_{t-1}^{-1} are updated based on the new data:

$$\hat{\Theta}_{i,t} = \hat{\Theta}_{i,t-1} + V_{t-1}^{-1}\phi(s_t)(y_t - \hat{\Theta}_{i,t-1}^{\top}\phi(s_t))^{\top}\frac{1}{1+\phi(s_t)^{\top}V_{t-1}^{-1}\phi(s_t)} \tag{6}$$

The L2_batch update equation (6) is central to FALCON's approach to updating model parameters over time. Let's break down the intuition behind each term:

- V_{t-1}^{-1}: The inverse design matrix, crucial for regularization. It dampens the impact of high-variance components and stabilizes the learning process.
- y_t: The target output at time t. This is what the model is trying to predict.
- $\hat{\Theta}_{i,t-1}^{\top}\phi(s_t)$: This term represents the model's prediction at time $t-1$, providing a basis for adjustment in the next iteration.
- Error term $(y_t - \hat{\Theta}_{i,t-1}^{\top}\phi(s_t))^{\top}$: The difference between the actual output and the model's prediction. This error guides the update of the model parameters.
- $\frac{1}{1+\phi(s_t)^{\top}V_{t-1}^{-1}\phi(s_t)}$: A normalization term that ensures the updates are scaled appropriately, preventing overly large adjustments that could destabilize the learning process.

Regularization through the Inverse Design Matrix: The inverse design matrix V_{t-1}^{-1} is a pivotal component in this regularization process:

$$V_{t-1}^{-1} = (\Phi_t\Phi_t^{\top} + \lambda I)^{-1} \tag{7}$$

This matrix serves as a regularizer by adding a small constant λ to the diagonal elements of $\Phi_t\Phi_t^{\top}$. In the frequency domain, this regularization slightly penalizes all frequencies, preventing any single frequency from dominating due to overfitting. Consequently, this bias towards smaller Θ values dampens the impact of higher-variance frequencies in the Fourier basis, focusing the model more on stable frequencies with lower variance.

In FALCON, the regularization term λ in the inverse design matrix plays a pivotal role in injecting bias into the model. This term affects the model's interpretation of the Fourier frequencies. Let's break down the impact of this regularization on the Fourier basis:

- The Fourier basis $\phi(s_t)$ represents the state s_t in the frequency domain. Each frequency component in this representation contributes to modeling the system's dynamics.
- The inverse design matrix, defined as $V_{t-1}^{-1} = (\Phi_t \Phi_t^\top + \lambda I)^{-1}$, adds a regularization term λI to the covariance of the Fourier basis. This addition penalizes each frequency component to some extent.
- The penalty is more significant for frequencies with higher variance. In the context of turbulent environments, higher variance frequencies might correspond to chaotic or less predictable aspects of the system dynamics. The regularization term λI adds a constant λ to the diagonal of $\Phi_t \Phi_t^\top$, which alters the eigenvalues of the covariance matrix of the Fourier basis. This alteration results in a penalization of the eigenvalues, with a greater impact on those associated with higher variance.
- By penalizing these higher variance components, the model inherently biases towards more stable, lower variance frequencies. These stable frequencies are presumed to encapsulate the core dynamics of the system, which is crucial in turbulent environments where predictability and stability are key.

Impact on Turbulent Environments: In turbulent environments, such as those encountered in fluid dynamics or aerodynamics, the ability to focus on stable, predictable patterns is crucial. The regularization strategy employed in FALCON:

- Helps the model to ignore or downplay chaotic, high-variance components which are often less reliable for prediction and control.
- Enhances the model's focus on the stable components of the system, which are more indicative of the underlying physics and more useful for reliable control decisions.
- Contributes to the overall stability and reliability of the control system, crucial in dynamic, potentially chaotic environments.

4.2 Analysis of FALCON Implementation Outcomes

4.2.1 Continuous Pendulum Environment Visualization

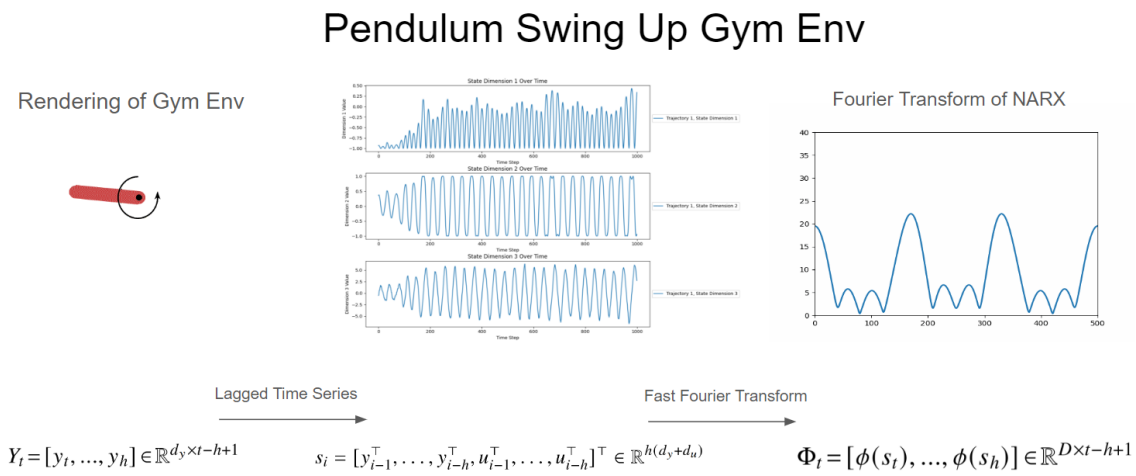

Figure 2: Continuous Pendulum Swing Up Gym Environment and Fourier Transform Analysis. The top panel illustrates the rendering of the Gym environment, depicting the continuous pendulum's state over time. The bottom left shows the lagged time series for the state dimensions, capturing the evolution of the system's states. The bottom right provides a visualization of the Fast Fourier Transform (FFT) applied to the NARX model, revealing the dominant frequencies that characterize the system's dynamics. This figure underscores the complexity of the environment, highlighting the Fourier basis's ability to encapsulate the pivotal frequencies that drive the system's behavior.

Figure 2 demonstrates the effective use of Fourier series to translate time-domain dynamics into the frequency domain. The FFT analysis of the NARX model elucidates the prominent frequencies that govern the pendulum's behavior, providing insights into the cyclical patterns inherent to the system. The lagged time series plots offer a temporal perspective, essential for understanding the initial state formation and subsequent prediction through the Fourier basis. This synergy between time and frequency-domain analyses forms the backbone of the FALCON approach, aiming for a comprehensive understanding and control of complex, dynamic systems like the continuous pendulum swing-up task.

4.2.2 Grid Search Performance Metrics

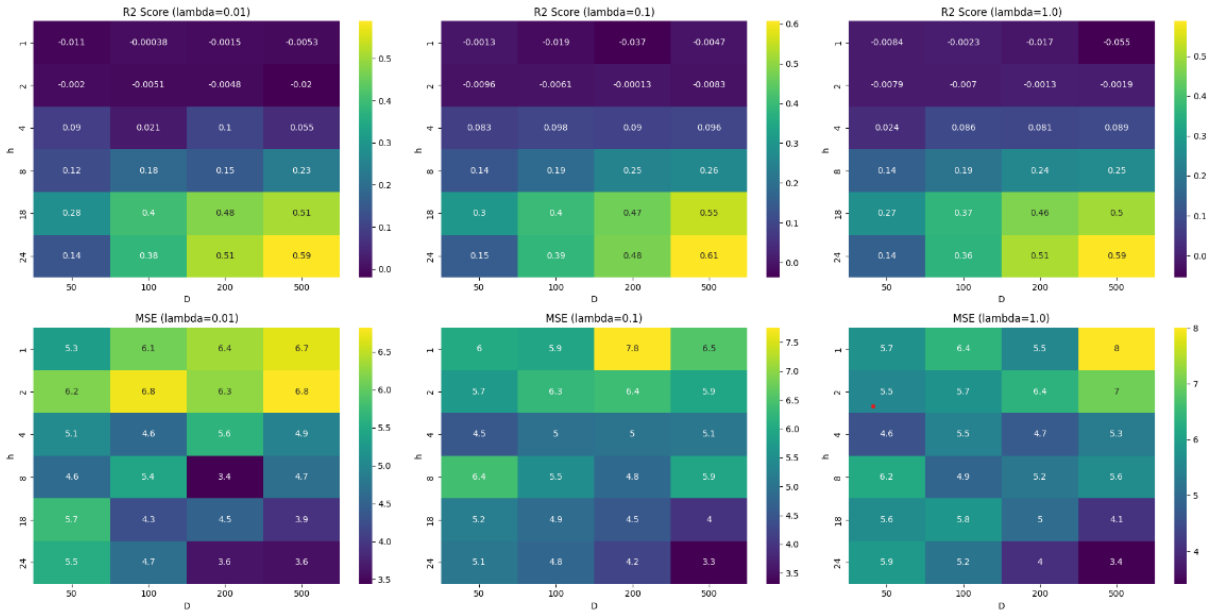

Figure 3: Performance Metrics from Grid Search Optimization. The figure presents six heatmaps, three for the R^2 score and three for Mean Squared Error (MSE), each corresponding to different regularization strengths (λ) across varying dimensions of the Fourier basis (D) and noise levels (σ). These metrics provide an in-depth analysis of the model's predictive accuracy, with the R^2 scores indicating the proportion of variance captured by the model and the MSE representing the average squared difference between the estimated values and the actual values. The R^2 score heatmaps show that higher values of D generally lead to better model performance, particularly at lower noise levels, as indicated by the increasing yellow intensity. Conversely, the MSE heatmaps reveal that lower D values and higher noise levels result in higher errors, as depicted by the darker regions. This comprehensive grid search optimization allows for the fine-tuning of FALCON's parameters to strike a balance between model complexity and performance, adapting to the noise inherent in turbulent environments.

In Figure 3, the optimization process's outcomes are visualized, depicting how different parameters affect the FALCON algorithm's performance. The R^2 score heatmaps highlight the variability of model performance with changes in the dimensionality of the Fourier basis and noise levels, providing a nuanced understanding of the model's predictive capabilities. Similarly, the MSE heatmaps showcase the impact of these parameters on the model's precision, reflecting the trade-offs between dimensionality and noise robustness. These visualizations underscore the significance of parameter tuning in achieving optimal performance, particularly in the context of learning in turbulent environments where the complexity of the system dynamics necessitates careful consideration of model capacity and regularization.

4.2.3 Fourier Basis and Model Weight Analysis

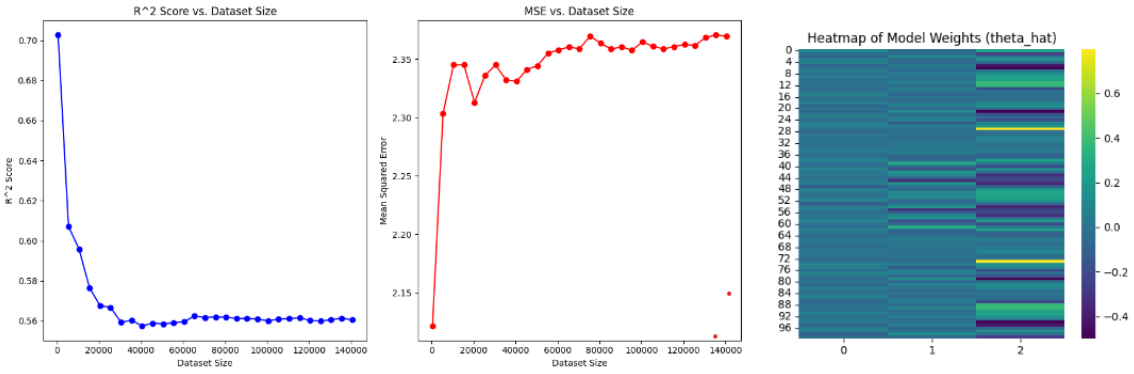

Figure 4: Correlation Between Dataset Size and Model Performance Indicators. This figure combines line graphs and a heatmap to present the relationship between dataset size and model performance metrics such as R^2 score and Mean Squared Error (MSE), along with the heatmap of model weights (theta_hat). The line graphs show a positive correlation between dataset size and R^2 score, suggesting that as more data is incorporated, the model's performance improves, capturing a greater variance of the system's dynamics. On the contrary, MSE fluctuates with the increase in dataset size, indicating the complexity of achieving model precision. The heatmap illustrates the model weights, providing a visual representation of their magnitudes and sign, which are crucial for understanding the contribution of each Fourier basis function to the predicted state. The distribution of weights across the basis functions reflects the underlying structure and dynamics captured by the model, where the color intensity represents the strength of the weights, thus informing the model's decision-making process.

Figure 4 encapsulates the model's learning trajectory as it assimilates data and refines its parameters. The R^2 score's improvement with dataset size augmentation signifies the model's growing capacity to encapsulate the system's dynamics. The MSE's oscillation highlights the challenges in model precision, an expected phenomenon in complex systems where noise and nonlinearity prevail. The heatmap of model weights (theta_hat) offers a visual assessment of how the Fourier basis functions' significance evolves, portraying the adaptive learning process as the model weights adjust to better represent the dynamics of the turbulent environment. Collectively, these visualizations furnish a comprehensive perspective on the model's learning progression, emphasizing the critical role of data in refining the model for optimal control in turbulence-dominated scenarios.

4.2.4 Stability of Rollouts

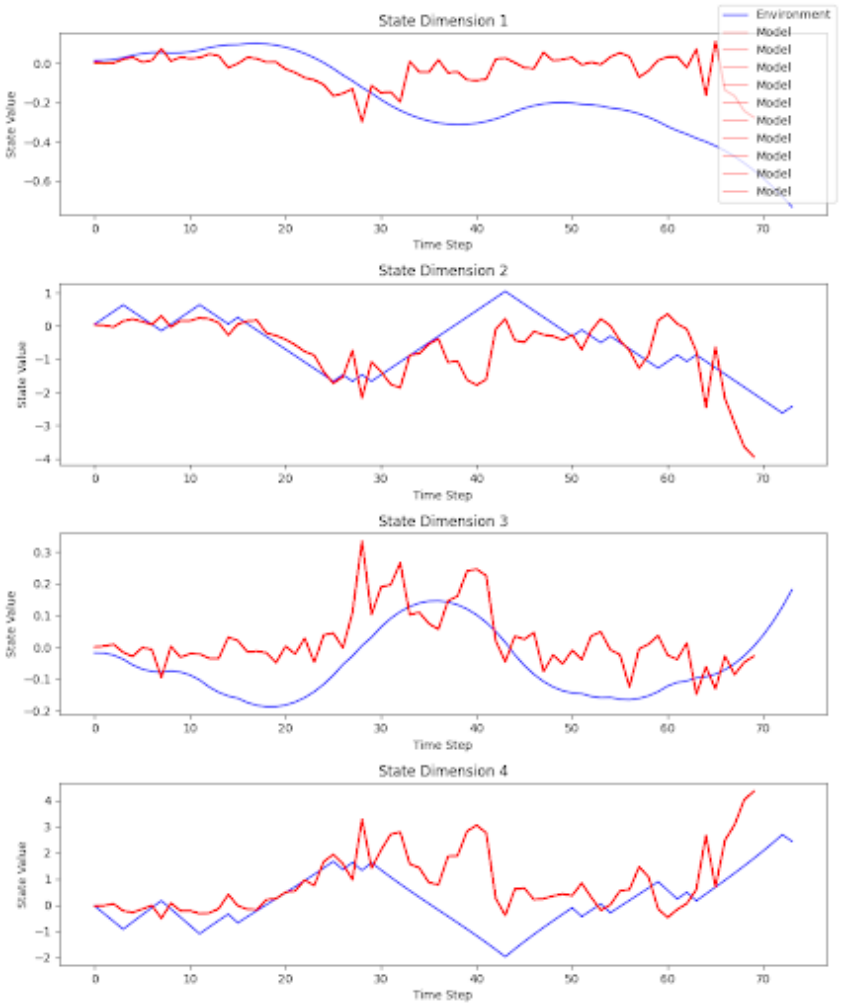

Figure 5: Trajectory Rollout Stability in the Cartpole Environment. The figure depicts the trajectory of state dimensions over time, contrasting the actual environment states (in red) with seven model predictions (in varying shades of blue) up to 100 timesteps. The consistent proximity of the model trajectories to the environment states across all four dimensions illustrates the stability and predictive accuracy of the model, even in extended temporal scenarios. This demonstrates the model's robustness and its capacity to maintain stability over prolonged periods, which is essential for reliable long-term control in dynamic systems.

5 Discussion

5.1 Overview

In this study, we have successfully developed the Fourier Adaptive Learning for Control (FALCON) in Python, showcasing its utility in turbulent control environments. The core of our work was the implementation of L2, L1, and neural network models for system dynamics, alongside a custom version of the Cross-Entropy Method tailored to the FALCON framework. Through rigorous testing, we demonstrated the ability of these models to handle complexity, ensuring accuracy and stability while avoiding overfitting.

Significant effort was placed on the development of custom evaluation tools that allowed for detailed analysis of the algorithm's performance. The visualization tools and timing mechanisms implemented provided critical insights into the adaptive learning process, revealing the impact of various parameters on system control. The robust software package produced serves as a versatile tool for researchers to explore and implement adaptive control strategies effectively.

Overall, the workload of the project was heavily skewed towards coding and testing, ensuring that the developed models and tools met the stringent requirements of controlling turbulent systems. The resulting software package stands as a versatile and robust framework for future research and practical applications in reinforcement learning and adaptive control.

5.2 Improvements

5.2.1 Addressing Errors in Discrete and Episodic Environments

Future research should delve into the challenges identified in discrete and episodic environments, where the formation of subsequences has been observed to introduce errors. Notably, the phenomenon where the R^2 score decreases with an increase in data points in these cases calls for a comprehensive analysis to identify the underlying causes and potential remedies. This anomaly suggests that the current model may not fully capture the environment's dynamics or that overfitting occurs as the data volume increases.

5.2.2 Implementation in Partially Observable Environments

Our next steps will focus on implementing FALCON within partially observable environments, with the ultimate goal of applying it to the NASA hump model for active flow control. As an intermediate step, we plan to utilize the lunar lander environment to refine the algorithm's robustness in simpler settings. Gradually, we aim to transition to basic Navier-Stokes environments, progressively building up the complexity of the scenarios to ensure a solid foundation for the final implementation on the NASA hump model.

5.2.3 Optimization for High-Performance Computing Environments

Another crucial development will be the translation of the FALCON algorithm into C++ with a Python wrapper to offer a high-level interface. This modification will undoubtedly enhance performance, especially when parallelized across heterogeneous High-Performance Computing (HPC) environments. Reducing latency is of paramount importance for real-time applications, and exploiting the capabilities of HPC resources could significantly accelerate computation and improve efficiency.

he results presented in this study elucidate the robustness of the FALCON framework, underpinned by the Fourier series representation and sophisticated optimization techniques. The figures underscore the efficacy of the algorithm across various dimensions of system control, from parameter tuning to real-world trajectory stability.

5.2.4 FALCON Robustness

Figure 1 illustrates the bifurcation of the FALCON framework into the Warm-Up and Adaptive Control phases. The Warm-Up phase is crucial for establishing a preliminary understanding of the system's dynamics, which is instrumental in constructing a viable

Fourier basis. The Adaptive Control phase leverages this basis for real-time decision-making, reflecting the algorithm's capacity to adapt to the evolving dynamics of turbulent environments.We acheived a 143.56 Hz refresh rate (6 ms environment interaction) which demonstrates the capability to meet operational inference times.

Figure 2 and Figure 3 collectively demonstrate the algorithm's expressiveness and precision. The Continuous Pendulum Environment Visualization provides an empirical testament to the model's fidelity in capturing the essential frequencies that govern the system's behavior. The Grid Search Performance Metrics further substantiate the model's accuracy, where the dimensionality of the Fourier basis, D, and the horizon, h, along with the regularization parameter, λ, are tuned for optimal performance. Higher values of D and h, paired with an appropriate λ, result in a model that balances complexity and generalization, capable of accommodating the noise intrinsic to turbulent systems.

The interplay between expressiveness and stability is further highlighted in Figure 4, where the model's weights, learned over time, suggest a maturing understanding of the system's dynamics. The positive correlation between dataset size and R^2 score signifies an improved performance, while the fluctuations in Mean Squared Error (MSE) imply the nuanced challenges in precision.

Figure 5 is particularly telling of the model's stability. The consistent alignment of the model predictions with the actual environmental states, even over an extended horizon of 100 timesteps, is indicative of the model's robustness. The ability to maintain stable predictions over long durations is imperative for practical applications, where persistent performance is a non-negotiable requirement.

Together, these figures paint a comprehensive picture of the FALCON algorithm's capabilities. Through meticulous tuning of the Fourier basis dimensionality and regularization, coupled with the fine-grained optimization afforded by the Cross-Entropy Method, FALCON emerges as a powerful tool for adaptive control. Its proficiency in learning from high-dimensional data, managing complex matrix operations, and adhering to stability constraints ensures that FALCON is well-positioned for deployment in safety-critical applications, such as active flow control over airfoils with plasma actuators.

Future work will aim to harness these capabilities further, exploring their implications for NASA engineers concerned with the safety and reliability of deploying reinforcement learning models in the field. The development trajectory will progress towards partially observable environments, with the intent to eventually apply FALCON to the NASA hump model, thus bridging the gap between theoretical constructs and practical, real-world utility.

5.3 Future Work: Stability Guarantees and Their Implications for Active Flow Control

5.3.1 Stability Guarantees through Krasovskii's Method and Lyapunov Function Construction

As we advance the capabilities of the FALCON algorithm, one of the critical areas for future research lies in the realm of stability guarantees. Stability, particularly in control systems, is foundational to ensuring that the system's behavior remains predictable and reliable over time. Krasovskii's method and Lyapunov function construction are two pillars in control theory that provide systematic approaches to verify stability. Below we

provide a quick introduction to the method and discuss it's potential use in the future development of FALCON for NASA.

Krasovskii's Method in Control Systems: Krasovskii's method involves constructing a Lyapunov function, a scalar function that provides a measure of system energy or potential. The Lyapunov function $V(x)$ is crafted to be positive definite and to decrease along system trajectories, thus indicating system stability. For a discrete-time nonlinear system $x_{t+1} = f(x_t, u_t)$, Krasovskii's Lyapunov function can be formulated as [4]:

$$V(x) = (x - f(x, u))^T M (x - f(x, u)) \tag{8}$$

[**s**] where M is a positive definite matrix. The system is said to be stable if there exists a matrix M such that $G(x, \theta)^T M G(x, \theta) - M \leq -\bar{\epsilon} I$ for some $\bar{\epsilon} > 0$, with $G(x, \theta)$ being the Jacobian of the closed-loop system.

Lyapunov Function Construction: The construction of a Lyapunov function $V(x)$ is central to proving stability. The function must satisfy:

- $V(x) > 0$ for all $x \neq 0$, and $V(0) = 0$ (positive definiteness).
- $V(x_{t+1}) < V(x_t)$ for all x_t (decrease along system trajectories).

In essence, the Lyapunov function acts as an energy measure that decreases over time, leading the system towards a stable state or set of states.

5.3.2 Implications for Safety-Critical Applications

The implications of these stability guarantees are profound, particularly for safety-critical applications such as active flow control over airfoils. The ability to construct a Lyapunov function provides a mathematical guarantee that the system will not exhibit unwanted or dangerous behavior, a significant concern for NASA engineers and others in the field of aerospace engineering.

For applications in active flow control using plasma actuators, the stability guarantees mean that the controllers designed using the FALCON algorithm can be proven to bring the system to a desired operating point without the risk of divergence or oscillatory instabilities. This assurance is crucial when deploying RL models in real-world, safety-critical environments where unpredictability can lead to catastrophic outcomes.

Research Directions for Active Flow Control: Future work will explore the integration of Krasovskii's method into the FALCON framework for designing controllers that not only perform well but also provide stability guarantees. This involves:

- Developing models that accurately capture the dynamics of airflows over airfoils, including the effects of plasma actuators.
- Constructing Lyapunov functions tailored to the specific dynamics of these models, ensuring that the controllers maintain stability even in the presence of disturbances or modeling errors.
- Verifying these stability properties both in simulation and through rigorous mathematical proofs, providing the necessary confidence for field deployment.

In summary, future advancements in FALCON's algorithm will not only focus on performance but also on ensuring the stability of the control system. This dual focus will help bridge the gap between theoretical research and practical applications, ultimately leading to the deployment of reliable and safe RL models in the field of aerospace engineering.

References

[1] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (1989), pp. 303–314.

[2] Xiaowei Jin et al. "NSFnets (Navier-Stokes Flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations". In: *Journal of Computational Physics* 426 (Feb. 2021). arXiv:2003.06496 [physics], p. 109951. ISSN: 00219991. DOI: 10.1016/j.jcp.2020.109951. URL: http://arxiv.org/abs/2003.06496 (visited on 09/18/2023).

[3] G. E. Karniadakis et al. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440.

[4] Ali Sahin Lale. "Learning and Control of Dynamical Systems". Defended May 3rd, 2023. PhD thesis. Pasadena, California: California Institute of Technology, 2023. URL: URL.

[5] Xin-Yang Liu and Jian-Xun Wang. "Physics-informed Dyna-Style Model-Based Deep Reinforcement Learning for Dynamic Control". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 477.2255 (Nov. 2021). arXiv:2108.00128 [cs, math], p. 20210618. ISSN: 1364-5021, 1471-2946. DOI: 10.1098/rspa.2021.0618. URL: http://arxiv.org/abs/2108.00128 (visited on 09/19/2023).

[6] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

A Detailed Breakdown of the FALCON Algorithm Implementation

This appendix provides a comprehensive breakdown of the FALCON algorithm, implemented in Python. The code is dissected into its fundamental components, detailing the functionality and the mathematical concepts underlying each segment.

A.1 Introduction

FALCON (Fourier Approximation for Learning CONtrollers) is an algorithm that leverages the Cross-Entropy Method (CEM) for planning in conjunction with Fourier basis functions for state representation. The algorithm is designed to operate in environments with continuous state and action spaces, as commonly found in reinforcement learning problems.

A.2 Implementation Overview

The Python implementation of FALCON comprises several key classes and methods, each serving distinct roles in the algorithm's workflow. The primary components include:

- **CEM Interface**: An abstract base class for different implementations of the Cross-Entropy Method.
- **DefaultCEM**: A null implementation of the CEM interface.
- **FALCON Class**: The main class encapsulating the FALCON algorithm.

A.2.1 CEM Interface

The CEM interface serves as a template for various implementations of the Cross-Entropy Method. It defines an abstract method, `plan`, which is intended to be overridden in derived classes.

```
from abc import ABC, abstractmethod

class CEMInterface(ABC):
    @abstractmethod
    def plan(self, model, state, h, cost_functions):
        pass
```

The `plan` method is designed to receive a model function, the current state of the environment, a planning horizon `h`, and a sequence of cost functions. The method should return an action or sequence of actions based on the CEM.

A.2.2 DefaultCEM

The **DefaultCEM** class is a null implementation of the CEM interface. It serves as a placeholder, providing no functional implementation of the planning method.

```
class DefaultCEM(CEMInterface):
    def plan(self, model, state, h, cost_functions):
        return None
```

A.2.3 FALCON Class

The **FALCON** class is the core of the implementation, where the majority of the algorithm's functionality resides. This class is initialized with several parameters, including the environment, dimensionality of the Fourier basis, regularization parameters, and an optional instance of a CEM implementation.

```
import numpy as np
from collections import deque

class FALCON:
    def __init__(self, env, D=100, lambda_val=0.1, alpha_val=0.1, h=4,
                                        K=1000, warmup_data=None, cem=
                                        None):
        % Initialization of various parameters and data structures
        % [...]
```

A.3 Features of the FALCON Implementation

The FALCON implementation is rich in features, offering flexibility, customization, and detailed analysis capabilities. These features enhance the algorithm's adaptability to various scenarios and provide insights into its performance and behavior. Below are key features of the FALCON implementation:

A.3.1 Multiple Regularization Options

FALCON allows the use of different regularization techniques to estimate the model parameters, catering to varying requirements of the learning task:

- **L2 Regularization (Ridge Regression)**: Implemented in the `l2_sklearn_complete` method, this approach uses Ridge Regression for parameter estimation, balancing the model's complexity and its performance.
- **L1 Regularization (Lasso)**: The implementation structure supports L1 regularization, useful for feature selection and sparse models. Used in `l1_regression_complete`
- **Neural Network Based Learning**: The `l2_tensorflow_complete` method demonstrates the use of a neural network for parameter estimation, offering a non-linear alternative to the linear models.

A.3.2 Timing Analysis

The implementation includes functionality to analyze and record the timing of different operations, particularly during the model evaluation phase. This feature is critical for understanding the computational efficiency of the algorithm, especially in real-time applications:

```
def evaluate\_model(self, num\_sims=1, max\_steps=100, min\_steps=35,
                                        display=True):
    % Model evaluation with timing analysis
    % [...]
```

A.3.3 Flexible Parameter Settings

FALCON's design allows easy modification of various parameters, enabling it to be tailored to specific environments or requirements. Key parameters include the dimension of the Fourier basis (D), regularization parameters (lambda_val, alpha_val), and the planning horizon (h). These can be adjusted during the initialization of the FALCON class.

A.3.4 Trajectory Rollout and Analysis

The algorithm can perform trajectory rollouts based on a given action sequence and initial state. This feature is useful for predicting future states and analyzing the behavior of the system under different action sequences.

```
def rollout\_trajectory(self, action\_sequence, initial\_state, collect
                                        \_timing=False):
    % Rollout of a trajectory based on an action sequence
    % [...]
```

A.3.5 Visualization Tools

FALCON includes various visualization tools, such as plotting the trajectories, Fourier basis, and model performance metrics. These visualizations aid in understanding the algorithm's learning process and the quality of its predictions.

```
def plot\_trajectories(self, env\_states, model\_trajectories):
    % Visualization of trajectories
    % [...]
```

A.3.6 Comprehensive Logging

Throughout the implementation, comprehensive logging is provided, giving insights into the internal workings and states of the algorithm during execution. This feature is invaluable for debugging and performance analysis.

A.4 Conclusion

The FALCON algorithm's Python implementation is characterized by its rich set of features, offering a versatile and powerful tool for reinforcement learning tasks. Its ability to employ different regularization techniques, coupled with extensive parameter customization, timing analysis, visualization tools, and comprehensive logging, makes it a robust and adaptable solution for a wide range of applications in the field of machine learning and control systems.